SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY - INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Computational Science and Engineering

Implementation and Evaluation of Riemann Solvers in ExaHyPE 2

Ludwig Kratzl

SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY - INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Computational Science and Engineering

Implementation and Evaluation of Riemann Solvers in ExaHyPE 2

Implementierung und Evaluierung von Riemann-Lösern in ExaHyPE 2

Author:	Ludwig Kratzl
Supervisor	Univ.-Prof. Dr. Michael Bader
Advisor:	M. Sc. Marc Marot-Lassauzaie
Submission Date:	15/09/2023

I hereby declare that this thesis is entirely the result of my own work except where otherwise indicated. I have only used the resources given in the list of references.

15/09/2023 Ludwig Kratzl

Acknowledgments

I would like to thank Prof. Dr. Michael Bader for again giving me the opportunity to write my thesis in his chair. My sincere thanks also go to my advisor Marc Marot-Lassauzaie for his unlimited support and guidance throughout this work. Last, I want to thank my family and friends, who have always supported and never stopped believing in me in the last five years.

Failure was never an option.

"From this distant vantage point, the Earth might not seem of any particular interest. But for us, it's different. Consider [...] that dot. That's here. That's home. That's us. On it everyone you love, everyone you know, everyone you ever heard of, every human being who ever was [...].

-Carl Sagan

Abstract

Riemann solvers form a crucial component in solving partial differential equations (PDEs) numerically using Discontinuous Galerkin (DG) or Finite Volumes methods, which are invoked extensively often in the final code. This thesis implements HLL-type, exact, and augmented Riemann solvers in the open-source software ExaHyPE 2, an engine to generate simulations for hyperbolic PDEs in first-order formulation. Thereby, the solvers are evaluated and verified with the ADER-DG approach applied to the non-homogeneous shallow water and elastic wave equations in various example problems like dam break, lake at rest, and oscillating lake. The work shows that in this context, the choice of Riemann solver influences properties such as accuracy or well-balancedness. However, due to the high order of convergence, the differences between the solvers are small, in particular, if scenarios with smooth solutions are considered. Furthermore, the necessary adjustments to the solvers to allow for numerically demanding problems like inundation and tsunami simulations are outlined.

Kurzfassung

Riemann-Löser bilden eine wesentliche Komponente in der numerischen Lösung partieller Differentialgleichungen (PDEs) mittels Diskontinuierlicher Galerkin (DG) und Finiter Volumen Verfahren, die im finalen Code sehr häufig aufgerufen wird. Diese Arbeit implementiert HLL-basierte, exakte und erweiterte Riemann-Löser in der Open-Source Software ExaHyPE 2, einer Engine zur Generierung von Simulationen für hyperbolische PDEs erster Ordnung. Dabei werden die Löser mit dem ADER-DG-Ansatz angewandt auf die Flachwasser- und elastische Wellengleichungen mittels Dammbruch- sowie ruhender und oszillierender See-Szenarien evaluiert und verifiziert. Es wird gezeigt, dass die Wahl des Riemann-Lösers in diesem Kontext Einfluss auf Eigenschaften wie Genauigkeit und Stabilität hat. Allerdings sind die Unterschiede in den Lösern aufgrund der hohen Konvergenzordnung klein, insbesondere wenn Gleichungen mit glatten Lösungen betrachtet werden. Darüber hinaus werden die notwendigen Anpassungen an den Lösern für numerisch anspruchsvolle Probleme wie Überschwemmungs- und Tsunami-Simulationen skizziert.

Contents

Part I.

Introduction and Fundamentals

1. Introduction

Earthquakes are some of the most devastating natural disasters worldwide: They can cause enormous damage to human settlements and cost thousands of lives. The earthquake in Turkey and Syria on February 6th 2023, for example, claimed over 50,000 victims and numerous injuries, destroying the livelihood of many people for decades. Another frightening consequence of earthquakes underwater are tsunamis: The sudden displacement of water due to the release of underground tension leads to large-scale tidal waves, posing a threat to any neighboring coastal regions. For instance, the Boxing Day tsunami, caused by the Sumatra-Andaman earthquake in 2004, was one of the most severe since the beginning of recording, with over 200,000 deaths and three affected continents.

A crucial component in mitigating the consequences of future earthquakes are early warning systems, which aim to inform the population in time so that they can leave the danger zone. As a result of the Sumatra earthquake, for example, the GITEWS framework was developed. It allowed for both locating the earthquake's source as well as forecasting where the tsunami waves would hit (sometimes denoted as Run-Up). The underlying model for wave propagation was based on the shallow water equations, a non-linear hyperbolic system of partial differential equations (PDE) to simulate fluid flows where the horizontal length scale is significantly greater than the vertical one. Thus, the mathematics of marine modeling and computational seismology helps to reduce risks for tsunami and earthquake-prone countries.

However, creating and running such simulations is a challenging task for several reasons: On one hand, the sheer size of the considered domain is overwhelming. Tsunamis can extend thousands of kilometers for hours while forming both regions with high activity as well as areas where the ocean is at rest. Therefore, regular Cartesian grids, even used on high-performance machines, are hardly employable. Instead, unstructured meshes together with adaptive mesh refinement are to be preferred. On the other hand, the mathematics and numerical treatment are demanding: Stability requirements imposed by (almost) discontinuous bottom topography, potential shock waves, or inundation scenarios need special care when discretizing space and time. Finite Volumes (FVM) or Discontinuous Galerkin (DG) methods are here the means of choice over traditional approaches like Finite Elements (FEM) or Finite Differences.

ExaHyPE (An Exascale Hyperbolic PDE Engine) is an open-source software that helps generate simulations with such requirements: Instead of writing the code from scratch, it enables source code generation for any hyperbolic system of PDEs in first-order formulation with source terms and non-conservative product, hiding most of the mathematical and computational complexity. The user can choose between DG and FVM as a numerical scheme to solve the equations. Initially funded and completed in 2020, ExaHyPE is currently being rewritten under the name ExaHyPE 2 to improve the code base, allowing for additional features in the future.

One of the key components of the generated code is the so-called Riemann solver. A Riemann solver resolves the discontinuity between two faces arising when employing DG or FVM, or in other words, it solves the Riemann problem between these faces. This solver is called tremendously often throughout the simulation, as any two neighboring grid cells share at least one such discontinuity per time step. Furthermore, the solver can be realized using different goals and strategies in mind, for example, one might implement it exactly and approximately or completely and incompletely - Consequently, they play an important role in the simulation's performance and accuracy. Even though researching Riemann solvers isolated or in the context of FVM has a long tradition, their performance embedded in ExaHyPE 2 has not been investigated yet. This thesis aims to close this gap and implement and evaluate various Riemann solvers in the newer framework. The work is structured as follows:

Chapter 2 introduces the topic by first deriving hyperbolic PDEs and describing analytical solutions. Afterward, numerical methods are discussed, and it is shown where Riemann problems occur in the final schemes, justifying the necessity of using Riemann solvers. Finally, ExaHype 2 and applications are presented.

Chapter 3 discusses common Riemann solvers using the shallow water equations as an example. Besides exact solvers specific to this equation, approximate but universal ones are derived in detail and selected for the rest of this thesis. Subsequently, the solvers are implemented in ExaHyPE 2 and checked for correctness in Chapter 4.

Finally, Chapter 5 compares the Riemann solvers in the context of ExaHyPE 2 with different criteria: First, dam-break scenarios are used to test how well the solvers resolve resulting shock and rarefaction waves in the setting of the homogeneous shallow water equations and DG method. Next, bathymetry is added to the equations, and the capability of the solvers to maintain an ocean at rest is assessed. The modifications to the solvers to deal with dry initial states and inundation scenarios are outlined subsequently, providing the necessary steps to enable tsunami simulations. Finally, the universal solvers are tested with the three-dimensional elastic wave equation to show that the work can be used for other applications too. This work ends with a summary and conclusion and describes potential future work with ExaHyPE 2 and Riemann solvers.

2. Fundamentals

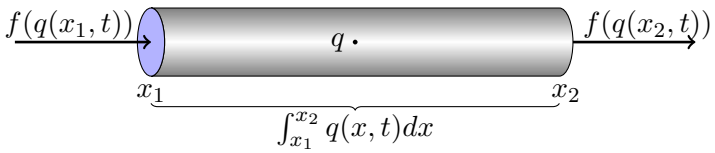
This chapter summarizes the main foundations of this thesis. First, hyperbolic partial differential equations and their solution methods are introduced. Then, the terms Riemann problem and Riemann solver are explained in greater detail using the example of linear equations and the need for them is clarified with an overview of numerical methods. Finally, ExaHyPE 2 and its applications, which play an extensive role in the rest of the work, are briefly addressed.

2.1. Conservation Laws and Hyperbolic PDEs

The physical basis of many hyperbolic PDEs form so-called conservation laws: They are characterized such that the size of a physical quantity does not change over time. For instance, the conservation of mass states that mass can not be created or destroyed within a closed system. In the following, conservation laws and hyperbolic PDEs together with analytical solution approaches are introduced, forming a solid basis for understanding Riemann solvers.

2.1.1. Derivation

An often-found intuitive example in literature (e.g. [30, chap. 1]) for deriving conservation laws is advection through the cross-section $[x_1, x_2]$ of a very thin (i.e., one-dimensional) pipe, which is summarized in the following:



Here, the mathematical description of the considered quantity is a scalar-valued function $q(x,t)$ with its overall sum expressed by the integral $\int_{x_1}^{x_2} q(x,t)dx$. On the contrary, the rate of change is given by the inflow and outflow $f(q)$ (flux) at both ends of the pipe, modeled using the time derivative:

$$\frac{\partial}{\partial t}\int_{x_1}^{x_2} q(x,t)dx = f(q(x_1,t)) - f(q(x_2,t)) = -[f(q(x,t))]_{x_1}^{x_2} = -\int_{x_1}^{x_2} \frac{\partial}{\partial x} f(q(x,t))dx \quad (2.1)$$

If assuming $q \in C^1(\mathbb{R}\times\mathbb{R},\mathbb{R})$, $f \in C^1(\mathbb{R},\mathbb{R})$, integral and partial derivative are exchangeable, and with that, the equation can be rewritten to

$$\int_{x_1}^{x_2} \frac{\partial}{\partial t} q(x,t) + \frac{\partial}{\partial x} f(q(x,t)) = 0, \quad (2.2)$$

yielding the integral-form of a one-dimensional conservation law. Since smoothness was assumed, this integral equation is fulfilled if

$$\frac{\partial}{\partial t}q(x,t) + \frac{\partial}{\partial x}f(q(x,t)) = 0, \tag{2.3}$$

which is denoted as the differential form of the conservation law. Expanding the derivative under the assumption of smoothness creates a further useful version of the PDE:

$$\frac{\partial}{\partial t}q(x,t) + f'(q(x,t))\frac{\partial}{\partial x}q = 0, \tag{2.4}$$

where $f'(q)$ denotes the Jacobian matrix of the function. This version is called quasi-linear form of the PDE and is essential to understanding non-linear Riemann problems, as will be demonstrated in Chapter 3.

The example is transferable to two dimensions as well [30, chap. 18] by introducing two flux functions $f(q)$ and $g(q)$ in x and y direction and the double integral $\int \int_\Omega q(x,y,t)dx\,dy$ over the considered domain Ω. The rate of change is then expressible by a flux in normal direction n along the domain's boundary $\partial\Omega$:

$$\frac{\partial}{\partial t}\int\int_\Omega q(x,y,t)dx\,dy = -\int_{\partial\Omega}(n_x, n_y)\begin{pmatrix} f(q) \\ g(q)\end{pmatrix} ds. \tag{2.5}$$

Similar assumptions on smoothness as in the one-dimensional case then lead to the integral form in two dimensions:

$$\int\int_\Omega \frac{\partial}{\partial t}q(x,y,t) + \frac{\partial}{\partial x}f(q) + \frac{\partial}{\partial y}g(q)dx\,dy = 0. \tag{2.6}$$

Dropping function arguments and using subscripts for partial derivatives lead to the compact differential form

$$q_t + f(q)_x + g(q)_y = 0, \tag{2.7}$$

which is even further generalizable by setting f to a vector-valued functions with component-wise fluxes $f_1(q), ..., f_n(q)$ in n spatial directions and using the divergence operator $\nabla \cdot f = \sum_{i=1}^{n} \frac{\partial f_i}{\partial x_i}$:

$$q_t + \nabla \cdot f(q) = 0. \tag{2.8}$$

An even more abstract notation of Equation 2.8 takes an additional source term $\psi(q)$ such as gravity or friction:

$$q_t + \nabla \cdot f(q) = \psi(q). \tag{2.9}$$

This equation is denoted as balance law since flux function and source term need to cancel out and require additional care when solving the equation numerically or analytically, as demonstrated in Section 3.3.

2.1.2. Analytical Solutions

For some of the equations defined before, there exists an analytical solution [30, chap. 2]. For example, if assuming f in 2.3 to be a constant-coefficient linear function, i.e., $f(q) = vq$, the advection equation $q_t + vq_x = 0$ can be obtained. Its solution is just a translation with speed v of the initial distribution $q(x, 0) = \tilde{q}(x)$ of the considered quantity in time without change of shape:

$$q(x, t) = \tilde{q}(x - vt). \tag{2.10}$$

If f is equal to a diagonalizable constant-coefficient matrix $A \in \mathbb{R}^{m\times m}$ with real eigenvalues, i.e., $f(q) = Aq = R\Lambda R^{-1}q$ exists, the equation can be rewritten to a diagonal system by defining $w = R^{-1}q$:

$$q_t + Aq_x = 0 \Rightarrow q_t + (R\Lambda R^{-1})q_x = 0 \Rightarrow R^{-1}q_t + \Lambda w_x = 0 \Rightarrow w_t + \Lambda w_x = 0. \tag{2.11}$$

Each of the arising advection equations can then be solved independently using Equation 2.10. Thus, the solution of a conservation law of the form

$$\begin{cases} q_t(x, t) + Aq_x(x, t) & = 0 \\ q(x, 0) & = \tilde{q}(x) \end{cases} \tag{2.12}$$

can be viewed as a superposition of m independent waves, where the p-th wave has the form $\tilde{w}_p(x) = (R^{-1}\tilde{q})_p(x)$ and advects with velocity λ_p without change of shape:

$$q(x, t) = \sum_{p=1}^{m} \tilde{w}_p(x - \lambda_p t)r_p. \tag{2.13}$$

The curves $X(t) = x_0 + \lambda_p t$ are called p-characteristics, and the scalar coefficients $w_p(x, t)$ characteristic variables. The solution method is known as method of characteristics. It is generalizable to other types of PDES, for example, the non-homogeneous scalar transport equation $q_t + f(x)q_x = b(x)$, which is not further discussed here.

To find Solution 2.13 from above, it was assumed that the diagonalization $A = R\Lambda R^{-1}$ exists and that all eigenvalues are real. This property is also called hyperbolicity. For the two-dimensional homogeneous version $q_t+Aq_x+Bq_y = 0$ hyperbolicity is fulfilled if both matrices $A, B \in \mathbb{R}^{m\times m}$ are diagonalizable with real values and so is any linear combination $\check{A} = n_xA+n_yB$ of them for any choice of $n = (n_x, n_y) \in \mathbb{R}^2$. Intuitively, this transfers the idea of m propagating waves in two spatial directions. If A and B are simultaneously diagonalizable, i.e., they share the same eigenvectors, then the decompositions $A = R\Lambda_x R^{-1}, B = R\Lambda_y R^{-1}$ again yield an analytical solution by defining $w = R^{-1}q$:

$$w_t + \Lambda_x w_x + \Lambda_y w_y = 0. \tag{2.14}$$

Using these solution approaches allows the implementation of linear or linearized Riemann solvers. To begin with, the term Riemann problem is explained in more detail.

2.2. The Riemann Problem

Figure 2.1 shows a special type of initial value for a one-dimensional scalar-valued PDE: Two constant states q_l (left) and q_r (right) are separated by a discontinuity at $x = 0$. A conservation law with this initial value is called a Riemann problem. Mathematically, it is described as a Cauchy problem of the form

$$q_t + f(q)_x = 0, \tag{2.15}$$

$$q(x, 0) = \tilde{q}(x) = \begin{cases} q_l, & x < 0 \\ q_r, & x > 0 \end{cases}, \tag{2.16}$$

where $q_l, q_r \in \mathbb{R}$ and q, f might also be vector-valued. A Riemann solver tries to answer how this discontinuity propagates and changes over time. Riemann problems arise in various ways. On the one hand, it may make sense to use them directly as a model. One example here is the dam break scenario, where the sudden collapse of a dam results in the collision of two roughly constant but different water levels. This problem is used in Section 5.1.1 as a test problem for the evaluation of various Riemann solvers. Another model is acoustics in a layered medium, where the propagation of sound waves through a material with two stripes of constant density is investigated. The Riemann problem then emerges in the transition of the two densities. The by far larger application, however, is the numerical treatment of hyperbolic conservation laws. Here, the Riemann problem originates from the spatial discretization when employing Discontinuous Galerkin or Finite Volume methods, which is further discussed in Section 2.3. Before outlining this, the next part describes an analytical strategy for solving simple Riemann problems first, which helps understand more complex examples.

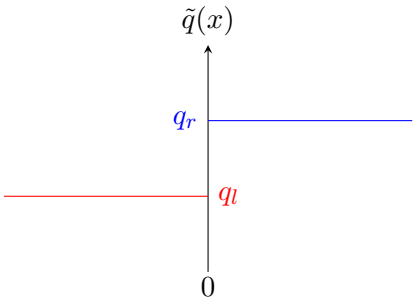


Figure 2.1.: Initial value of a one-dimensional scalar PDE. At $t = 0$, two constant values q_l and q_r are separated by a single discontinuity. A conservation law together with this initial value is called a Riemann problem. A Riemann solver answers how this discontinuity evolves.

2.2.1. An Analytical Solution

The previous section introduced the scalar-valued one-dimensional advection equation

$$q_t + vq_x = 0, v \in \mathbb{R}. \tag{2.17}$$

Together with the initial value from 2.16, this yields the simplest non-trivial Riemann problem. To solve the equation, the solution procedure for constant-coefficient hyperbolic PDEs from Section 2.1.2 can be applied. Here, the constant coefficient matrix A is just the scalar value v which is, per assumption, real-valued. Thus, the eigendecomposition reads $R = 1, \Lambda = v, R^{-1} = 1$, and, therefore, the solution is given by Equation 2.18.

$$q(x,t) = \sum_{p=1}^{m} \tilde{w}_p(x - \lambda_p t) r_p = \tilde{q}(x - vt) = \begin{cases} q_l, & x - vt < 0 \\ q_r, & x - vt > 0 \end{cases}. \tag{2.18}$$

Subsequently, one can conclude that the discontinuity translates at speed v along the characteristic $x - vt$ as expected from the previous investigations.

An often seen visualization of the analytical solution to Riemann problems is the x/t-plane, as shown for example for 2.18 for positive v in Figure 2.2.1. This diagram should be understood as follows: the black intersecting line marks the solution q at a fixed time t (here: $t = 1.5$) over the whole x-dimension. In the left (red) region, the value of $q(x,t)$ equals q_l, whereas within the right (blue) part $q(x,t)$ is q_r. The discontinuity between the two values is located where the x point intersects the purple line (in the figure marked with a dotted line).

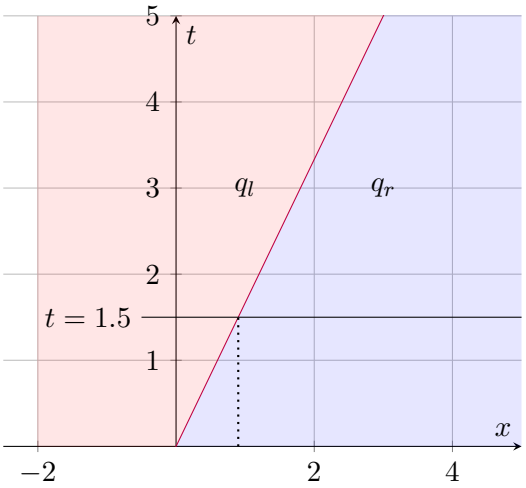


Figure 2.2: Analytical solution for an example Riemann problem with Equation 2.17 for positive $v \in \mathbb{R}$. The discontinuity advects along the characteristic $X(t) = x - vt$ marked in purple. The red area has the value q_l, while the blue are has q_r. The dotted line marks the x position of the jump at fixed $t = 1.5$.

A generalization of this solution idea to a one-dimensional system of m equations is straightforward. The Riemann problem extends to

$$q_t + Aq_x = 0, A \in \mathbb{R}^{m \times m}, \tag{2.19}$$

$$q(0,x) = \tilde{q}(x) = \begin{cases} q_l, & x < 0 \\ q_r, & x > 0 \end{cases}. \tag{2.20}$$

As hyperbolicity is assumed, the decomposition $A = R\Lambda R^{-1}$ with real eigenvalues exists, and thus the constant initial states q_l and q_r are decomposable into

$$q_l = \sum_{p=1}^{m} (w_l)_p r_p, q_r = \sum_{p=1}^{m} (w_r)_p r_p \tag{2.21}$$

with coefficients $(w_k)_p \in \mathbb{R}, k \in \{l, r\}$, as the eigenvectors $r_p \in \mathbb{R}^m$ form a basis of $\mathbb{R}^m$. Then, the p-th initial condition from 2.20 reads

$$\tilde{q}_p(x) = \begin{cases} (w_l)_p, & x_p < 0 \\ (w_r)_p, & x_p > 0 \end{cases} \tag{2.22}$$

And with that, the solution can be written into a sum of left and right-going waves

$$q(x,t) = \sum_{p|\lambda_p < \frac{x}{t}} (w_r)_p r_p + \sum_{p|\lambda_p > \frac{x}{t}} (w_l)_p r_p. \tag{2.23}$$

To make this essential solution concept clearer, example 3.1 (a) from LeVeque's book [30] is outlined in the following. The Riemann problem is given by

$$q_t + Aq_x = 0, A = \begin{pmatrix} 0 & 4 \\ 1 & 0 \end{pmatrix}, q_l = (0,1)^T, q_r = (1,1)^T. \tag{2.24}$$

It is easy to verify that the system is hyperbolic and its diagonalization $A = R\Lambda R^{-1}$ with normed eigenvectors is

$$R = \frac{1}{\sqrt{5}} \begin{pmatrix} 2 & -2 \\ 1 & 1 \end{pmatrix} = (r_1|r_2), \Lambda = \begin{pmatrix} 2 & 0 \\ 0 & -2 \end{pmatrix}, R^{-1} = \frac{\sqrt{5}}{4} \begin{pmatrix} 1 & 2 \\ -1 & 2 \end{pmatrix}. \tag{2.25}$$

To get the decomposition for $q_l = (w_l)_1 r_1 + (w_l)_2 r_2, q_r = (w_r)_1 r_1 + (w_r)_2 r_2$ one needs to solve the linear systems $Rw_l = q_l$ and $Rw_r = q_r$, respectively, yielding

$$w_l = R^{-1} q_l = \frac{\sqrt{5}}{2} \begin{pmatrix} 1 \\ 1 \end{pmatrix}, w_r = R^{-1} q_r = \frac{\sqrt{5}}{4} \begin{pmatrix} 3 \\ 1 \end{pmatrix}. \tag{2.26}$$

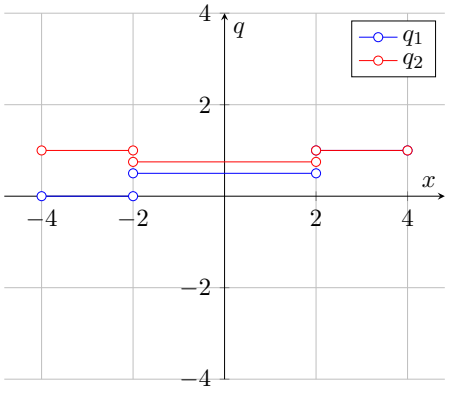
The components of w thus are Riemann problems with the following modified initial conditions:

$$\tilde{w}_1(x) = \begin{cases} \frac{\sqrt{5}}{2}, & x < 0 \\ \frac{3\sqrt{5}}{4}, & x > 0 \end{cases}, \tilde{w}_2(x) = \begin{cases} \frac{\sqrt{5}}{2}, & x < 0 \\ \frac{\sqrt{5}}{4}, & x > 0 \end{cases}. \tag{2.27}$$

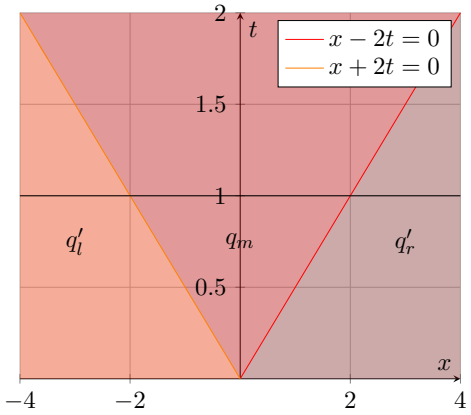
And finally, the solution of the Riemann problem is given in Equation 2.28.

$$q(x,t) = \tilde{w}_1(x - \lambda_1 t) r_1 + \tilde{w}_2(x - \lambda_2 t) r_2 = \tilde{w}_1(x - 2t) \frac{1}{\sqrt{5}} \begin{pmatrix} 2 \\ 1 \end{pmatrix} + \tilde{w}_2(x + 2t) \frac{1}{\sqrt{5}} \begin{pmatrix} -2 \\ 1 \end{pmatrix}. \tag{2.28}$$

Figure 2.3a visualizes the solution q at $t = 1$ in q/x plane: Each component has two shock waves that separate three constant regions. On the contrary, Figure 2.3b shows q_2 in x/t plane. Here, each wedge marks the transition from one shock wave to another. The black intersecting line marks $t = 1$, where the results are shown in the left figure. Again, one can interpret that the Riemann problem describes a superposition of discrete waves (here: two) that propagate over time.



(a) Visualization in x/q plane at $t = 1$.



(b) Visualization in x/t plane for the second component.

Figure 2.3.: Visualization of the one-dimensional system of Riemann problems with two components taken from LeVeque [30], first example 3.1. Left: visualization in x/q plane for both components at $t = 1$. Right: visualization in x/t plane for the second component of q. q_l', q_r' represent the modified variants of q_l, q_r resulting from the decomposition of the initial values. q_m is the middle state separating the two constant states using two characteristics $x - 2t$ and $x + 2t$

Carrying the idea of the example further, it can be concluded that a linear Riemann problem of size m has m characteristics and $m + 1$ areas with constant values, i.e., as one crosses the p-th characteristic, $X_p(t) = x - \lambda_p t$ becomes zero and therefore, the value changes from $(q_l)_p$ to $(q_r)_p$ (or vice versa). In other words, the solution jumps as shock wave from the left value $(w_l)_p r_p$ to the right value $(w_r)_p r_p$ by a scalar value α_p:

$$((w_l)_p - (w_r)_p) r_p = \alpha_p r_p. \tag{2.29}$$

The resulting vector is still an eigenvector of A, leading to the Rankine-Hugoniot jump condition for linear Riemann problems: The difference $(w_l - w_r)$ is an eigenvector, and the velocities are eigenvalues of A. For arbitrary given q_l, q_r, this is not always true. Subsequently, the Riemann problem can be considered as the approach to split $q_r - q_l$ into jumps that fulfill the condition. Using this fact leads to a different solution strategy: Decompose the jump in the initial values by solving

$$q_r - q_l = R\alpha \Rightarrow \alpha = R^{-1}(q_r - q_l), \tag{2.30}$$

where $\alpha_p r_p$ is the jump in the p-th discontinuity and express the solution by either modifying the left data by all left-going waves or the right data by all right-going ones:

$$q(x,t) = q_l + \sum_{p:\lambda_p < \frac{x}{t}} \alpha_p r_p = q_r - \sum_{p:\lambda_p \geq \frac{x}{t}} \alpha_p r_p. \tag{2.31}$$

2.2.2. Generalized Riemann Problems

The previous section derived a general analytical solution strategy for a constant-coefficient linear Riemann problem. However, there are several generalizations that have not been covered, yet. For example, the flux function may be a nonlinear or spatially varying system, i.e., the Riemann problem now takes the form

$$q_t + f(x,q)_x = 0, \tag{2.32}$$

$$q(0,x) = \tilde{q}(x) = \begin{cases} q_l, & x < 0 \\ q_r, & x > 0 \end{cases}. \tag{2.33}$$

An even more abstract equation is stated by Toro [43, chap. 19]. He discusses a non–linear system of hyperbolic balance laws

$$q_t + f(x,q)_x = b(q), \tag{2.34}$$

$$q(0,x) = \tilde{q}(x) = \begin{cases} q_l(x), & x < 0 \\ q_r(x), & x > 0 \end{cases}, \tag{2.35}$$

summarized under the name GRP_K (generalized Riemann Problem of order K) where next to the additional source term the initial states are now introduced as k-times non-trivial continuously differentiable functions in spatial direction instead of scalar values. In that case, GRP_0 is the classical Riemann problem from Equation 2.16. Additionally, Riemann problems in more than one spatial direction have not been described at all. However, as it turns out, for the main field of application of Riemann solvers in the numerical solution of hyperbolic PDEs, it is not necessary to be able to solve these problems explicitly. In fact, many of these equations can be effectively reduced to a classical Riemann problem in one spatial direction, maybe nonlinear or with spatially varying flux-function, by either splitting in each direction or clever reformulation. Then, approximate or exact Riemann solvers based on the solution ideas of one-dimensional linear systems can be employed to complete the numerical scheme. Hence, the next section illuminates how exactly (generalized) Riemann problems arise in the numerical solution and how they are relatable to classical formulations.

2.3. Numerical Treatment of Hyperbolic PDEs

In Section 2.1.2, analytical solutions for some types of hyperbolic PDEs have been outlined. However, most hyperbolic PDEs do not have such a solution or are only solvable under certain restrictions. Moreover, there are other types of equations as well that are not necessarily hyperbolic but rather elliptic or parabolic. Therefore, three main categories of numerical solvers have been developed over the past years: The oldest ones are Finite Difference methods, where the differential form of a PDE is approximated by a point-wise discretization in space and advanced by implicit or explicit Runge-Kutta schemes in time. The Finite Volumes method, on the contrary, approximates the integral form of a PDE by dividing the domain into constant cell averages that communicate through fluxes at the cell edges. Lastly, the Finite Element method approaches the variational formulation of the problem by introducing

the so-called weak form of the PDE together with local control elements, which reconstruct the solution with higher-order polynomials. Not all numerical schemes create Riemann problems. Therefore, all of them are summarized in the following, and it is shown why or why not Riemann solvers are necessary. For this purpose, a homogeneous hyperbolic system of the form

$$q_t + f(q)_x = 0 \tag{2.36}$$

is examined as example problem.

2.3.1. Finite Differences

Following Larsson and Thomeé [27, chap. 12] or Toro [43, chap. 5], Finite Differences first divide the domain Ω into a Cartesian grid with discrete points $x_i = i \cdot \Delta x$. Subsequently, the partial derivatives are expanded to

$$f(q)_x = f'(q)q_x, \tag{2.37}$$

under the assumption of smoothness and approximated using backward difference quotients for the differential operator

$$q_x = \frac{\partial q}{\partial x} \approx \frac{q(t, x_i) - q(t, x_{i-1})}{\Delta x}, \tag{2.38}$$

derived by a Taylor expansion for least two times continuously differentiable functions $q(t, x \pm \Delta x)$. Applying similar for the time derivative with $t_n = n \cdot \Delta t$ but using forward differencing

$$q_t = \frac{\partial q}{\partial t} \approx \frac{q(t_{n+1}, x) - q(t_n, x)}{\Delta t} \tag{2.39}$$

and rewriting the equation yields the first-order upwind scheme in two dimensions to solve 2.36

$$Q_i^{n+1} = Q_i^n + \frac{\Delta t}{\Delta x}(f'(Q_i^n)(Q_i^n - Q_{i-1}^n)), \tag{2.40}$$

where $Q_i^n \approx q(t_n, x_i)$. Similar formulations lead to two-dimensional schemes as well. Equation 2.40 can also be seen as Riemann-free numerical formulation, as no Riemann problems have to be solved at all. As Toro points out [43, chap. 5], this yields a suitable scheme when applied to the one-dimensional advection equation. There are many other well-studied Finite Difference schemes that might have better convergence properties (e.g., the Lax-Wendroff scheme), however, they all share certain disadvantages: For example, Finite Differences are limited to structured grids. In addition, boundary conditions have to be handled with care so that they do not destroy the convergence properties of higher-order schemes. Extrapolation when using Neumann boundary conditions, as well as the Shortley-Weller scheme for curvilinear boundaries, are listed as examples. Furthermore, the classical derivative of a discontinuous solution is not defined. In other words, Finite Differences impose several assumptions on the smoothness of the solution, which excludes certain classes of simulations, especially in the context of hyperbolic PDEs.

Another important property that Finite Differences lack without modification is the concept of conservation, which means that the behavior of the underlying conservation law is imitated. Toro here mentions the necessity of any numerical scheme for conservation laws to be conservative [43, chap. 5]: On the one hand, he cites the results of Hou and LeFloch [22], stating that non-conservative schemes do not converge to the physically correct solution in the case of shock waves. On the other hand, he combines this with the results of Lax and Wendroff, according to which convergent schemes converge to the weak solution of the conservation law [28]. As shock waves are an essential part of conservation laws (as it is also shown in Section 3.1), the following focus relies on intrinsic conserving schemes, starting with the Finite Volumes method.

2.3.2. Finite Volumes

The starting point of the Finite Volumes method is the division of the spatial domain into local volumes $\Omega = \bigcup_i \Omega_i$. In one dimension, these are intervals of non-equidistant size $\Omega_i = [x_{i-\frac{1}{2}}, x_{i+\frac{1}{2}}]$. Furthermore, it is assumed that each cell stores an average of the conserved quantity stored in its center at discrete time t_n:

$$Q_i^n \approx \frac{1}{x_{i+\frac{1}{2}} - x_{i-\frac{1}{2}}} \int_{\Omega_i} q(x, t_n) dx. \tag{2.41}$$

The integral form of a conservation law, as found in Equation 2.2, implies that the rate of change is the flux difference at both ends of the cell. Integrating this form from one point t_n to another t_{n+1} gives

$$\int_{\Omega_i} q(x, t_{n+1}) dx - \int_{\Omega_i} q(x, t_n) dx = \int_{t_n}^{t_{n+1}} f(q(x_{i-\frac{1}{2}}, t)) dt - \int_{t_n}^{t_{n+1}} f(q(x_{i+\frac{1}{2}}, t))) dt. \tag{2.42}$$

Inserting the approximation 2.41 into 2.42 and rewriting creates

$$Q_i^{n+1} = Q_i^n + \int_{t_n}^{t_{n+1}} f(q(x_{i-\frac{1}{2}}, t)) dt - \int_{t_n}^{t_{n+1}} f(q(x_{i+\frac{1}{2}}, t))) dt. \tag{2.43}$$

To continue, it is helpful to define an approximation to the flux of the average:

$$F_{i-\frac{1}{2}}^n \approx \frac{1}{t_{n+1} - t_n} \int_{t_n}^{t_{n+1}} f(q(x_{i-\frac{1}{2}, t})) dt. \tag{2.44}$$

Because then, Equation 2.43 takes the well-known form

$$Q_i^{n+1} = Q_i^n - \frac{t_{n+1} - t_n}{x_{n+1} - x_n} (F_{i+\frac{1}{2}}^n - F_{i-\frac{1}{2}}^n) \overset{\text{equidistance}}{=} Q_i^n - \frac{\Delta t}{\Delta x} (F_{i+\frac{1}{2}}^n - F_{i-\frac{1}{2}}^n), \tag{2.45}$$

which Toro additionally defines as conservative method [43, chap. 5]. Though it has not been yet discussed how to obtain the approximation of the flux average, it may be assumed that the value can be reconstructed only using Q^n:

$$F_{i-\frac{1}{2}}^n = \mathcal{F}(Q_{i-1}^n, Q_i^n), F_{i+\frac{1}{2}}^n = \mathcal{F}(Q_i^n, Q_{i-1}^n), \tag{2.46}$$

where $\mathcal{F}$ is also called numerical flux. Since a homogeneous conservation law was considered, the overall change of the averaged quantity from Q_i^n to Q_i^{n+1} can be seen as difference of the values going in - and outside of the cell. However, this value is ambiguous in the above discretization as Figure 2.4 illustrates: At $x_{i+\frac{1}{2}}$, the analytical flux (and thus the modification at the right cell edge) is determinable by either choosing $f(Q_i^n)$ or $f(Q_{i+1}^n)$. Hence, a tempting approach might be to define the numerical flux as the average of the two options:

$$\mathcal{F}(Q_i^n, Q_{i+1}^n) = \frac{1}{2}(f(Q_i^n) + f(Q_{i+1}^n)), \tag{2.47}$$

which, however, turns out to be an unstable choice. A breakthrough contribution to this was performed by Godunov in 1958 [18]: He replaced the time integral 2.44 by the analytical flux calculated using the solution $q^{\downarrow}(Q_i^n, Q_{i+1}^n)$ to the Riemann problem with initial data $q_l = Q_i, q_r = Q_{i+1}$ and the original conservation law:

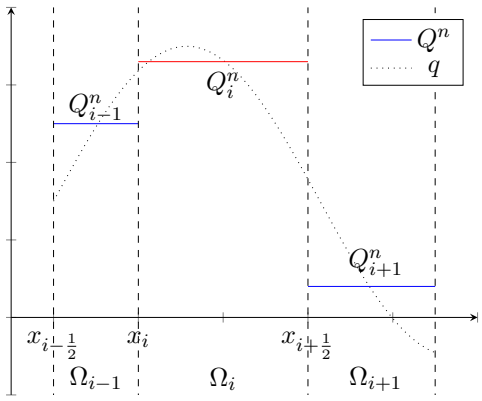


Figure 2.4.: Sketch of a Finite Volumes discretization: The domain is subdivided into local cells Ω_i that store an averaged value Q_i^n of the analytical solution q at time t_n. An essential difficulty is to define the flux at the cell edges, which is solvable by a local Riemann problem.

$$F_{i+\frac{1}{2}}^n \approx \mathcal{F}(Q_i^n, Q_{i+1}^n) = f(q^{\downarrow}(Q_i^n, Q_{i+1}^n)). \tag{2.48}$$

Or in other words, the (numerical) flux along the cell edges is the (analytical) flux computed with the solution of the Riemann problem at $x = 0$. The Riemann problem can thereby be solved exactly (e.g. if dealing with linear constant-coefficient systems, as seen in Section 2.2.1) or suitably approximated. The latter prevents complicated generalized Riemann problems from being solved analytically, which is further discussed in Section 3.2.

LeVeque introduced a different viewpoint on the Finite Volumes method that is often used in the definition of Riemann solvers: The current averaged value Q_i^n is modified from the left cell with right-going waves and from the right cell with left-going ones. He denotes this as fluctuations, yielding a generalized wave-propagation algorithm of the form

$$Q_i^{n+1} = Q_i^n - \frac{\Delta t}{\Delta x}(\mathcal{A}^+ \Delta Q_{i-\frac{1}{2}} + \mathcal{A}^- \Delta Q_{i+\frac{1}{2}}), \tag{2.49}$$

where $\mathcal{A}^+ \Delta Q_{i-\frac{1}{2}}$ denotes the right-going waves from the left cell and $\mathcal{A}^- \Delta Q_{i+\frac{1}{2}}$ the left-going waves from the right one. For a constant-coefficient linear system, for instance, the modifying waves expand to

$$\mathcal{A}^+ \Delta Q_{i-\frac{1}{2}} = \sum_{p=1}^{m} (\lambda_p)^+ \mathcal{W}_{i+\frac{1}{2}}^p, \mathcal{A}^- \Delta Q_{i-\frac{1}{2}} = \sum_{p=1}^{m} (\lambda_p)^- \mathcal{W}_{i+\frac{1}{2}}^p, \tag{2.50}$$

where $\mathcal{W}_{i-\frac{1}{2}}^p := (\alpha_{i-\frac{1}{2}})_p r_p$ are numerical waves denoting the modification to the initial discontinuity of the Riemann problem as introduced in Equation 2.30. $\lambda^+ := \max(\lambda, 0), \lambda^- :=$

$\min(\lambda, 0)$, on the other hand, filter right-going and left-going waves from the cell edges, respectively. LeVeque's contribution provides a more general view of Finite Volumes, which is in particular useful when discussing more complicated PDEs. An illustration of his idea can be found in Figure 2.5

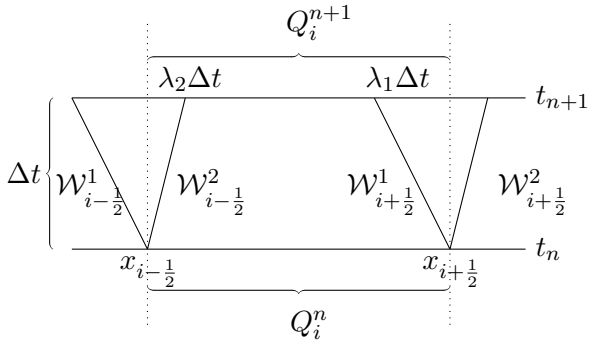


Figure 2.5: Visualization of LeVeque's wave propagation algorithm based on Finite Volumes [30, chap. 4]: An average cell quantity Q_i^n is modified from the left with $\mathcal{W}^2_{i-\frac{1}{2}}$ and from the right with $\mathcal{W}^2_{i-\frac{1}{2}}$ within a single time step $t_{n+1} = t_n + \Delta t$.

When considering two-dimensional equations of the form 2.7, the starting point is now the scheme 2.5 and the average quantities are expressed via the integral $Q^n_{i,j} \approx \frac{1}{\Delta x \Delta y} \int_\Omega q(x, y, t_n)\, dxdy$. Applying similar reformulations as above and assuming a rectangular grid then gives the fully-discrete flux-differencing method, which is very close to the formulation in one dimension:

$$Q^{n+1}_{i,j} = Q_{i,j} - \frac{\Delta t}{\Delta x}(F^n_{i+\frac{1}{2},j} - F^n_{i-\frac{1}{2},j}) - \frac{\Delta t}{\Delta y}(G^n_{i,j+\frac{1}{2}} - G^n_{i,j-\frac{1}{2}}), \tag{2.51}$$

where there are several approaches to deal with the fluxes. For example, Godunov's method implies that the flux can be approximated by solving the Riemann problem in each direction

$$F^n_{i+\frac{1}{2},j} \approx f(q^\downarrow(Q^n_{i,j}, Q^n_{i+1,j})), G^n_{i,j+\frac{1}{2}} \approx g(q^\downarrow(Q^n_{i,j}, Q^n_{i,j+1})). \tag{2.52}$$

Alternatively, one can apply dimensional splitting, where the following equations are solved for each direction:

$$q_t + f(q)_x = 0, q_t + g(q)_y = 0, \tag{2.53}$$

which LeVeque states to be an effective approach [30, chap. 19]. Regardless, in both methods it is sufficient to solve a one-dimensional Riemann problem. Therefore, discussing and implementing Riemann problems for each spatial direction individually is in the context of Finite Volumes is adequate in many situation.

Finite Volumes have several advantages over Finite Differences, even though they are closely related: For example, they are conserving per definition and are not restricted to a particular grid. They allow for adaptive mesh refinement without losing their convergence property, leading to successful use in real-life applications. However, in their original form, they are only of first order and although there exist variations that increase the order of convergence (e.g., by adding correction terms to the wave propagation method in 2.49), this is typically seen as one of this method's main weaknesses. Therefore, Discontinuous Galerkin methods have been developed, which are summarized in the following based on introductory books [27] [21] and papers [6].

2.3.3. Discontinuous Galerkin Methods

The Discontinuous Galerkin (DG) method is sometimes considered a combination of the classical Finite Element (or Continuous Galerkin) method with the Finite Volumes approach. DG approximates the solution to the variational problem, which is derived using the weak form of the conservation law. To obtain the latter, Equation 2.36 is multiplied with a function $\phi \in \Pi^n$ from a suitable space of polynomial test functions of degree n and integrated over the spatial domain:

$$\int_\Omega q_t\phi(x) + f(q)_x\phi(x)dx = 0. \tag{2.54}$$

As in Finite Volumes, the domain is split into non-overlapping subdomains $\bigcup_k \Omega_k = \Omega$ with $\Omega_k = [a_k, b_k]$. The linearity of the integral operator then gives

$$\sum_k \int_{\Omega_k} q_t\phi(x) + f(q)_x\phi(x)dx = 0. \tag{2.55}$$

In the following, only a single element is considered. Using integration by parts subsequently expands 2.55 to

$$\int_{\Omega_k} q_t\phi(x)dx + [f(q)\phi(x)]_{a_k}^{b_k} - \int_{\Omega_k} f(q)\phi_x(x)dx = 0. \tag{2.56}$$

As in the Continuous Galerkin method, the solution space V of q is approximated with the finite-dimensional space of test functions $\Pi^n = \langle\phi_1, ..., \phi_n\rangle$ with basis functions $\phi_i(x)$ where equality holds for $n \to \infty$. Then, q can be approached with a sum of time-dependent coefficients $a_i(t)$:

$$q(x,t) \approx \sum_{i=1}^{n} a_i(t)\phi_i(x). \tag{2.57}$$

Inserting the approximation into 2.56 and reformulations then demands that for all ϕ_j holds:

$$\sum_{i=0}^{n} \frac{\partial a_i}{\partial t}(t) \int_{\Omega_k} \phi_i(x)\phi_j(x)dx + [f(q)\phi_j(x)]_{a_k}^{b_k} - \int_{\Omega_k} f(\sum_{i=0}^{n} a_i(t)\phi_i(x))\frac{\partial\phi_j}{\partial x}dx = 0. \tag{2.58}$$

For instance, if applying the procedure to the advection equation $q_t + vq_x = 0$ from Section 2.1.2 on $[0, 1]$ and using monomial basis polynomials of degree 1, $\phi_0(x) = 1-x, \phi_1(x) = x$, then gives for the first term:

$$\int_0^1 \phi_j(x)\phi_i(x)dx = \begin{pmatrix} \frac{1}{3} & \frac{1}{6} \\ \frac{1}{6} & \frac{1}{3} \end{pmatrix} =: M_{i,j}, \tag{2.59}$$

where M is typically denoted as mass matrix and for the last term

$$\int_0^1 f(\sum_{i=0}^{M} a_i(t)\phi_i(x))\frac{\partial\phi_j}{\partial x}dx \overset{f(q)=vq}{=} v\sum_{i=0}^{M} a_i(t)\int_{\Omega_k} \phi_i(x)\frac{\partial\phi_j}{\partial x}dx \tag{2.60}$$

$$\Rightarrow \int_0^1 \phi_i(x)\frac{\partial\phi_j}{\partial x}dx = \begin{pmatrix} -\frac{1}{2} & -\frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} \end{pmatrix} =: S_{i,j}, \tag{2.61}$$

with S called stiffness matrix. The calculations above using the local interval $[0, 1]$ is thereby not necessarily a restriction, as a mapping ξ_k,

$$\xi_k : [0, 1] \to [a_k, b_k], x \mapsto (b_k - a_k)x + a_k, \tag{2.62}$$

can be used together integration by substitution

$$\int_{a_k}^{b_k} \phi(x)dx = \int_0^1 \phi(\xi_k)(b_k - a_k)dx, \tag{2.63}$$

to transfer the cell-local coordinates to a reference element. With a suitable choice of polynomials, the integrals are precomputable and lead to a stable and efficient matrix-vector product. Furthermore, the mass matrix can end up in diagonal form, making computations efficiently. In practice, Lagrange polynomials together Gauss-Lobatto or Legendre quadrature nodes x_i, x_j are used for that purpose:

$$\phi_i(x) = \Pi_{j=0, j\neq i}^{n} \frac{x - x_j}{x_i - x_j}. \tag{2.64}$$

The latter are the roots of the Legendre polynomials and are non-equidistantly distributed. This avoids the occurrence of oscillations that arise when using an equidistant grid and polynomial interpolation with higher-order (the so-called Runge phenomenon). A visualization of the nodes for order $n = 3$ can be found in Figure 2.6 along $[-1, 1]$. The remaining task is to handle the flux term in the middle of the equation. Here, the essential difference between DG and classical finite elements becomes visible: The latter assumes polynomials whose support spans the global domain continuously and, with that, produces a continuous solution. In the former, the basis functions are assumed to exist only locally and thus to be discontinuous over the full domain. This property is outlined in Figure 2.7. Therefore, an evaluation of f at the edges of the local interval is not defined and consequently, the full semi-discrete system requires special treatment of the face integral, which gets reinterpreted as local Riemann problem:

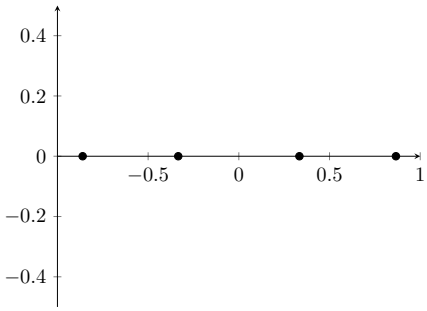
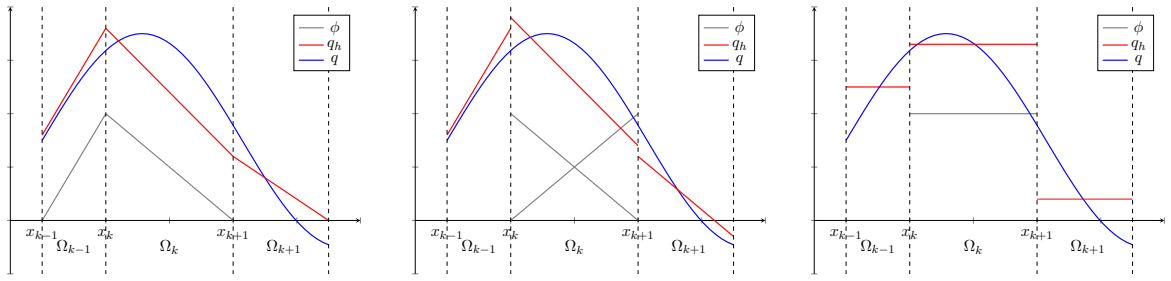


Figure 2.6.: Distribution of the Legendre quadrature nodes for $n = 3$ on $[-1, 1]$. The non-equidistant distribution avoids Runge's phenomenon, however, needs to be kept in mind for point-wise comparisons.

$$[f(q)\phi_j(x)]_{a_k}^{b_k} = f(q^{\downarrow}(a_k))\phi_j(x) - f(q^{\downarrow}(b_k))\phi_j(x) =: f_j. \tag{2.65}$$

Overall, this suggests to study a numerical scheme of the following form for each local element:

$$M\frac{\partial m}{\partial t} + f - Sk = 0 \Rightarrow \frac{\partial m}{\partial t} = M^{-1}(Sk - f), \tag{2.66}$$



(a) Continuous Galerkin (b) Discontinuous Galerkin (c) Finite Volume

Figure 2.7.: Sketch of different numerical solution strategies for an artificial one-dimensional conservation law: Continuous Galerkin (a) uses linear hat functions that span the domain continuously and, therefore, enable continuous reconstruction of the solution. Using DG (b), on the other hand, discontinuities across elements are allowed due to the purely local existence of the basis functions, requiring the solution of a Riemann problem between the elements. Finite Volumes (c) can also be seen as DG method with zero-order polynomials. Here, a Riemann problem has to be solved as well.

where the remaining coefficients are grouped into time-dependent vectors m and k. This semi-discretized system has excellent convergence properties in space: Cockburn et al. show, e.g., that the optimal convergence rate is of order $\mathcal{O}(h^{n+1})$ on a triangular mesh for transport-reaction problems [5]. However, efficiency starts to decrease if the polynomial degree exceeds $n > 8$, as Kronbichler observes [26]. In addition, the increased computational effort could be considered as a disadvantage for this: A higher polynomial degree raises the number of support points and, thus, also the size of the element-wise matrix. Furthermore, discretizing in time needs special care. If using Runge-Kutta methods, the number of stages scales super-linear compared to the order of convergence (also called butcher barrier): For a convergence order ≥ 5 in time there, for example, there are at least six computational stages required. This issue reduces the efficiency of the scheme dramatically or enforces an artificial reduction of the convergence order in time. To overcome this gap, Toro et al. contributed the idea of so-called ADER (Arbitrary DERivative in space and time) schemes [46]. Since a detailed derivative exceeds the scope of this section, a brief summary is given in the following:

First, Equation 2.58 is integrated over a control volume in time $[t_n, t_{n+1}]$ and expanded via integration by parts. Then, a generalized Riemann problem GRP_n is defined on the cell edges with polynomial discontinuous initial data, where the state at each cell interface is expanded using a Taylor series in time. With the help of the Cauchy-Kowalevski product, the time derivatives are subsequently replaced using spatial derivatives, which are calculable using classical Riemann solvers. The result is then a solution of GRP_n, which is a time evolution of the numerical solution. Dealing directly with the Cauchy-Kowalevski product can become tedious or even unfeasible when non-linear hyperbolic PDEs need to be solved, and it is furthermore not able to deal with stiff source terms [9]. Thus, the ADER-DG scheme is implemented using a space-time predictor in practice, which can also be found in the open-source software ExaHyPE 2 [10]. Its usage is sketched in the following section using an example problem definition.

2.4. ExaHyPE 2

ExaHyPE 2 (An Exascale Hyperbolic PDE Engine, second generation)[1] provides an engine to simulate hyperbolic PDEs of the form

$$\frac{\partial Q}{\partial t} + \nabla \cdot F(Q) + \sum_i B(Q) \frac{\partial Q}{\partial x_i} = S(Q), \tag{2.67}$$

where B denotes the non-conservative product and S potential source terms. ExaHyPE 2 aims to reduce the complexity of implementing numerical schemes as derived above by allowing the user to specify their problem into a Python file and then generating the necessary code for HPC platforms. With that, ExaHyPE 2 hides away required computational and numerical details, which are cumbersome to implement. Next to adaptive mesh refinement on block-structured grids, ExaHyPE 2 enables hybrid OpenMP parallelization as well as GPU support using SYCL standard and load balancing with space-filling curves (however, note that by the time of writing many of these features were only available on an experimental basis). Even though main parts of the code are generated, the engine allows the implementation of certain kernels, such as the Riemann solver, by the user to provide tailor-made optimizations. An example problem definition is given in Appendix B.1, which shall be quickly discussed in the following:

- Line 5 defines the project together with the created namespace structure and the name of the resulting executable.
- From 7 to 13, the numerical scheme is defined: Here, ADER-DG with cubic polynomials and the Rusanov approximate Riemann solver in combination with adaptive time stepping has been chosen. The equation has four conserved quantities, i.e., $q = (q_1, q_2, q_3, q_4)^T$.
- Lines 15 and 16 combine project and numerical scheme. As no other options (like source term or non-conservative product) have been specified, the considered equation has the form $q_t + \nabla \cdot f(q) = 0$.
- 18-27 set the simulation parameters: The domain is two-dimensional restricted on the interval $[0, 1]^2$. The simulation runs from $t = 0$ until $t = 1$ and plots an output once every 0.005 time steps have been passed. No periodic boundary conditions are supposed.
- Finally, load balancing is activated, and the path to the necessary Peano installation is set (29-34).

After executing the ExaHyPE 2 specification file, the generated code is almost up and running: The only thing that needs to be implemented is the generated main C++ file, where at least the definition of the flux $\nabla \cdot f(q)$, the maximum eigenvalue of the Jacobian of the flux function $\max |\lambda_p|$ (in this case needed for the Riemann solver and the control of the time step size), and boundary conditions together with initial values have to be incorporated. How to conduct this in particular is described more in detail in Section 4.1. For further information on ExaHyPE 2 in general, refer to the tutorial and documentation provided with the source code repository.

[1] https://gitlab.lrz.de/hpcsoftware/Peano.git

2.5. Applications

Before concluding the chapter, two application examples of ExaHyPE 2 are shown: The shallow water equations for modeling water flows embedded between topography and free surface for simulation of waves found in oceans and atmosphere, and the three-dimensional linear elastic wave equations for seismic wave simulations. Both are used in the subsequent chapters to evaluate and implement Riemann solvers.

2.5.1. Shallow-Water Equations

The shallow water equations are derived from the depth-averaged Navier-Stokes equations by assuming a significantly larger horizontal length scale compared to the vertical one. For further reading on how to derive the model, one can refer to the literature, e.g. LeVeque [30, chap. 13]. Let denote h the water height, hu/hv the momentum in x/y direction, and b the height of the bathymetry. Then, the two-dimensional shallow water equations neglecting Coriolis, frictional, and viscous forces read as

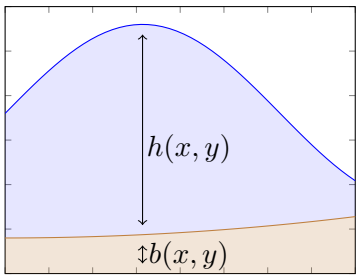


Figure 2.8.: Cross section of the two-dimensional shallow water equations. Water of height h is embedded between the bottom topography b and a free surface.

$$q_t + f(q)_x + g(q)_y = \psi(q), \tag{2.68}$$

with $q = (h, hu, hv)^T$ being the conserved quantity and flux vectors as well as source terms given by

$$f(q) = \begin{pmatrix} hu \\ hu^2 + \frac{1}{2}gh^2 \\ huv \end{pmatrix}, g(q) = \begin{pmatrix} hv \\ huv \\ hv^2 + \frac{1}{2}gh^2 \end{pmatrix}, \psi(q) = \begin{pmatrix} 0 \\ -gh\partial_x b \\ -gh\partial_y b \end{pmatrix}. \tag{2.69}$$

A sketch of this model is shown Figure 2.8. The semi-discrete form $q_t + f'(q)q_x + g'(q)g(q) = \psi(q)$ is given with the Jacobian

$$f'(q) = \begin{pmatrix} 0 & 1 & 0 \\ -u^2 + gh & 2u & 0 \\ -uv & v & u \end{pmatrix} g'(q) = \begin{pmatrix} 0 & 0 & 1 \\ -uv & v & u \\ -v^2 + gh & 0 & 2v \end{pmatrix}. \tag{2.70}$$

For the following work, the eigendecomposition of that form is needed. As e.g. LeVeque shows [30, chap. 21], the eigenvalues and eigenvectors for $f'(q)$ are

$$\lambda_1 = u - \sqrt{gh}, \lambda_2 = u, \lambda_3 = u + \sqrt{gh}, \tag{2.71}$$

$$r_1^{(x)} = \begin{pmatrix} 1 \\ u - \sqrt{gh} \\ v \end{pmatrix}, r_2^{(x)} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}, r_3^{(x)} = \begin{pmatrix} 1 \\ u + \sqrt{gh} \\ v \end{pmatrix}, \tag{2.72}$$

and for $g'(q)$:

$$\lambda_1 = v - \sqrt{gh}, \lambda_2 = v, \lambda_3 = v + \sqrt{gh}, \tag{2.73}$$

$$r_1^{(y)} = \begin{pmatrix} 1 \\ u \\ v - \sqrt{gh} \end{pmatrix}, r_2^{(y)} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, r_3^{(y)} = \begin{pmatrix} 1 \\ u \\ v + \sqrt{gh} \end{pmatrix}. \tag{2.74}$$

Define the matrices $R_1 = (r_1^{(x)}|r_2^{(x)}|r_3^{(x)}), R_2 = (r_1^{(y)}|r_2^{(y)}|r_3^{(y)}), c := \sqrt{gh}$. The inverses are then given by

$$R_1^{-1} = \begin{pmatrix} \frac{u+\sqrt{gh}}{2\sqrt{gh}} & -\frac{1}{2\sqrt{gh}} & 0 \\ -v & 0 & 1 \\ -\frac{u-\sqrt{gh}}{2\sqrt{gh}} & \frac{1}{2\sqrt{gh}} & 0 \end{pmatrix} = \begin{pmatrix} \frac{u+c}{2c} & -\frac{1}{2c} & 0 \\ -v & 0 & 1 \\ -\frac{u-c}{2c} & \frac{1}{2c} & 0 \end{pmatrix}, \tag{2.75}$$

$$R_2^{-1} = \begin{pmatrix} \frac{v+\sqrt{gh}}{2\sqrt{gh}} & 0 & -\frac{1}{2\sqrt{gh}} \\ -u & 1 & 0 \\ -\frac{v-\sqrt{gh}}{2\sqrt{gh}} & 0 & \frac{1}{2\sqrt{gh}} \end{pmatrix} = \begin{pmatrix} \frac{v+c}{2c} & 0 & -\frac{1}{2c} \\ -u & 1 & 0 \\ -\frac{v-c}{2c} & 0 & \frac{1}{2c} \end{pmatrix}, \tag{2.76}$$

which then completes the full eigendecomposition of the system.

2.5.2. Elastic Wave Equation

The three-dimensional elastic wave equations model elastic deformation as, e.g., found in earthquakes. As LeVeque shows [30, chap. 22], they can compactly be written in the form

$$q_t + A(s)q_x + B(s)q_y + C(s)q_z = 0. \tag{2.77}$$

$q(s,t) = (\sigma_{xx}, \sigma_{yy}, \sigma_{zz}, \sigma_{xy}, \sigma_{yz}, \sigma_{xz}, u, v, w)^T$ is a nine-component vector with stresses σ_{ij} and velocities in Cartesian space $s = (x, y, z)^T \in \mathbb{R}^3$. A, B, C are 9×9 matrices incorporating the Lamé parameters λ and σ, describing the relationship between stress and strain in elastic deformation as well as velocities u, v, w (see the next page for a component-wise description for A, B, C) The eigenvalues of A, B, C are:

$$s_1 = -c_p, s_2 = -c_s, s_3 = -c_s, s_4 = 0, s_5 = 0, s_6 = 0, s_7 = c_s, s_8 = c_s, s_9 = c_p, \tag{2.78}$$

with

$$c_p = \sqrt{\frac{\lambda + 2\mu}{\rho}}, c_s = \sqrt{\frac{\mu}{\rho}}, \tag{2.79}$$

taken from literature [11].

$$A = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & -(\lambda+2\mu) & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -\lambda & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -\lambda & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\mu & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\mu \\ -\frac{1}{\rho} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -\frac{1}{\rho} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -\frac{1}{\rho} & 0 & 0 & 0 \end{pmatrix}, B = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\lambda & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -(\lambda+2\mu) & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\lambda & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -\mu & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\mu \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -\frac{1}{\rho} & 0 & 0 & 0 & 0 & 0 \\ 0 & -\frac{1}{\rho} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -\frac{1}{\rho} & 0 & 0 & 0 & 0 \end{pmatrix}, \tag{2.80}$$

$$C = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\lambda \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\lambda \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -(\lambda+2\mu) \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\mu & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -\mu & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -\frac{1}{\rho} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -\frac{1}{\rho} & 0 & 0 & 0 & 0 \\ 0 & 0 & -\frac{1}{\rho} & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

The eigenvectors and their inverses found in the work of Dumbser and Käser [11] are provided for A in Equation 2.81. The decomposition for the other matrices can be found in Appendix A.1 - A.4

$$R_{(x)} = \begin{pmatrix} \lambda+2\mu & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \lambda+2\mu \\ \lambda & 0 & 0 & 0 & 1 & 0 & 0 & 0 & \lambda \\ \lambda & 0 & 0 & 0 & 0 & 1 & 0 & 0 & \lambda \\ 0 & \mu & 0 & 0 & 0 & 0 & 0 & \mu & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \mu & 0 & 0 & 0 & \mu & 0 & 0 \\ c_p & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -c_p \\ 0 & c_s & 0 & 0 & 0 & 0 & 0 & -c_s & 0 \\ 0 & 0 & c_s & 0 & 0 & 0 & -c_s & 0 & 0 \end{pmatrix}, R^{-1}_{(x)} = \begin{pmatrix} \frac{1}{2(\lambda+2\mu)} & 0 & 0 & 0 & 0 & 0 & \frac{1}{2c_p} & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2\mu} & 0 & 0 & 0 & \frac{1}{2c_s} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{2\mu} & 0 & 0 & \frac{1}{2c_s} \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ -\frac{\lambda}{\lambda+2\mu} & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -\frac{\lambda}{\lambda+2\mu} & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{2\mu} & 0 & 0 & -\frac{1}{2c_s} \\ 0 & 0 & 0 & \frac{1}{2\mu} & 0 & 0 & 0 & -\frac{1}{2c_s} & 0 \\ \frac{1}{2(\lambda+2\mu)} & 0 & 0 & 0 & 0 & 0 & -\frac{1}{2c_p} & 0 & 0 \end{pmatrix}. \tag{2.81}$$

In the next chapter, a step back is taken, and Riemann solvers for these applications are discussed.

Part II.

Riemann Solvers and Implementation

3. Riemann Solvers

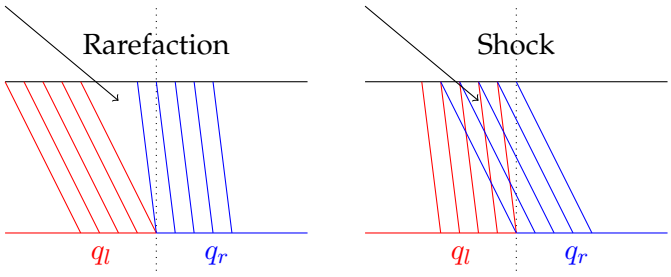
The previous part of this thesis showed that conservation and balance laws can be used to model frequent physical phenomena. Typically expressed via systems of hyperbolic PDEs, their numerical discretization using Finite Volumes or Discontinuous Galerkin methods drives the need to solve one or many Rieman problems at cell interfaces, a hyperbolic PDE with a step function as initial data. The problems can be solved analytically in some cases, for example when considering constant-coefficient equations. ExaHyPE 2 is an engine to generate code for simulating hyperbolic PDEs in first-order formulation, such as the two-dimensional shallow water and the three-dimensional linear elastic wave equation. In the following, different strategies to solve a Riemann problem are presented, and implementation details are highlighted. Here, the first sections deal exclusively with the shallow water equations as an example of a non-linear multi-dimensional system. Initially, the derivations are performed using the one-dimensional, homogeneous version:

$$q_t + f(q)_x = 0, f(q) = \begin{pmatrix} hu \\ hu^2 + \frac{1}{2}gh^2 \end{pmatrix} \tag{3.1}$$

and then subsequently extended to deal with the dimensional-split formulation as well as bathymetry and dry states.

3.1. Exact Solvers

First, an exact Riemann solver is derived based on introductory works [30, chap. 15] [25]. This will then be used in Chapter 5 to decide whether the increased computational effort justifies the gain in accuracy. It should be noted that even though the Riemann problem can be calculated exactly, it is not derivable in a closed form due to its non-linearity. In general, it is helpful to recall the insights gained for linear hyperbolic systems in Section 2.2: A system of

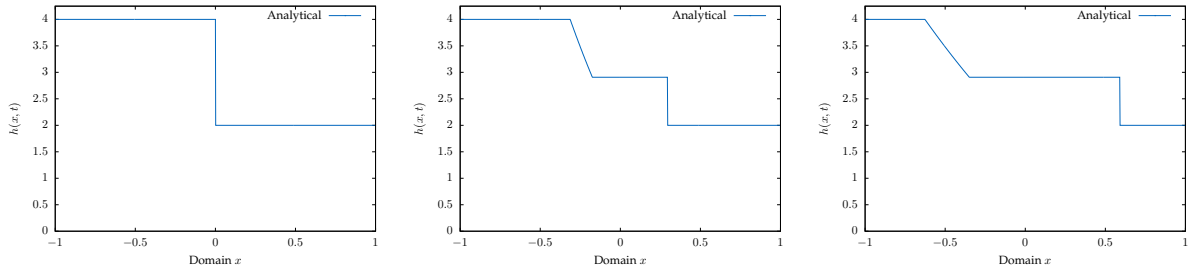


(a) Rarefaction (b) Shock

Figure 3.1: Sketch of the x/t plane for a Riemann problem with rarefaction and shock waves. As the problem is nonlinear, the characteristics are no longer parallel to each other.

m equation results in m separate waves. However, they can now be either shock or (centered) rarefaction waves and arise because the characteristics are no longer parallel to each other as illustrated in Figure 3.1.

What these waves look like when applied to the shallow equation is shown in Figure 3.2: The chosen Riemann problem is the dam break scenario with initial water height $h = 4$ for $x < 0$ and $h = 2$ for $x > 0$. Apart from that, the water is assumed to be at rest, i.e., $hu = 0$. As one can see, two different waves propagate over time: On the left, there are two water levels of height $h = 4$ and $h = 2$ connected by a linear sloping function, the rarefaction wave. On the right, a shock wave between $h = 3$ and $h = 2$ propagates.



(a) $t = 0$ (b) $t = 0.05$ (c) $t = 0.10$

Figure 3.2.: Analytical solution to the dam break Riemann problem for the one-dimensional shallow water equations. The initial discontinuity splits into a rarefaction and shock wave. In a general solution strategy, one has to determine which type of connecting waves occur.

Thus, the solution to the Riemann problem can be rephrased to the following strategy: For each direction, find a state q_m such that q_l is connected to q_m by a (physically correct) 1-shock or 1-rarefaction wave and q_r is connected to q_m by a (physically correct) 2-shock or 2-rarefaction wave, yielding in total four possible combinations. q_m must therefore satisfy both conditions simultaneously.

3.1.1. Two Shock Waves

Section 2.2.1 outlined the derivation of the Rankine-Hugoniot jump conditions, which for linear systems take the form

$$A(q_l - q_r) = \lambda(q_l - q_r). \tag{3.2}$$

Or in other words, the speed of a shock wave is an eigenvalue, and the difference of both discontinuities an eigenvector of q. A generalization of this formula can be derived as well:

$$f(q_l) - f(q_r) = s(q_l - q_r). \tag{3.3}$$

If it was known that one side forms a shock wave, then 3.3 can be used to resolve the Riemann problem. Set q_l to a connecting vector q_* and q_r to an unknown state q and expand the shallow

water equations. Then, a system of two equations arises:

$$f(q_*) - f(q) = \begin{pmatrix} h_*u_* - hu \\ h_*u_*^2 - hu^2 + \frac{1}{2}g(h_*^2 - h^2) \end{pmatrix} = \begin{pmatrix} s(h_* - h) \\ s(h_*u_* - hu) \end{pmatrix} = s(q_* - q_r). \tag{3.4}$$

This system is over-determined as there are three unknowns: h, u, s, whereas h_*, u_* stay fixed. The family of solutions is resolvable by parameterization using h and by rearranging the first equation and inserting it into the second:

$$u^2 - 2u_*u + (u_*^2 - \frac{g}{2}(\frac{h_*}{h} - \frac{h}{h_*})(h_* - h)) = 0 \Rightarrow u(h) = u_* \pm \sqrt{\frac{g}{2}(\frac{h_*}{h} - \frac{h}{h_*})(h_* - h)}. \tag{3.5}$$

Reparameterizing with $h = h_* + \alpha$ gives

$$hu = h_*u_* + \alpha(u_* \pm \sqrt{gh_*(1 + \frac{\alpha}{h_*})(1 + \frac{\alpha}{2h_*})}). \tag{3.6}$$

Now suppose that there are two shocks on every side of the Riemann problem and subsequently enter $* \in \{r, l\}$ for q_* into 3.6. This yields two equations as the middle state q_m lies on both curves:

$$u_m = u_r + (h_m - h_r)\sqrt{\frac{g}{2}(\frac{1}{h_m} + \frac{1}{h_r})}, u_m = u_l + (h_m - h_l)\sqrt{\frac{g}{2}(\frac{1}{h_m} + \frac{1}{h_l})}. \tag{3.7}$$

Both equations together give a formula for h_m that can be solved using a nonlinear root finder, e.g. via Newton's method. Inserting h_m into one of them then gives the solution to the two-shock Riemann problem $q_m = (h_m, h_mu_m)$.

3.1.2. Two Rarefaction Waves

The overall solution of the Riemann problem with a p-rarefaction wave has the structure

$$q(x,t) = \begin{cases} q_l, & \text{if } \frac{x}{t} \leq \lambda_p(q_l) \\ \tilde{q}(\frac{x}{t}), & \text{if } \lambda_p(q_l) \leq \lambda_p(q_r) \\ q_r, & \text{if } \frac{x}{t} \geq \lambda_p(q_r) \end{cases}, \tag{3.8}$$

where $\lambda_p(q_K), K \in \{l, r\}$, denotes the characteristic speed of each side.

In contrast, inside a rarefaction, the values $\tilde{q}$ change proportionally to a corresponding eigenvector r_p, or in other words, they lie on a curve defined by

$$\tilde{q}'(\xi) = r_p(\tilde{q}(\xi)), \tag{3.9}$$

where $\tilde{q}(\xi), \xi = \frac{x}{t}$ is a parameterization of the solution. Expanding this equation for the shallow water equations gives

$$\begin{cases} h'(\xi) = 1 \\ (hu)'(\xi) = u \pm \sqrt{gh} \end{cases} . \tag{3.10}$$

The different signs again yield the 1- or 2-wave. Fixing the point (h_*, h_*u_*) and integrating the equations results then into

$$hu = hu_* \pm 2h(\sqrt{gh_*} - \sqrt{gh}), \tag{3.11}$$
$$\Rightarrow u \pm 2\sqrt{gh} = u_* \pm 2\sqrt{gh_*}, \tag{3.12}$$

where the two quantities are called Riemann invariants for the shallow water equations.

As rarefactions are similarity solutions, they are constant along $\xi = \frac{x}{t}$. This suggest to define the characteristic speeds from 3.8 as

$$\lambda_p(\tilde{q}(\xi)) = \xi. \tag{3.13}$$

If considering the 1-rarefaction connecting q_l with q_m yields, for example, $\lambda_1(\xi) = \tilde{u} - \sqrt{g\tilde{h}} = \xi$ as a result from the system's eigendecomposition. Rewriting this to $\tilde{h} = \frac{(\tilde{u}-\xi)^2}{g}$ and inserting into the Riemann invariants from 3.12 then ends up with

$$\tilde{h} = \frac{(u_l + 2\sqrt{gh_l} - \xi)^2}{9g}, \tilde{u} = \frac{u_l + 2\sqrt{gh_l}}{3} + \frac{2}{3}\xi. \tag{3.14}$$

For a two-rarefactions solution it is sufficient to consider the two integral curves described by Equation 3.11, that connect q_l and q_m as well as q_m and q_r by a rarefaction wave:

$$u_m = u_l + 2(\sqrt{gh_l} - \sqrt{gh_m}), u_m = u_r - 2(\sqrt{gh_r} - \sqrt{gh_m}). \tag{3.15}$$

Equating both formulas yield an explicit term for h_m under the assumption of positivity:

$$h_m = \frac{1}{16g}(u_l - u_r + 2(\sqrt{gh_l} + \sqrt{gh_r}))^2. \tag{3.16}$$

The remaining question for a full solver is how rarefactions can be distinguished from shock waves. A rather simple criterion is the Lax entropy condition, which holds for 2-shock solutions: It postulates that the characteristic speed to the left is greater than the one to the right. For a 1-shock this results in

$$u_l - \sqrt{gh_l} > u_m - \sqrt{gh_m}. \tag{3.17}$$

Together with the generalized Rankine-Hugoniot jump conditions from 3.3 these inequalities simplify to: $h_m > h_l$ and similar for the 2-shock: $h_m > h_r$. If the condition is violated, a rarefaction wave can be assumed.

3.1.3. Generalized Solver

For a full solution to the Riemann problem of Equation 3.1, the intermediate state q_m is allowed to be connected with either a rarefaction or a shock to both quantities q_l, q_r, depending on the underlying initial values. Thus, finding q_m is equivalent to defining the functions

$$\Phi_l(h) = \begin{cases} u_l + 2(\sqrt{gh_l} - \sqrt{gh}), & \text{if } h < h_l \\ u_l - (h - h_r)\sqrt{\frac{g}{2}(\frac{1}{h} + \frac{1}{h_l})}, & \text{otherwise} \end{cases} \tag{3.18}$$

together with

$$\Phi_r(h) = \begin{cases} u_r - 2(\sqrt{gh_r} - \sqrt{gh}), & \text{if } h < h_r \\ u_r + (h - h_r)\sqrt{\frac{g}{2}(\frac{1}{h} + \frac{1}{h_r})}, & \text{otherwise} \end{cases} \tag{3.19}$$

and applying a nonlinear root finder such as Newton's method to $\Phi(h) := \Phi_l(h) - \Phi_r(h)$, returning h_m. The value can subsequently be used in Φ_l to obtain u_m. The structure inside potential rarefaction waves is then reconstructable with the previous section if required.

3.1.4. Passive Tracer

A special form of the shallow water equation arises when the two-dimensional formulation is split along x direction:

$$q_t + \begin{pmatrix} hu \\ hu^2 + \frac{1}{2}gh^2 \\ huv \end{pmatrix}_x = 0. \tag{3.20}$$

The system is also denoted as shallow water equations with passive tracer, as v or u can be interpreted as the concentration of a tracer that flows with the fluid but does not influence it. This equation is of importance for the following sections in the sense that the associated Riemann problem must be solved and implemented in ExaHyPE 2 if the homogeneous version of the shallow water equations (i.e., Equation 2.68 with $\psi(q) = 0, \forall q$) should be simulated. A deeper insight on ExaHyPE 2's design is given in Chapter 4. For now it is sufficient to observe that solving the Riemann problem for 3.20 is not exactly the same as for 3.1 because an additional third component with speed u is involved, and, therefore, three waves occur in total. However, as Toro points out [42], the solution procedure for h and hu are the same as derived above, but for the advected quantity the solution reads

$$v(x,t) = \begin{cases} v_l, & \text{if } u_m > \frac{x}{t} \\ v_r, & \text{if } u_m < \frac{x}{t} \end{cases}, \tag{3.21}$$

as the third equation in 3.20 is decoupled from the other two and changes only across $\frac{dx}{dt} = u$. This property of v is denoted as a linearly degenerate field or contact discontinuity. The same thoughts hold for the split in y direction and u as tracer variable.

3.1.5. Dry Bed

In case the water height h in 3.20 approaches zero, i.e., $h \to 0$, an additional consideration of this case is necessary. The reason is that the velocities $u = hu/h$ and $v = hv/h$ are not defined. These scenarios are also called dry bed problems and occur in the shallow water equations, for instance, because islands are simulated in the topography as sketched in Figure 3.3 or if initially flooded areas dry out due to fluctuations in the water height. There are two main aspects to consider: On the one hand, the numerical model must remain stable and positivity-preserving. However, depending on the method chosen, this entails comparatively many difficulties (such as dividing by a water height that is almost equal to 0 [3]), as will be further discussed in Section 5.1.4 and 5.1.5. On the other hand, the Riemann solver must also be adapted to handle these scenarios, even when neglecting bathymetry b and just setting one side of the initial values to zero. Following Ketcheson et al. [25] or George [16], a dry state q_r to the right is always connected to a wet state q_l to the left via a centered rarefaction wave. This suggests defining the solution to this scenario as follows:

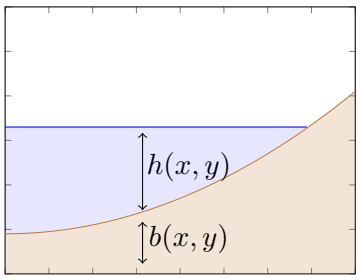


Figure 3.3.: Sketch of a shallow water problem with dry states: the water stops at a coastal region. Dealing with dry states involves difficulties in the numerical scheme as well as the Riemann solver, as the velocities u, v are not defined and need separate treatment.

$$q(x,t) = \begin{cases} q_l, & \text{if } \frac{x}{t} \leq u_l - \sqrt{gh_l} \\ \tilde{q}_l, & \text{if } u_l - \sqrt{gh_l} \leq \frac{x}{t} \leq u_l + 2\sqrt{gh_l} \\ 0, & \text{otherwise} \end{cases}, \tag{3.22}$$

where $\tilde{q}$ is given in Section 3.1.2. A visualization for this scenario in one dimension can be found in Appendix B.2. The same can be stated for the opposite case, i.e., q_l is dry and q_r is wet. For completely dried regions ($h_l = h_r = 0$), it is intuitive to define the Riemann solution to zero, as neither water nor momentum can change at all. To summarize the exact Riemann solution, a pseudo algorithm is provided:

```
IN: (hL, huL, hvL) = qL, (hR, huR, hvR) = qR, x, t
if hL <= 0 and hR <= 0:
  flux := 0
else if hL > 0 && hR <= 0:
  qm = (3.22)
else if hR > 0 && hL <= 0:
  qm = (3.22, analogously)
else:
  hm = findroot((3.18) - (3.19))
  um = phi_l(hm) // (3.18)
  vm = if um > 0.0 then vL else vR
  qm = determine_rarefaction_structure(hm, um, vm) // using (3.14)
```

As the pseudo code shows, several steps are involved in the exact solution, which might increase computational complexity. Subsequently, the next section tries to find cheaper, but potentially less accurate, alternatives.

3.2. Approximate Solvers

The previous section derived an exact Riemann solver for the shallow water equation with passive tracer and dry beds. Every exact Riemann solver has the property of being complete, i.e., it fully covers the wave structure of the Riemann problem. However, this is not always necessary for an approximation of the solution. For example, one could argue that the flux is driven by two strong waves only - and thus, calculating their influence suffices to properly approximate all unknowns. Riemann solvers that use only a part of the underlying wave structure are called incomplete and are always an approximation to the underlying true solution. Examples of the latter are HLL-type solvers, which typically use two waves to approach the solution. However, there are also complete approximate solvers, such as the Roe or other linearized solvers. In the following, both types of solvers are briefly discussed on the homogeneous shallow water equations with passive tracer.

3.2.1. HLL-family

One way to derive a general class of two-wave solvers are the Rankine-Hugoniot jump conditions and assuming only two present waves $\mathcal{W}_1, \mathcal{W}_2$ with speeds s_1, s_2 [25]. Then, the difference between the fluxes on the right and left side is just the modifications from both edges:

$$\begin{aligned} f(q_r) - f(q_l) &= s_1\mathcal{W}_1 + s_2\mathcal{W}_2 = s_1(q_m - q_l) + s_2(q_r - q_m) \\ \Rightarrow q_m &= \frac{f(q_r) - f(q_l) - s_2 q_r + s_1 q_l}{s_1 - s_2}, \end{aligned} \tag{3.23}$$

leading to a scheme of the form

$$q_m = \begin{cases} q_l, & \text{if } 0 \leq s_1 \\ \frac{f(q_r) - f(q_l) - s_2 q_r + s_1 q_l}{s_1 - s_2}, & \text{if } s_1 < 0 < s_2 \\ q_r, & \text{if } 0 \geq s_2 \end{cases}. \tag{3.24}$$

In this case, the wave velocities were explicitly denoted by s_i instead of λ_i because they do not necessarily have to coincide with the eigenvalues as they do for linear problems. This basic scheme is also denoted as two-wave or HLL-based solver and gives a template for Riemann solvers depending on the choice of s_1, s_2. The most common implementations are the Lax-Friedrich, HLL, and HLLE solvers. There are also extensions to the scheme to allow for three waves, which the HLLC solver demonstrates.

Lax-Friedrich and Local Lax-Friedrich

The Lax-Friedrich approach can be gained from Equation 3.23 by assuming $-s_1 = s_2 = s$. Then it reduces to

$$q_m = \frac{f(q_r) - f(q_l)}{2s} - \frac{1}{2}(q_r + q_l). \tag{3.25}$$

The choice of s then determines the method. The original Lax-Friedrich method uses $s = \max(|\lambda(f'(q))|)$ on the entire grid, whereas the local Lax-Friedrich selects the maximum on

the current local domain $s = \max_{\Omega_k}(|\lambda(f'(q))|)$. The latter choice is also denoted as Rusanov flux. An alternative form [30] that directly yields the flux at the cell interface is

$$F_{i-\frac{1}{2}} = f(q_m) = \frac{1}{2}(f(q_r) + f(q_l) - s(q_r - q_l)). \tag{3.26}$$

HLL

Both Lax-Friedrich methods suffer from strong numerical diffusion [30, chap. 4], which smears out discontinuous solutions. The original HLL (Harten, Lax, van Lear) [20] solver helps in mitigating that issue. It can be obtained by either substituting

$$s_1 = \min_{\Omega_k}(\lambda), s_2 = \max_{\Omega_k}(\lambda) \tag{3.27}$$

directly in 3.24, or alternatively, one can integrate the integral form of a one-dimensional conservation law 2.2 over a control volume $[x_l, x_r] \times [0, T]$ and rearrange the equation, as summarized by Toro [44]. This results then in

$$F_{i-\frac{1}{2}} = \begin{cases} f(q_l) & \text{if } 0 \leq s_1 \\ \frac{s_2 f(q_l) - s_1 f(q_r) + s_1 s_2 (q_r - q_l)}{s_2 - s_1} & \text{if } s_1 \leq 0 \leq s_2 \\ f(q_r) & \text{if } 0 \geq s_2 \end{cases}. \tag{3.28}$$

HLLE

Einfeldt proposed a further development of the HLL scheme in the context of gas dynamics [13], which resolves shocks more accurately:

$$s_1 = \min_p(\min(\lambda(f'(q_l))_p, \hat{\lambda}_p)), s_2 = \max_p(\max(\lambda(f'(q_r))_p, \hat{\lambda}_p)), \tag{3.29}$$

where $\hat{\lambda}_p$ denotes the p-th eigenvalue of the Roe average, which is further discusses in Section 3.2.2. In addition, HLLE does not need an entropy fix like the Roe solver.

Another modification to HLLE is HLLEM [14], where the attempt was made to approximate a potential third wave as well. This scheme has also been successfully employed in shallow-water simulations [8].

HLLC

Another approach to resolve more than two waves was made by Toro [43]. He generalizes the idea of HLL to a three-wave solver of the form using three velocities S_L, S_R, S_* and summarized his approach under the name HLLC (HLL with contact discontinuity). The starting point is Equation 3.30 [44], which is obtainable by integrating the integral form over

$[x_L, 0] \times [0, T]$ and $[0, x_R] \times [0, T]$ under assumption of a generic intermediate speed S_*:

$$F_{i+\frac{1}{2}} = \begin{cases} F_L & 0 \leq S_L \\ F_{*L} = F_L + S_L(Q_{*L} - Q_L) & S_L \leq 0 \leq S_* \\ F_{*R} = F_R + S_R(Q_{*R} - Q_R) & S_* \leq 0 \leq S_R \\ F_R & 0 \geq S_R \end{cases}. \tag{3.30}$$

For the shallow water equations the star vectors expand to

$$Q_{*L} = \begin{pmatrix} h_* \\ h_* S_* \\ h_* \psi_L \end{pmatrix}, Q_{*R} = \begin{pmatrix} h_* \\ h_* S_* \\ h_* \psi_R \end{pmatrix}. \tag{3.31}$$

Using the assumptions $h_{*L} = h_{*R} = h_*$ and $u_{*L} = u_{*R} = u_* = S_*$, which are fulfilled in the exact solution, yields

$$h_* = \frac{h_R(u_R - S_R)}{u_* - s_R}, S_* = u_* = \frac{s_L h_R(u_R - S_R) - s_R h_L(u_L - S_L)}{h_R(u_R - S_R) - h_L(u_L - S_L)}. \tag{3.32}$$

Let be $a_L = \sqrt{g h_l}, a_R = \sqrt{g h_R}$. As stable and efficient estimate for S_L and S_R he states

$$S_L = u_L - q_L a_L, S_R = u_R + q_R a_R, \tag{3.33}$$

with

$$q_K = \begin{cases} \sqrt{\frac{1}{2} \frac{\bar{h}(\bar{h} + h_K)}{h_K^2}}, & \text{if (shock) } \bar{h} > h_K \\ 1, & \text{otherwise (rarefaction)} \end{cases} \tag{3.34}$$

for $K \in \{R, L\}$ as well as

$$\bar{h} = \frac{1}{g}(\frac{1}{2}(a_L + a_R) - \frac{1}{4}(u_R - u_L))^2. \tag{3.35}$$

In general, HLL-type Riemann solvers provide stable results under low computational costs. Their quality can be seen as a good compromise between efficiency and accuracy for many applications, even though they introduce numerical diffusion and dissipation and are not able to deal with contact discontinuities (except for the HLLC or HLLEM scheme). However, their main advantage lies in the universal applicability without the need to know the full wave structure of a system. The latter is especially difficult when dealing with large highly non-linear systems, for which it is difficult to derive a full eigendecomposition and, with that, an exact Riemann solver. Subsequently, all of them are implemented in ExaHyPE 2 for evaluation throughout this work, except for HLLEM, as the HLLC solvers seem to be more widespread in literature. However, when focusing on accuracy, more elaborate approximate solvers are potentially required, and it might be enlightening to compare them. One of the most popular, the so-called Roe solver, is derived in the following.

3.2.2. Linearized Solvers and Solver of Roe

A simple but complete approximate class of Riemann solvers are linearized solvers. Here, the initial problem $q_t + f(q)_x = 0$ is replaced by a suitable linearization at each cell interface:

$$q_t + \hat{A} q_x = 0, \tag{3.36}$$

where $\hat{A}$ is valid approximation of f' at Q_i, Q_{i-1}. LeVeque states that employment of linearized solvers is easily justified, as the solution to conservation laws typically consists of isolated shocks and contact discontinuities [30, chap. 15]. Here, the variations in the Jacobian matrix are roughly constant, and $\hat{A}$ is an acceptable approximation to the real flux. However, he emphasizes that the chosen linearization should satisfy the following conditions [30, chap. 15]:

1. **Hyperbolicity**: $\hat{A}$ it is diagonalizable with real eigenvalues.
2. **Consistency**: $\hat{A} \to f'(q)$ for $Q_{i-1}, Q_i \to q$.
3. **Conservation** across discontinuities: $f(q_r) - f(q_l) = \hat{A}(q_r - q_l)$.

Then, the theory for analytical solutions of constant-coefficient system from Section 2.2 can be applied and the solution then consists of m waves with speeds $\hat{\lambda}$. LeVeque also discusses some choices for $\hat{A}$ [30, chap. 15], e.g.

$$\hat{A} = f'(\hat{Q}), \tag{3.37}$$

where $\hat{Q}$ is an average of q_l, q_r, e.g. $\hat{Q} = \frac{1}{2}(q_l + q_r)$. Another option could be to average the fluxes directly as in

$$\hat{A} = \frac{1}{2}(f'(q_l) + f'(q_r)). \tag{3.38}$$

However, both choices may not fulfill the desired properties. Roe's idea, on the other hand, was to establish a matrix $\hat{A}$ based on the properties (1)-(3), which is equal to $f'(\hat{q})$ when evaluating at a specific state $\hat{q}$ [40]. $\hat{q}$ is thereby denoted as Roe average and $\hat{A}$ as Roe linearization or Roe matrix. For now, assume that $\hat{A}$ has already been obtained. Then, it can be shown that the Roe numerical flux is directly given by

$$f(q_m) \approx \hat{A} q_m = \frac{1}{2}(f(q_l) + f(q_r) - \sum_{p=1}^{m} (\hat{\lambda}_p^+ - \hat{\lambda}_p^-)\alpha_p \hat{r}_p), \tag{3.39}$$

where the $\hat{\lambda}_p, \hat{r}_p$ are eigenvectors and eigenvalues of the Roe matrix and $\alpha = \hat{R}^{-1}(q_r - q_l)$ the modifications obtained from the inverse of the eigendecomposition of it.

In his original idea, Roe chose a set of parameter vectors for the Euler equations to derive $\hat{A}$. A more straightforward approach found in Ketcheson et al. [25] is to start with property (3) to derive the average states:

$$f(q_r) - f(q_l) = \hat{A}(q_r - q_l) \Rightarrow f(q_r) - f(q_l) = f'(\hat{q})(q_r - q_l). \tag{3.40}$$

Solving this equation for the x-dimensional split shallow water equation yields

$$\hat{h} = \frac{h_r + h_l}{2}, \hat{u} = \frac{\sqrt{h_l}u_l \pm \sqrt{h_r}u_r}{\sqrt{h_l} \pm \sqrt{h_r}}, \hat{v} = \frac{(h_r u_r v_r - h_l u_l v_l) - \hat{u}(h_r u_r - h_l u_l)}{(h_r u_r - h_l u_l) - \hat{u}(h_r - h_l)}, \tag{3.41}$$

even though LeVeque remarks that $\hat{v}$ calculated analogously to $\hat{u}$ can be used successfully in practice [30, chap. 21]. The Roe solver offers a good approximation for contact discontinuities and shock waves, with exact results for the latter in certain problems. The properties (1)-(3) of the derived matrix stabilize the solver additionally. Moreover, the Roe averages are easily computed in many applications, e.g., for the Euler equations. However, it has several weaknesses, as Toro comments [43, chap. 11]: First, it cannot represent rarefaction waves since linear Riemann problems consist only of shocks, which can make the solution very inaccurate, especially in case a (tran)sonic rarefaction occurs (i.e., a rarefaction where the sign of the characteristics changes from negative to positive). However, this problem is circumventable by a so-called entropy fix. Furthermore, the Roe solver is not positivity-preserving for the shallow water equations, resulting in negative values for the water height h. Unfortunately, this property is shared by all linear solvers [25]. As an alternative, the HLLE solver can be used, which, on the one hand, does not require an entropy fix and, on the other hand, is depth positive-semidefinite. Nevertheless, as the Roe solver is a standard technique in the context of Riemann solvers, it is also implemented in ExaHyPE within this thesis.

3.2.3. Other Solvers

To conclude this section, other solvers are briefly discussed that are often found in literature: The Riemann solver of Osher [15] uses integration in phase space, where the final scheme depends on the selected integration paths and their ordering as well as the choice of intersection and sonic paths [43, chap. 12]. With that, the Osher solver yields a complete approximate solver that does not need an entropy fix and is well applicable to nonlinear equations, as it may capture both shocks and contact discontinuities accurately [12].

Another class is approximate state solvers, where an approximation to the state rather than to the flux is calculated [43, chap. 9]. Possible choices here are, for example, solvers based on the exact solution, like a two rarefactions exact solver. Alternatively, so-called adaptive Riemann solvers fall as well into this categorization, where simple solvers are used in smooth regions and isolated shocks but more advanced ones elsewhere.

A maybe interesting addition to the group of linearized solvers is the Riemann solver of Donat and Marquina [7], where an alternative flux is used to overcome the problems of the Roe solver, similar to an entropy fix.

Furthermore, there exist specific versions of the HLL-type solver dependent on the underlying physical problems, such as the HLLD solver of Miyoshi and Kusano [34] or the HLLEMS solver of Feng et al. [36] show, for instance.

Furthermore, Feng et al. [37] provide a broader classification of Riemann solvers. While they classify Roe and HLL-type solvers as Flux Difference Splitting (FDS) schemes, they state two other classes: One of them are the Flux Vector Splitting (FVS) schemes with the Steger-Warming Splitting as well as the van Leer's FVS scheme, the other are Flux Splitting schemes with the AUSM scheme as example.

3.3. Including Source Terms and Non-Conservative Products

The Riemann problems for the shallow water equations considered in the previous section were all homogeneous. However, for realistic scenarios, a simulation of the bathymetry is essential, which in one spatial dimension leads to a hyperbolic equation of the form

$$q_t + f(q)_x = \psi(q, x), \tag{3.42}$$

as can be seen when splitting Equation 2.68 in x or y direction. These types of equations are also called non-conservative hyperbolic PDE or balance law because the flux function needs to be balanced with the source term. If the term involves derivatives of q (like it is the case for the shallow water equations),

$$q_t + f(q)_x = \psi(\nabla q, x), \tag{3.43}$$

ψ is sometimes denoted as non-conservative product instead of (algebraic) source term. This form is required in ExaHyPE 2, as the source terms are not allowed to depend on derivatives of the quantities (recall Equation 2.67). Mathematically, the more general formulation 3.42 is preferable, and subsequently, this section discusses approaches to solve the belonging Riemann problem.

Sometimes, it is possible to rewrite balance laws back to conservation laws. Under the assumption of smoothness, this can be done for the shallow water equations as well (a derivation of this form can be found in Appendix B.1) [29]:

$$q_t + f(q)_x + g(q)_y = 0$$

$$q = \begin{pmatrix} h \\ u \\ v \\ b \end{pmatrix}, f(q) = \begin{pmatrix} hu \\ \frac{1}{2}u^2 + g(h+b) \\ huv, \\ 0 \end{pmatrix}, g(q) = \begin{pmatrix} hv \\ huv, \\ \frac{1}{2}v^2 + g(h+b) \\ 0 \end{pmatrix}. \tag{3.44}$$

Then, the solution ideas from the previous sections are applicable again. Note that the conserved variables have now changed (velocities u, v instead of momentum hu, hv).

However, such reformulations are not always desirable, e.g. if the underlying assumptions are not applicable. One way to circumvent the Riemann problem with source term is the fractional step or operator splitting method. Here, one first solves the homogeneous equation and then a system of ODEs:

$$\begin{aligned} q_t + f(q)_x &= 0, \\ q_t &= \psi(q, x). \end{aligned} \tag{3.45}$$

However, this approach may break the well-balancedness of numerical schemes if both steps do not exactly balance, resulting in poor accuracy [31]. Well-balancedness is an essential property for the stability of shallow water equations, as will be further discussed in Chapter 5. Thus, many augmented schemes have been developed in the past to prevent such issues. Some of them are reviewed in the following.

One approach to balance source terms with fluxes was introduced by LeVeque in 1998 [31]: He modified the Riemann problem in his wave propagation algorithm for quasi-steady solutions of hyperbolic PDEs by introducing a new discontinuity for a cell interface i:

$$\frac{1}{2}(q_l + q_r) = q_i. \tag{3.46}$$

In combination with decomposing the flux into

$$f(q_l) - f(q_r) = \psi(q_i, x_i)\Delta x \tag{3.47}$$

it is aimed at keeping the cell averages unchanged and balancing source and flux. A similar idea was proposed by Jenny and Müller [24]. A more robust scheme was later created by Bale et al. [2] [33], allowing the solution of problems with spatially varying flux function and source terms, i.e., balance laws of the form

$$q_t + f(q, x)_x = \psi(q). \tag{3.48}$$

The key idea is to use a flux-based wave decomposition under the assumption of cell-centered averages on the left and right state, where the flux difference of both cells is decomposed similarly to the analytical approach for linear systems (see, for example, 2.30):

$$f(q_r) - f(q_l) - \Delta\Psi_{i-\frac{1}{2}} = \sum_{p=1}^{m} (\beta_{i-\frac{1}{2}})_p (r_{i-\frac{1}{2}})_p \equiv \sum_{p=1}^{m} (\mathcal{Z}_{i-\frac{1}{2}})_p. \tag{3.49}$$

The new waves $\mathcal{Z}_p$ are then called f-Waves. The eigenvectors and eigenvalues can be derived, for example, using the Roe linearization. On the other hand, the function ψ summarizes the effects of the source term, which for the shallow water equations expands to

$$\Delta x \Psi_{i-\frac{1}{2}} = -\frac{g}{2}(b_r - b_l)(h_r + h_l). \tag{3.50}$$

The flux at the intermediate state is then obtainable with the relation:

$$F_{i-\frac{1}{2}} = f(q_l) + \sum_{p:\lambda_p < \frac{x}{t}} (\mathcal{Z}_{i-\frac{1}{2}})_p. \tag{3.51}$$

A generalization of the f-Wave idea is based on relaxation schemes as LeVeque and Pelanti point out [33]: Here, the decomposition is based on $2m$ waves using the ansatz

$$\begin{pmatrix} q_r - q_l \\ f(q_r) - f(q_l) \end{pmatrix} = \sum_{p=1}^{2m} \alpha_p \begin{pmatrix} w_p \\ \phi_p \end{pmatrix}, \tag{3.52}$$

which allows for a more unified view of solvers as the HLL-type and Roe solvers. George captured this thought to establish his own augmented solver with $m + 1$ waves for the one-dimensional shallow water equations with bathymetry [17]:

$$\begin{pmatrix} h_r - h_l \\ h_r u_r - h_l u_l \\ \phi(q_r) - \phi(q_l) \\ b_r - b_l \end{pmatrix} = \sum_{i=0}^{3} (\alpha_{i-\frac{1}{2}})_p (w_{i-\frac{1}{2}})_p. \tag{3.53}$$

$\phi(q) = (hu^2 + \frac{1}{2}gh^2)$ is therefore the flux of momentum and $(w_{i-\frac{1}{2}})_p \in \mathbb{R}^3$ independent vectors. Then, the f-waves are the middle two components of

$$(\mathcal{Z}_{i-\frac{1}{2}})_p = (\alpha_{i-\frac{1}{2}})_p (w_{i-\frac{1}{2}})_p. \tag{3.54}$$

On the other hand, the vectors w and wave speeds s are chosen from a combination of the Roe averages for $p = 1, 3$ and a corrector wave for $p = 2$. George's solver unites the advantages of the Roe and HLLE solver, as well as the f-wave approach: It is positive-semidefinite and yields an exact solution for single-shock problems with entropy fix. Moreover, it is well-balanced in the context of LeVeque's wave propagation algorithm. For ExaHyPE 2, both f-Wave and the solver of George are implemented to include source terms. In addition, the fractional step approach of the software is applied to the homogeneous solver to allow for general inclusion of bathymetery. More implementation details are discussed in the following chapter.

4. Implementation in ExaHyPE 2

After introducing Riemann solvers in the previous chapter, their implementation is now considered. At first, the created code and built-in Riemann solvers are analyzed. Here, all statements refer to ExaHyPE 2 with non-linear kernels using ADER-DG. ExaHyPE 2 also allows for linear kernels, which result in an optimized code. However, except for investigation the correctness of the Riemann solvers for the shallow water equations, it has not been used in this work. The logic of the implementation remains the same for Finite Volumes and Runge-Kutta DG, but as the solvers are tightly integrated into the generated code, they cannot be isolated and adapted as a stand-alone component. After that, a brief discussion of the most important aspects of the implementation is given to assess the efficiency of the code later in the work. At the end of this chapter, the implemented code is examined for correctness using the linearized shallow water equations. ExaHyPE 2 is continuously under development. Thus, a fork[1] was created for this thesis, starting with commit 04daa08.

4.1. Analysis

After running ExaHyPE 2's code generation (recall Section 2.4 in combination with Appendix B.1 for a short introduction on that), the main C++ file has to be implemented. It can be found in the root directory where the problem definition was called. In the case of the example given in the appendix, it is named out.cpp and has empty methods bodies that need to be completed. These are (for readability, the method arguments were omitted):

- refinementCriterion (...) - A method that allows the implementation of a criterion to refine the grid based on the provided min_cell and max_cell parameter. For ExaHyPE 2, grid refinement is still experimental and has to remain static throughout the simulation.
- initialCondition (...) - With this function, initial conditions may be defined.
- boundaryConditions(...) - Here, boundary conditions (BCs) need to be stored in the parameter vector Qoutside based on the boundary inside of the domain Qinside. In many cases these are outflow boundary conditions (Qoutside[i] = Qinside[i]), but also reflecting boundary conditions (Qoutside[i] = −Qinside[i]) or no-slip BCs (Qoutside[i] = 0) are commonly used.
- maxEigenvalue(..) - Returns the maximum eigenvalue of the (quasi-)linear form of the PDE. This value is used for time step size control and the built-in Rusanov solver.
- flux (...) - Computes $f(q)_n$ along direction n.
- Not in the example but also required for providing all test scenarios from Chapter 5 is nonconservativeProduct(...) to add $\psi(\nabla q, q)$. For an explicit source term not depending on the quantity's derivative, the method algebraicSource (...) is to be preferred.

[1] https://gitlab.lrz.de/hpcsoftware/Peano/-/forks

ExaHyPE 2 generates per-default a Rusanov solver in the code. There are two ways to overwrite it: Either use the option

```
thesolver.set_implementation(riemann_solver=exahype2.solvers.aderdg.PDETerms.
  User_Defined_Implementation)
```

in the Python problem definition, which adds the method riemannSolver(...) to the C++ file or overwrite the same method within the file RiemannSolver.cpp in generated/kernels/aderdg/(non)linear. Both functions have the following prototype (with the template paremeter cCompType being double for most cases):

```
void generated::kernels::AderDG::nonlinear::riemannSolver(
examples::exahype2::tutorial::out &solver,
cCompType *__restrict__ FL,
cCompType *__restrict__ FR,
const cCompType *__restrict__ const QL,
const cCompType *__restrict__ const QR,
const double t,
const double dt,
const tarch::la::Vector<Dimensions, double> &faceCentre,
const tarch::la::Vector<Dimensions, double> &dx,
const int direction);
```

and are called once per face for each time step. To understand this more in detail, a two-dimensional cell is shown in Figure 4.1. Along each direction, there is a negative and a positive face (x direction: red/green, y direction: blue/orange), having a left and a right side. Each side therefore has $d+1$ nodes, which are vectors of unknowns, with d being the degree of the basis polynomials. For the example above, it holds $d=3$, thus there are in total 4 nodes with 3 unknowns each, which is furthermore illustrated in Figure 4.2. For each combination of left and right side, a Riemann problem has to be solved. In n dimensions, this concept transfers to 2^n faces with $(d+1)^{n-1}$ vectors of unknowns per cell. Or, coming back to the initial statement, one call of riemannSolver(...) needs to solve $(d+1)^{n-1}$ Riemann problems.

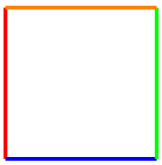


Figure 4.1.: Two-dimensional cell consisting of 4 faces. Each face has $d+1$ points, when considering DG with polynomials of degree d.

(a) Left edge (b) Right edge (c) Top edge (d) Bottom edge

Figure 4.2.: Faces for which ExaHyPE's Riemann solver is called for example problem B.1. Each left/right combination of black dot marks an (analytical) Riemann problem, while riemannSolver(...) solves all problems for a given face in one function call.

The function declaration of riemannSolver(...) shall be discussed briefly as well before designing new implementations:

- solver - A reference to the problem-solving component, which allows to call the methods from out.cpp as stated above.
- FL / FR - Pointers containing $f(q_l)$ and $f(q_r)$ for each point of a cell's face.
- QL / QR - Pointers containing q_l and q_r for each point of a cell's face.
- t - Current time.
- dt - Current time step size.
- faceCentre - Coordinates of the center of the current face.
- dx - Size of the cell.
- direction - The direction along which the face is considered (0 - x, 1 - y, 2 - z axis). As ExaHyPE 2 only allows for Cartesian grids, the Riemann problem vanishes in all other directions. In other words, a dimensional-split approach is followed, reducing the Riemann problem for the two-dimensional shallow water equations to the one-dimensional shallow water equations with passive tracer and bathymetry along each direction, for example.

The function requires overwriting FL and FR with the flux evaluated at the resulting intermediate state q_m of the Riemann solution for both vectors, i.e. FL = FR = $f(q_m)$. However, there are situations where the left and the right flux differ, for instance, if the flux shall be limited in certain areas or in case of non-homogeneous problems and fractional-step-based solvers.

4.2. Design and Implementation

Based on the analysis from the previous section and using pre-built methods of the code, a general Riemann solver for ExaHyPE 2 may look as follows:

```
1  void riemannSolver(...)
2  {
3    constexpr auto vectors = Dimensions == 2 ? getBasisSize() : getBasisSize() *
     getBasisSize();
4    constexpr auto unknowns = getNumberOfVariable();
5    cCompType FL2[unknowns];
6    cCompType FR2[unknowns];
7
8    for (auto v = 0; v < vectors; ++v)
9    {
10     auto *QL2 = &QL[v * unknowns];
11     auto *QR2 = &QR[v * unknowns];
12
13     //solve the dimensional split Riemann problem using QL2 and QR2, e.g. with
     Roe solver and store results in FL2, FR2
14
15     std::memcpy(&FL[v * unknowns], &FL2[0], unknowns * sizeof(cCompType));
16     std::memcpy(&FR[v * unknowns], &FR2[0], unknowns * sizeof(cCompType));
17   }
18 }
```

In line 3, the number of nodes on each side is determined at compile time. The same is repeated with the unknowns in line 4. 5-6 allocate buffers for the output of the Riemann problem. Line 8 iterates through all left/right combinations and 10-11 consequently move the pointers to the currently considered left/right pairs of initial values. If assuming that the Riemann problem has already been solved in FL2 and FR2 (marked by the comment in line 13), the local result can be copied into the global result vector (lines 15-16).

A shortened implementation for the HLLE solver, replacing the comment in the listing before, may then, for instance, be stated as follows, if s_1 and s_2 are already given:

```
1  if (s1 >= 0.0) { [...] }
2  else if (s1 <= 0.0 && 0.0 <= s2)
3  {
4    cCompType q_m[unknowns];
5    for (auto j = 0; j < unknowns; ++j)
6    {
7      q_m[j] = (FR2[j] - FL2[j] - s2 * QR2[j] + s1 * QL2[j]) / (s1 - s2);
8    }
9    solver.flux(&q_m[0], faceCentre, dx, t, dt, direction, &FL2[0]);
10   solver.flux(&q_m[0], faceCentre, dx, t, dt, direction, &FR2[0]);
11 }
12 else if(s2 >= 0.0) { [...] }
```

Here, Equation 3.24 is applied directly for each unknown in line 7. Lines 9 and 10 then calculate the flux of the result and store it in the previously allocated buffers.

To include non-conservative products, the fractional-step approach of the existing Rusanov solver is followed for non-augmented schemes, which has to be added after the solution to the homogeneous problem was already calculated:

```
1  cCompType Qavg[unknowns] = {0.0};
2  cCompType gradQ[unknowns] = {0.0};
3  cCompType ncp[unknowns] = {0.0};
4
5  for (int n = 0; n < unknowns; n++)
6  {
7    Qavg[n] = 0.5 * (QL2[n] + QR2[n]);
8    gradQ[n] = QR2[n] - QL2[n];
9  }
10
11 solver.nonconservativeProduct(Qavg, gradQ, ..., ncp);
12
13 for (int n = 0; n < unknowns; n++)
14 {
15   FR[v * unknowns + n] -= 0.5 * ncp[n];
16   FL[v * unknowns + n] += 0.5 * ncp[n];
17 }
```

Again, local buffers are created for intermediate results (1-3). Line 5-9 then approximates q and ∇q. These estimates are then used to call the non-conservative product (line 11), which in this case is supposed to be the only source term. The calculated flux of the Riemann problem is then modified in lines 13-17 from the right- and left-going results.

Based on these insights, the following Riemann solvers are implemented in this work:

Solver	Ref	Shallow water	Linear Elastic Wave
Exact	3.1	hom. and inhom. (Rusanov ansatz)	constant-coefficient
Rusanov	3.2.1	hom. and inhom. (Rusanov ansatz)	hom.
HLL	3.2.1	hom. and inhom. (Rusanov ansatz)	hom.
HLLE	3.2.1	hom. and inhom. (Rusanov ansatz)	-
HLLC	3.2.1	hom. and inhom. (Rusanov ansatz)	-
Roe	3.2.2	hom. and inhom. (Rusanov ansatz)	-
F-Wave	3.3	hom. and inhom.	-
George	3.3	hom. and inhom.	-
2-Rfw	3.1.2	homogeneous	-

Table 4.1.: Implemented Riemann solvers for the shallow water and linear elastic wave equations. Two Rarefactions (2-Rfw) is an exact solver based on the assumption of two rarefaction waves on both sides and recommended by Toro et al. for practical use [45]. The Rusanov ansatz for inhomogeneous problems is taken from the built-in solver of ExaHyPE 2 and modifies the homogenous solution based on an approximation of the non-conservative product.

4.3. Correctness

The correctness of the implemented solvers is verified with the linearized versions of the shallow water equations from Section 2.5.1. As LeVeque points out, they are obtainable by considering waves with only small amplitudes compared to fluid depth and assuming a constant depth h_0 and velocity u_0 [30, chap. 13], [32]. q then describes deviations from these constant states:

$$q(x,t) = \begin{pmatrix} h - h_0 \\ hu - h_0 u_0 \end{pmatrix}, q_0 = \begin{pmatrix} h_0 \\ h_0 u_0 \end{pmatrix}. \tag{4.1}$$

Calculating the flux function $f(q)_x$ and dropping higher-order terms $\mathcal{O}(||q||_2^2)$ yields a constant-coefficient linear system in one spatial direction by setting $A := f'(q_0)$:

$$q_t + Aq_x = 0. \tag{4.2}$$

In that case, the solution to the equation can be obtained analytically following section 2.2.1 and using the one-dimensional eigendecomposition in 2.5.1:

$$q(x,t) = \tilde{w}_1(x - \lambda_1 t)r_1 + \tilde{w}_2(x - \lambda_2 t)r_2. \tag{4.3}$$

Similar thoughts lead to the two-dimensional version of the linearized shallow water equations $q_t + Aq_x + Bq_y = 0$, where the dimensional-split equations

$$q_t + Aq_x = 0, q_t + Bq_y = 0 \tag{4.4}$$

can be solved analytically, too, giving a linearized version of the shallow water equations with passive tracer. These exact solutions are used to ensure correctness of the code using a test

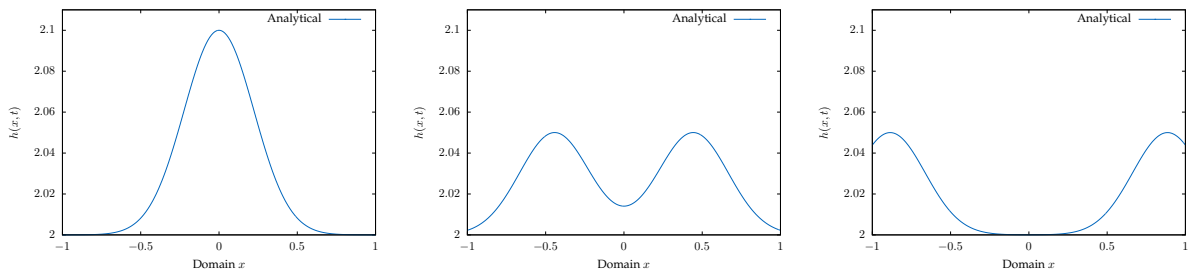
problem, which is defined in equation 4.5 for the following comparisons: A tiny water hump is released at $t = 0$ into $\Omega = [-1, 1] \times [-1, 1]$ and evolves over time. Since ExaHyPE 2 requires at least two dimensions, the problem is designed with constant values along the y-direction. Consequently, the numerical results along $(-1, 0)$ to $(1, 0)$ are comparable with the analytical solution of the first equation from 4.4.

$$q(x, y, 0) = \frac{1}{10} e^{-10x^2} + h_0, q_0 = \begin{pmatrix} h_0 \\ u_0 \\ v_0 \end{pmatrix} = \begin{pmatrix} 2 \\ 0 \\ 0 \end{pmatrix}. \tag{4.5}$$

Figure 4.3 shows the analytical results of h for $t \in \{0, 0.10, 0.20\}$. For the error analysis for a single unknown of q at fixed time t and arbitrary x-coordinates, the absolute error between the numerical solution $\tilde{r}$ and the analytical solution r is considered. The latter is calculated using the non-equidistant Legendre quadrature nodes employed by ADER-DG depending on the order and mesh size. Then, the 2-norm divided by the analytical results yields a relative error, enabling a percentage viewpoint:

$$E_{rel}(x, t) = \frac{||r - \tilde{r}||_2}{||r||_2}. \tag{4.6}$$

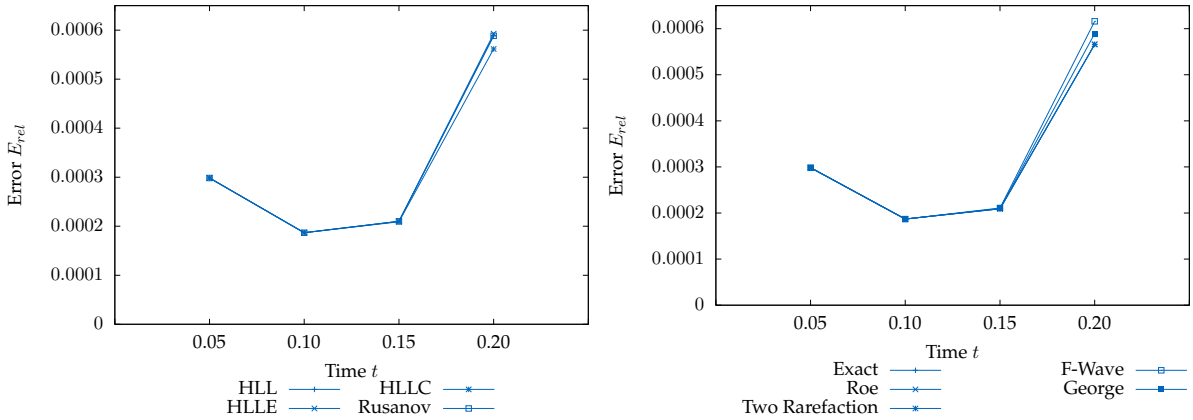
The results calculated with this approach are only reasonable when smooth solutions are considered since discontinuous regions might not be captured accurately enough, which is also taken up in the next chapter. Nevertheless, in this setting the assumption on smoothness apply. Figure 4.3 plots this error over the straight line $(-1, 0)$ to $(1, 0)$ using ExaHyPE 2 for selected points in time. The simulation employs ADER-DG with basis polynomials of degree 3 and fixed grid cells with size 0.1. All solvers reach an error below 0.0006, with the HLL-type solvers having the least at time 0.1. Towards the end of the simulation, the error increases slightly since the considered analytic solution does not incorporate boundary conditions, while ExaHyPE 2 assumes outflow of the waves. A similar error curve is shown in Figure 4.5 for hu, where the overall errors are higher but still decent. For hv, the results are not shown, as they match the analytical solution for the full simulation (i.e., $hv = 0$). These plots show two aspects: Firstly, the error is within an acceptable range for all solvers. Thus,



(a) $t = 0$ (b) $t = 0.1$ (c) $t = 0.2$

Figure 4.3.: Analytical solution of h for test problem 4.5 along the x axis: A tiny hump of water is released into the domain. The x range of the plot was scaled to $[2, 2.1]$ to make the results more visible.

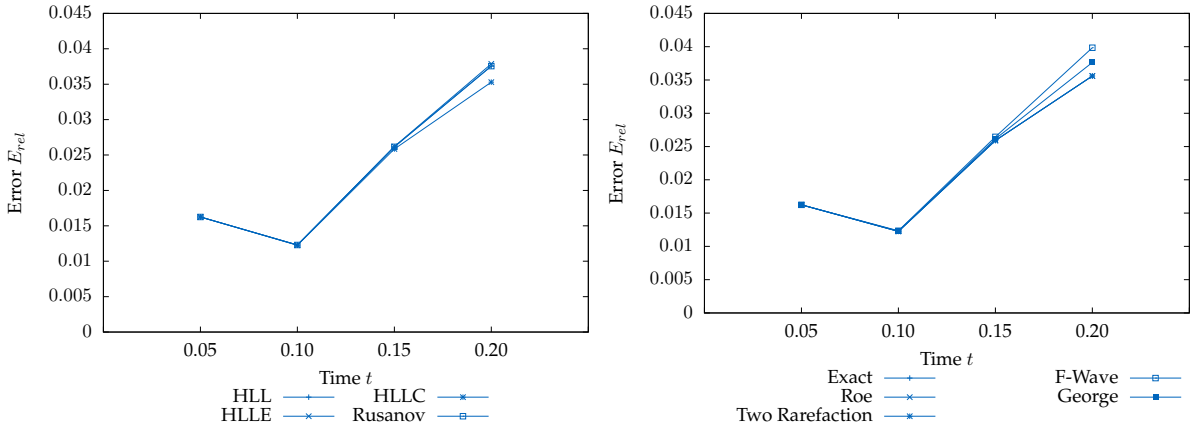
a correct implementation of the Riemann solvers for the shallow water equations with passive tracer in ExaHyPE 2 can be reasonably assumed. Secondly, the differences between the solvers are tiny in this setting, which primarily is attributable to the high order of convergence and the smoothness of the solution. In the next chapter, it is investigated if these insights also apply to other scenarios.



(a) HLL family

(b) Other solvers

Figure 4.4.: Relative error of h for test problem 4.5 at different points in time over the line $(-1, 0)$ to $(1, 0)$ calculated with Equation 4.5. As the magnitudes stay within an acceptable range, a correct implementation of the Riemann-solvers for the homogeneous shallow water equations can be assumed.



(a) HLL family

(b) Other solvers

Figure 4.5.: Relative error of hu for test problem 4.5 at different points in time over the line $(-1, 0)$ to $(1, 0)$ calculated with Equation 4.5. The numerical results for hv are as expected equal to 0. Therefore, this error plot is omitted.

Part III.

Results and Conclusion

5. Results

The previous chapters presented Riemann solvers and discussed their implementational aspects on the example of the shallow water equations with bathymetry. In this chapter, the solvers from Table 4.1 are compared using different example simulations. First, dam break scenarios in one spatial dimension are examined. Afterward, the solvers are evaluated concerning well-balancedness property using the shallow water equations with bathymetry. The treatment of dry states and more complex topography in ExaHyPE 2 is discussed subsequently, and the insights are then applied to inundation and tsunami scenarios. Finally, the linear elastic wave equations are examined as an example of a three-dimensional equation. The focus of this chapter lies on contrasting the implementations in terms of accuracy and certain properties. However, a summary of all solvers regarding performance is given in the last section.

5.1. Shallow Water Equations

If not otherwise stated, the following settings apply to the subsequent experiments using the shallow water equations:

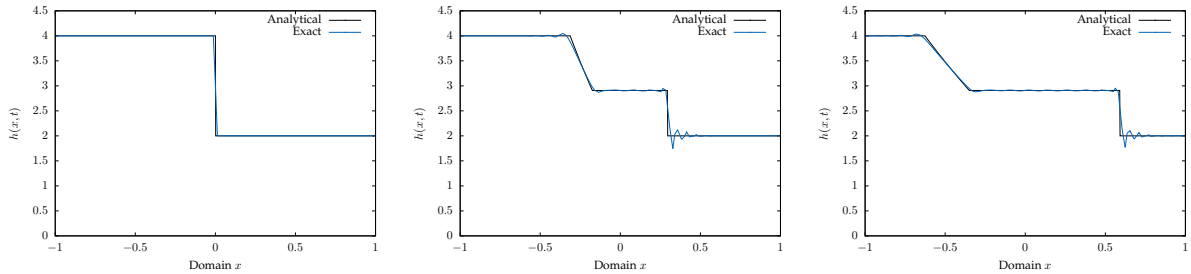
- The realization in ExaHyPE 2 is directly derived from Equation 2.68, where the source term is implemented as non-conservative product. The eigenvalues for time step control are taken from 2.71 and 2.73.
- Nonlinear ADER-DG with polynomials of degree $d = 3$ is employed.
- The mesh size is fixed with 0.1 on the domain $[-1, 1] \times [-1, 1]$ and the time step size is relaxed with the factor 0.9.
- The relative error is measured using the formula and derivations from Equation 4.6 with the norm of a point wise difference using the analytical results on the non-equidistant Legendre quadrature node. All analytical solutions are created using matlab.
- The numerical data is exported with a precision of 13 using paraview.
- For the dam break scenarios, the analytical solution is directly taken from Section 3.1 for comparison.
- Dry states are handled using the analytical solution taken from Section 3.1.5 for all solvers.

Using these properties, the Riemann solvers for the shallow water equations are discussed. Here, the homogeneous version ($\psi = 0$) is investigated first with the help of one-dimensional reference problems and then further extended with bathymetry. Subsequently, initial dry states ($h = 0, \psi \neq 0$) are considered on numerically challenging problems such as discontinuous topography, frequent drying and wetting scenarios, or complex bathymetry on real-life data.

5.1.1. Dam Break

The first considered scenario is a two-dimensional dam break simulation constant in y direction. The initial conditions are provided in Equation 5.1. The expectation for the analytical solution is a left rarefaction and a right shock wave in h along each fixed y, as already outlined in Section 3.1. Figure 5.1 shows numerical and analytical results using the exact Riemann solver along $(x, y = 0)$: While the rarefaction wave is well-captured, oscillations near the shock wave occur. These types of oscillations are called Gibb's type oscillations and are expected when employing ADER-DG with discontinuous solutions. Originally, they were used to describe the problem that the approximation of a discontinuous function by a Fourier series results in over- and undershoots in discontinuous regions [19].

$$q(x, y, 0) = \begin{cases} \begin{pmatrix} 4 \\ 0 \\ 0 \end{pmatrix} & x < 0 \\ \begin{pmatrix} 2 \\ 0 \\ 0 \end{pmatrix} & x > 0 \end{cases}. \tag{5.1}$$

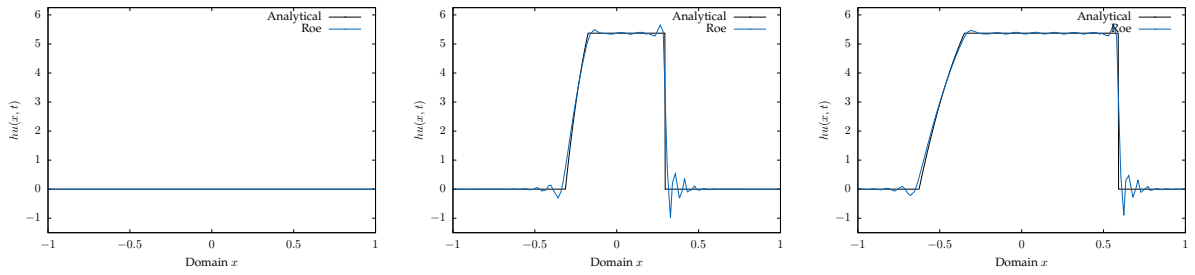


(a) $t = 0$ (b) $t = 0.05$ (c) $t = 0.10$

Figure 5.1.: Analytical and approximated solution of h for Equation 5.1 for fixed $y = 0$. A left rarefaction wave and a right shock wave propagate after release of the discontinuity. The numerical simulation uses the exact Riemann solver. Results are shown along $x \in [-1, 1]$ calculated with ExaHyPE 2 employing ADER-DG and cubic polynomials. Gibb's type oscillations occur around the shock wave while the rarefaction wave is captured more accurately.

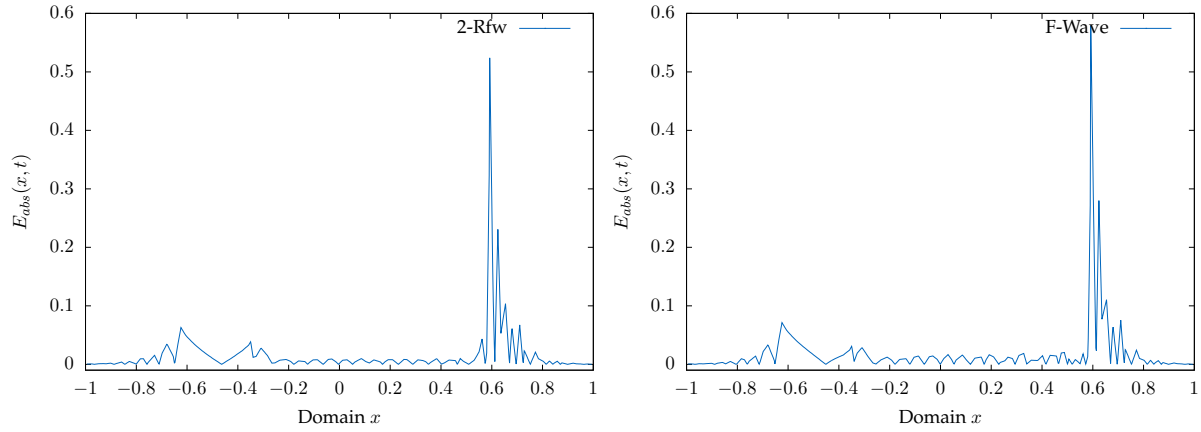
The analytical and numerical results for momentum hu are plotted in Figure 5.2 using the Roe Riemann solver. The oscillations arise here as well and are therefore not an issue of the Riemann solver but rather of the created numerical model. One option to resolve these oscillations is limiters or filters. As they were not available throughout large parts of this work, a discussion of the accuracy without them is performed in the following.

As a direct comparison of all solvers with the analytical results is not practical, the relative error from Equation 4.6 is considered in the following instead. Figure 5.4 shows the value calculated for h over $x \in [-1, 1]$ for fixed $y = 0$ and various points in time using point-wise differences on the non-equidistant grid, which are in addition shown in 5.3 for the George



(a) $t = 0$ (b) $t = 0.05$ (c) $t = 0.10$

Figure 5.2.: Analytical and approximated solution of hu for Equation 5.1. hv is equal to zero, as the problem is constant along y direction. The numerical simulation uses the Roe Riemann solver. Results are shown for fixed $y = 0$ along $x \in [-1, 1]$ calculated with ExaHyPE 2 employing ADER-DG and cubic polynomials. Gibb's type oscillations occur for the dam break scenario independent of the chosen Riemann solver.

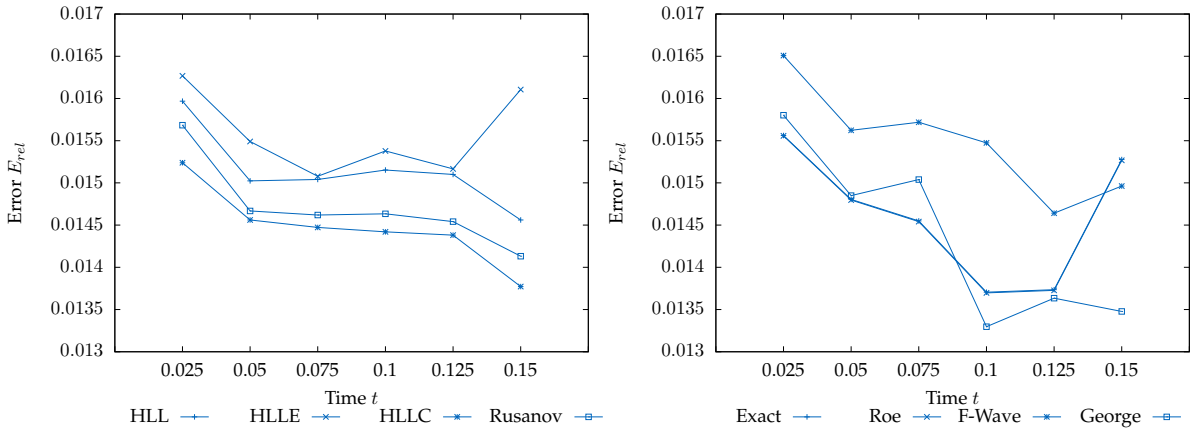


(a) Two-Rarefaction exact solver (b) f-Wave solver

Figure 5.3.: Point wise error of the approximated and analytical solution of h for Equation 5.1 for fixed $y = 0$ along $x \in [-1, 1]$ using the Legendre quadrature nodes from the numerical solution at $t = 0.1$. The results were calculated with ExaHyPE 2 using ADER-DG and cubic basis polynomials and are used to determine the relative error defined in Equation 4.6.

and f-Wave Riemann solver. Figure 5.4a groups HLL-type solvers in one plot, while 5.4b combines the remaining solvers except the two-rarefactions exact solver (since it is nearly identically to the exact solution). Note that the y range was scaled to $[0.013, 0.019]$. As expected due to the similarity of the schemes, the HLL solvers exhibit similar behavior with respect to the error, although the HLLC solver produces the lowest and the HLLE solver the highest error. The overall smallest error is achieved by George's augmented solver starting at $t = 0.1$. The Roe and exact solvers behave almost identically, although the mathematical analysis indicated that the former should not be able to resolve the rarefaction wave, as it is a linear solver. The f-Wave solver performs worst at the beginning and can only approach the

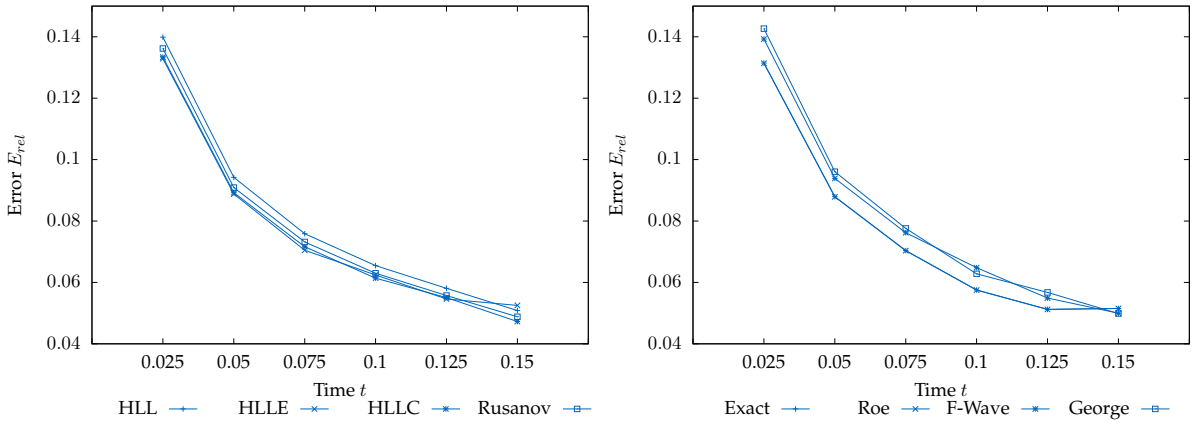
accuracy of the other solvers at later times.



(a) HLL family

(b) Other solvers

Figure 5.4.: Relative error of the dam break scenario 5.1 calculated with Equation 4.6 for unknown h along $x \in [-1, 1], y = 0$ over selected points in time. The HLLC solver performs best across the HLL-type solvers for this setup, while George's augmented solver produces the smallest error in later time points. The solution using the two rarefactions exact solver was omitted due to its similarity to the exact one. Note that the y-axis does not start from zero to make the differences more visible.



(a) HLL family

(b) Other solvers

Figure 5.5.: Relative error of the dam break scenario 5.1 calculated with Equation 4.6 for unknown hu along $x \in [-1, 1], y = 0$ over selected points in time. While the HLLC solver still performs best across the HLL-type solvers for this setup, George's augmented solver produces not as precise results as with h. The solution using the two rarefactions exact solver was omitted due to its similarity to the exact one. Note that the y-axis does not start from zero to make the differences more visible.

Figure 5.5 shows the relative error calculated analogously for hu. According to the graph, the solvers seem to behave almost identically. However, the differences are not as noticeable here due to the larger y range. Nevertheless, it can already be deduced that the HLLE and f-Wave solvers provide more accurate results for momentum than for water height in comparison to the other solvers. Also, HLLC is, in part, the most accurate among the HLL family, while George's augmented solver achieves less precise results. Again, Roe and the exact solver are closely related. A fairer visual comparison, omitting the initial error, can be found in C.1.

Gibb's type oscillations might reduce the overall accuracy of the results near the discontinuity, which is amplified when considering Figure 5.3 another time: The absolute error is large around the shock wave and, therefore, so is the overall relative one, while the values are by far lower in other regions. Therefore, it is more illuminating to distinguish the error on both types of waves, respectively. For example, at $t = 0.1$, the shock wave is located between $x \in [0.5, 0.7]$, while the rarefaction wave is within $x \in [-0.7, -0.3]$ as Figure 5.1c shows. On the contrary, the relative error over these intervals is plotted in Figure 5.6 for all Riemann solvers.

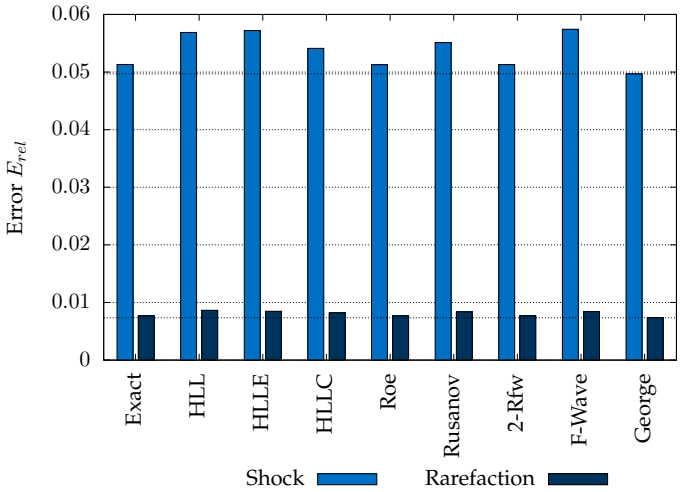
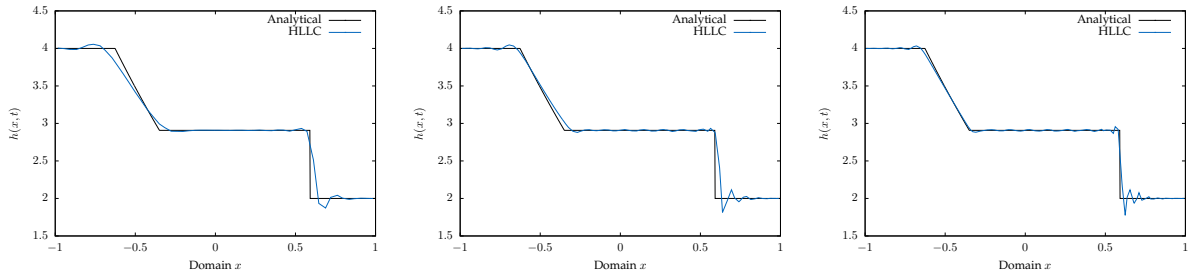


Figure 5.6.: Relative error of shock ($x \in [0.5, 0.7]$) and rarefaction $x \in [-0.7, -0.3]$ wave at $t = 0.1$ for dam break problem 5.1 calculated with Equation 4.6. The additional dotted lines mark the smallest error over all Riemann solvers (here reached by George's augmented solver). Due to Gibb's type of oscillations, the error of shock and rarefaction differ highly.

As can be derived from the figure, there is a remarkable difference for both types of wave: While the error concerning the rarefaction lies around $E_{rel} = 0.008$ (minimum ≈ 0.0075, maximum ≈ 0.0088), it is more than six times as high for the shock wave (minimum ≈ 0.0489, maximum ≈ 0.0568). This is no surprise, as the numerical simulations using any Riemann solver, on one hand, do not necessarily capture the shock accurately enough (and with that, a point-wise error produces large errors around this area) and, on the other hand, do not impose large oscillations in other parts (which increases the accuracy in general there). Thus, an error analysis is performed for both types of waves separately in the following.

Both minima for rarefaction and shock are obtained using George's Riemann solver, while the maxima are reached using HLL for the rarefaction and with f-Wave for the shock wave. Other well-performing solvers for the rarefaction wave are the exact and, as expected, the two-rarefaction exact solver (denoted as 2-Rfw in the figure). Interestingly, the latter also is reasonably precise for the shock wave, together with the Roe solver for the rarefaction. Larger deviations to the analytical solutions are extensively found in the HLL-type solvers, in particular, with the HLLE solver.

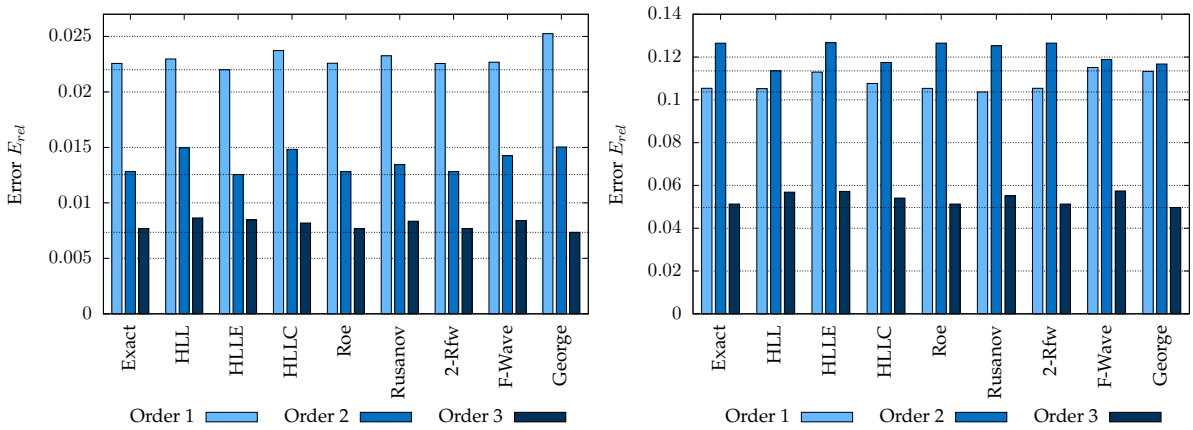
The oscillations in the region of the discontinuity are less significant when the polynomial degree is reduced, which is shown for simulations using the HLLC solver in Figure 5.7. Therefore, the question arises to what extent the numerical solution actually benefits from a higher order and which Riemann solvers profit most from it. For this purpose, the error for both wave types at time $t = 0.1$ is analyzed in Figure 5.8 but this time for the polynomial degrees $d = 1, 2$ in addition. Here, the rarefaction wave is best resolved by the HLLE solver for $d = 1, 2$. The Riemann solver of George has the highest error for these orders but is most accurate with $d = 3$ using this error estimate. Both exact solvers, as well as the Roe solver reveal consistently small deviations from the respective best result. Due to the oscillations, the error of the shock wave is overall higher than for the rarefaction wave. The Rusanov and HLL solver yield the smallest error for $d = 1$. On the other hand, no Riemann solver actually benefits using $d = 2$, as the shock wave is not captured well enough to fit the analytical solution. In contrast, George's solver has the smallest error for $d = 3$ followed by the two exact Riemann solvers, matching the previous results.



(a) $d = 1$ (b) $d = 2$ (c) $d = 3$

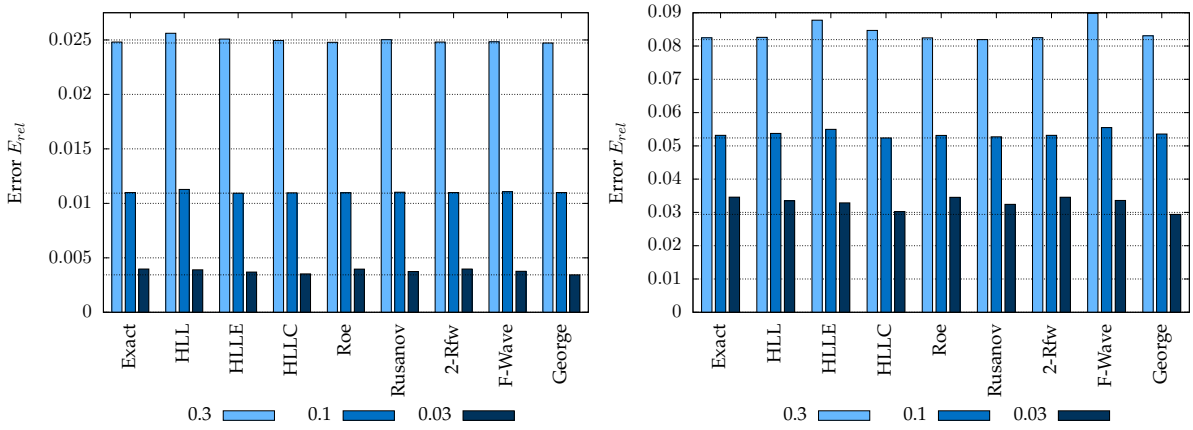
Figure 5.7.: Numerical solution of h for Equation 5.1 using the HLLC Riemann solver for fixed y along $x \in [-1, 1]$ in ExaHyPE 2 with ADER-DG and polynomials of varying order. The Gibb's tye oscillations are less present around the discontinuity for lower-order basis functions, and the Riemann solvers profit differently from them as Figure 5.8 shows.

Similar questions can be asked regarding varying mesh size as illustrated in Figure 5.9. However, as instabilities start to appear by the end of the simulations on finer grids due to enlarged oscillations, the comparison is performed at $t = 0.05$. Here, the rarefaction wave is located around $[-0.5, -0.1]$ and the shock wave within $[0.2, 0.4]$. The smallest error for the rarefaction is achieved by the Roe (mesh size 0.3), HLLC (0.1), and George (0.03) solver, while for the shock wave it is accomplished by Rusanov, HLLC, and George. On the other hand, the finest grid leads to larger error values for the exact and Roe solver for all types of waves. The same holds for HLL and HLLE.



(a) Rarefaction (b) Shock

Figure 5.8.: Relative error of shock ($x \in [0.5, 0.7]$) and rarefaction $x \in [-0.7, -0.3]$ wave at $t = 0.1$ for dam break problem 5.1 calculated with Equation 4.6 and different degrees of polynomial basis functions d. The additional dotted lines mark the smallest error over all Riemann solvers. HLLE and Rusanov can capture the rarefaction and shock waves best on lower orders, respectively, while George's solver has the smallest error using $d = 3$.



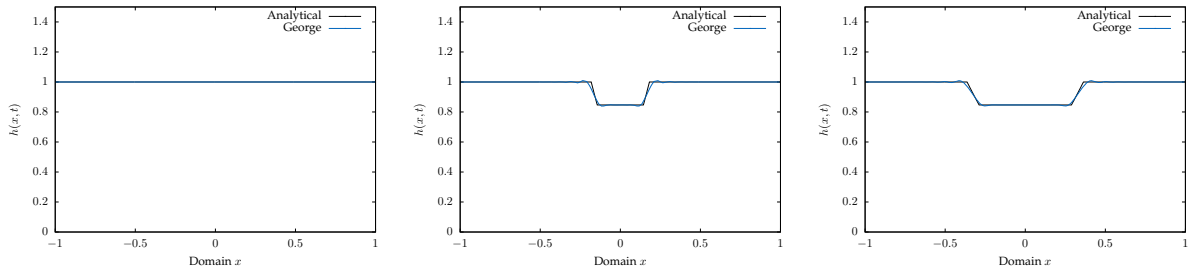
(a) Rarefaction (b) Shock

Figure 5.9.: Relative error of shock ($x \in [0.2, 0.4]$) and rarefaction $x \in [-0.5, -0.1]$ wave at $t = 0.05$ for dam break problem 5.1 calculated with Equation 4.6 and different mesh sizes. The additional dotted lines mark the smallest error over all Riemann solvers. George's augmented solver profits most from a finer mesh, while the Roe and exact solver have a larger error on all types of waves.

5.1.2. Two Rarefactions Dam Break

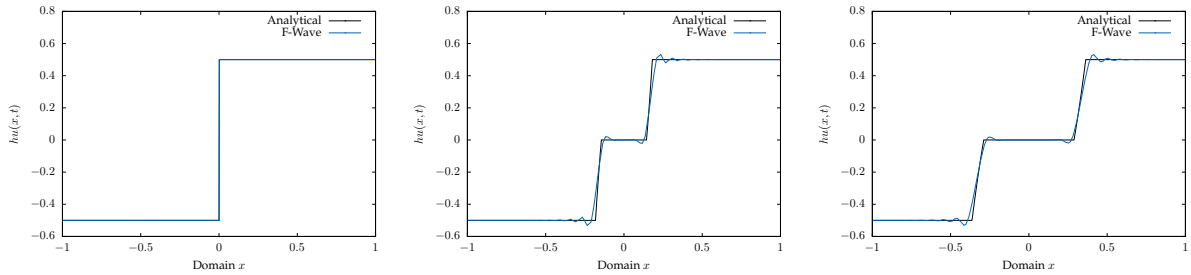
Before adding bathymetry to the simulations, similar experiments as in the previous subsection are performed using a two rarefactions dam break scenario, as found in LeVeque [30, chap. 13]:

$$q(x,y,0) = \begin{cases} \begin{pmatrix} 1 \\ -0.5 \\ 0 \end{pmatrix} & x < 0 \\ \begin{pmatrix} 1 \\ 0.5 \\ 0 \end{pmatrix} & x > 0 \end{cases}. \tag{5.2}$$



(a) $t = 0$ (b) $t = 0.05$ (c) $t = 0.10$

Figure 5.10.: Analytical solution of h for Equation 5.2 for fixed $y = 0$. Two rarefaction waves propagate after release of the discontinuity. Numerical results are calculated using George's Riemann solver along $x \in [-1, 1]$ in ExaHyPE 2 with ADER-DG and cubic polynomials. Gibb's tye oscillations occur but are less present due to the problem's wave structure.

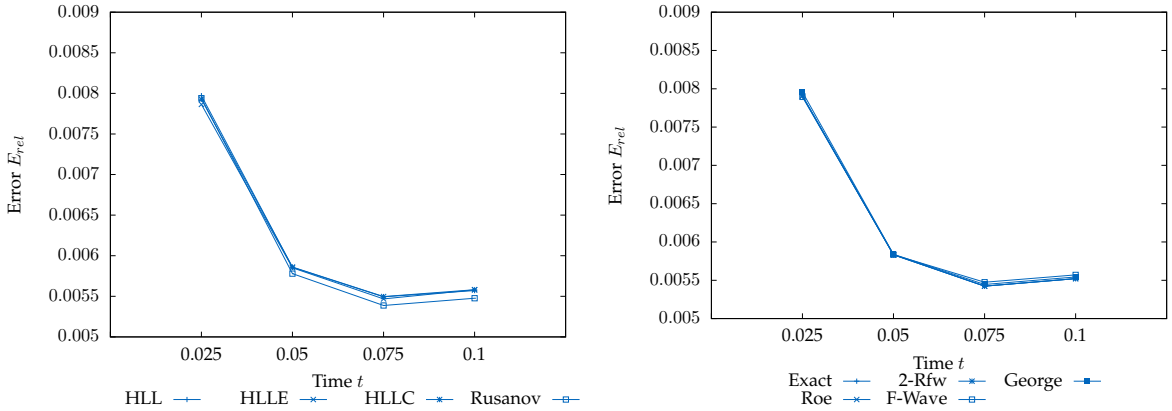


(a) $t = 0$ (b) $t = 0.05$ (c) $t = 0.10$

Figure 5.11.: Analytical solution of hu for Equation 5.2 for fixed $y = 0$. Numerical results using the f-Wave Riemann solver are computed along $x \in [-1, 1]$ in ExaHyPE 2 with ADER-DG and cubic polynomials. All Riemann solvers exhibit fewer oscillations due to the problem's wave structure.

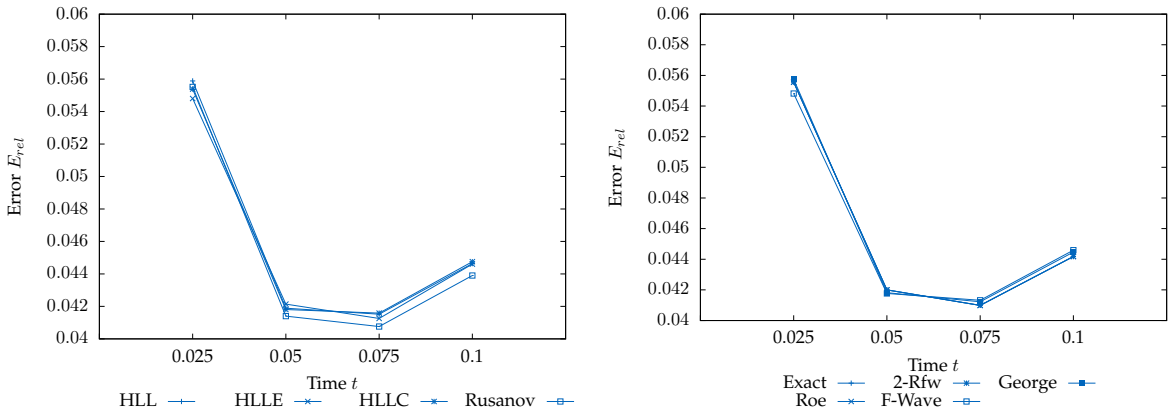
Figure 5.10 and 5.11 show the analytical together with the numerical solution for h and hu: Two rarefactions propagate through the domain with symmetric momentum. Due to the wave structure and the lower water height, fewer oscillations disturb the numerical solutions, as the simulations using the George and f-Wave Riemann solvers show.

The relative error calculated analogously to the previous section over the entire x domain is shown in Figure 5.12 for h and in 5.13 for hu: The differences in the results among all Riemann solvers are hardly recognizable here.



(a) HLL family (b) Other solvers

Figure 5.12.: Relative error of the two rarefactions dam break scenario 5.2 calculated with Equation 4.6 for unknown h along $x \in [-1, 1], y = 0$ over selected points in time. All solvers result in similar error values. The y axis in the plots was rescaled to make differences more visible.

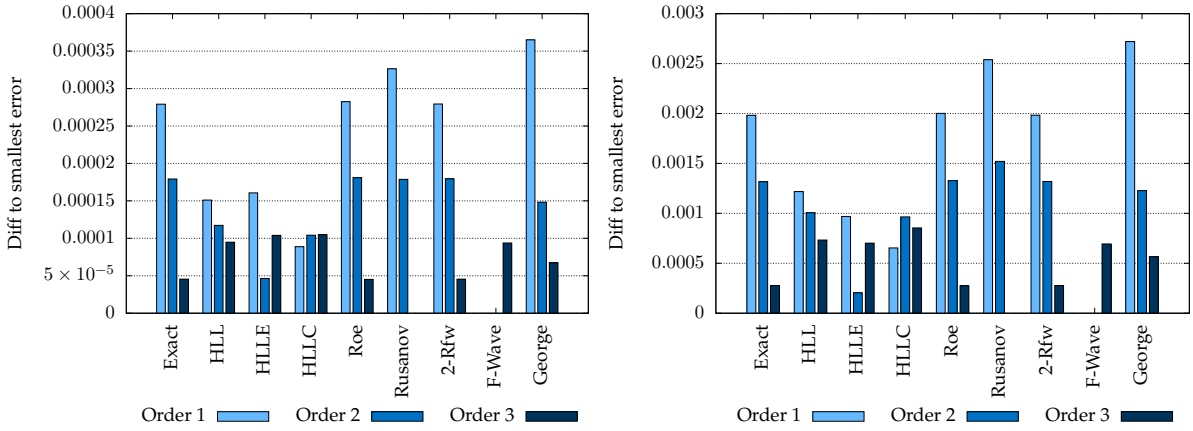


(a) HLL family (b) Other solvers

Figure 5.13.: Relative error of the two rarefactions dam break scenario 5.2 calculated for unknown hu along $x \in [-1, 1], y = 0$ over selected points in time using Equation 4.6. The y axis in the plots was rescaled to make differences more visible.

However, an analysis of the underlying data reveals that the Rusanov solver achieves the least error starting from $t = 0.05$ while HLLC yields the largest one. In contrast to the one-rarefaction one-shock dam break scenario, George's solver doesn't perform best in this case. f-Wave, on the other hand, gives a smaller error value than before. However, it should be noted that the difference between the values approximately adds up to 0.0002 and can thus be regarded as nearly negligible.

For $t = 0.1$ a comparison with different order is performed in Figure 5.14 both for h and hu. Due to the tiny variations within the solvers, this figure shows the differences to the respective smallest error. In contrast to the previous simulations, $d = 2$ now results in an error reduction for any solver. The f-Wave solver has the least error for $d = 1, 2$ for both h and hu. Regarding $d = 3$, the Rusanov solver reaches the least error for both unknowns. The exact solution assuming two rarefaction waves has an overall small error.



(a) h (b) hu

Figure 5.14.: Differences to the respective smallest relative error over the full x-axis at $t = 0.1$ for dam break problem 5.2 calculated with Equation 4.6 and varying degree of polynomial basis functions d. The smaller the bar, the better the results. HLLC and f-Wave can reproduce the analytical result well on lower orders, even though the differences remain tiny.

As an intermediate conclusion, the following can be drawn for both dam break scenarios and the homogeneous shallow water equations: With ADER-DG and order $d = 3$, HLL solvers can provide decent results despite their incompleteness. Due to their diffuse properties, they also mitigate the effects of Gibb's oscillations near discontinuous regions without limiters. In particular, they still perform well over lower polynomial degrees ($d = 1, 2$). Augmented solvers like George's or exact solvers give satisfying results especially when considering $d = 3$. In fact, the two-rarefactions exact solver behaves closely to the true Riemann solution, which confirms observations in literature [45], and thus can be seen as a reasonable complete alternative. Shock waves, however, cannot be adequately represented in the created numerical scheme with every solver, even though solvers like HLLC or George were designed to capture them accurately. This results in a decrease in the overall accuracy and should be considered in further studies. On the other hand, if a simulation such as the two rarefactions dam break scenario is considered, which is less sensitive to oscillations, only marginal dif-

ferences between the solvers can be observed. In particular, these are barely perceptible at higher orders. Similar is observed when coarser or finer grids are investigated, even though isolated solvers result in smaller errors. Therefore, the role of the Riemann solver in such settings seems to play a less important role when only accuracy is considered.

5.1.3. Well-Balancedness

An essential property of a numerical scheme for shallow water equations is the capability to maintain well-balancedness. That means that no or at least non-increasing artificial numerical waves are involved throughout the simulation, which could blow up the result even if no change is expected from the analytical solution. Well-balancedness is not only a property of a Riemann solver but rather a desired feature for the entire scheme created by ExaHyPE 2. Therefore, the lack of well-balancedness cannot be attributed to the implemented solvers alone. To investigate which Riemann solvers in the context of ExaHyPE's generated code end up in a well-balanced simulation, Scenario 5.3 is considered.

$$b = \frac{1}{4} e^{-40((x-0.5)^2+(y-0.5)^2)}, \eta = 0.5, hu = hv = 0 \tag{5.3}$$

Here, the water level $\eta = h + b$ is constant over the full domain $[0, 1] \times [0, 1]$. A smooth hump in the bathymetry b is set in the center of the domain. Apart from that, the ocean is at rest ($hu = hv = 0$), and the analytical solution should remain constant over the full runtime. A sketch of this scenario can be found in Figure 5.15.

It should be noted that this scenario involves varying topography, but does not have dry initial states. These properties are further investigated in the following section. Figure 5.16 shows the maximum deviation of momentum in x-direction for $t = 0$ to $t = 1$ s. Instabilities were observed for the Roe Riemann solver, which might be led back to the lack of preserving positivity - Thus, it is missing in the comparison and can be considered ill-balanced in the created scheme. The other solvers yield different values: For example, the f-Wave and HLLC solvers are the least well-balanced, while the other types of solvers result in similar (small) deviations. However, none of the solvers is able to keep the numerical solution strictly well-balanced, as the trend points upward for all of them, even though in a logarithmic way.

One way that can circumvent this issue is reformulating the equations as has been done using ExaHyPE 1 [38]. Here, Rannabauer states a modification of Equation 2.68, which allows for cancellation of the involved terms after spatial discretization:

$$q_t + \begin{pmatrix} hu \\ hu^2 \\ huv \\ 0 \end{pmatrix}_x + \begin{pmatrix} hv \\ huv \\ hv^2 \\ 0 \end{pmatrix}_y = \begin{pmatrix} 0 \\ gh(h+b)_x \\ gh(h+b)_y \\ 0 \end{pmatrix}. \tag{5.4}$$

Implementing scheme 5.4 shows well-balancedness for the George, Rusanov, and HLL Riemann solver.

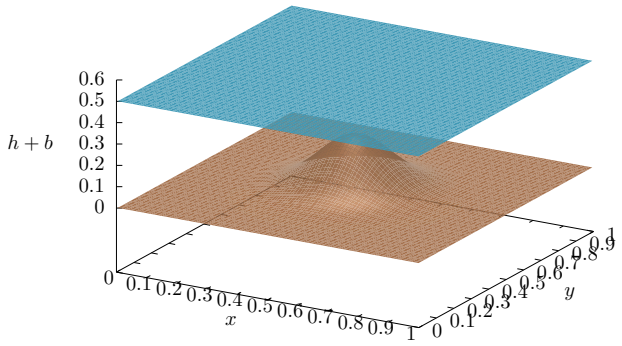
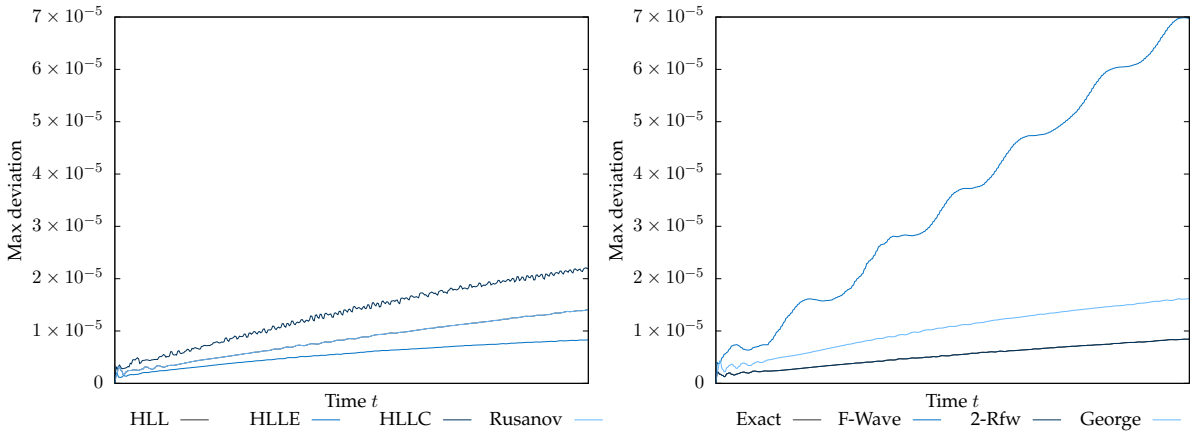


Figure 5.15.: Test scenario for well-balancedness using ExaHyPE 2 with ADER-DG and cubic basis polynomials together with reflecting boundary conditions. The numerical solution should keep the ocean at rest for the entire simulation without the appearance of spurious waves.



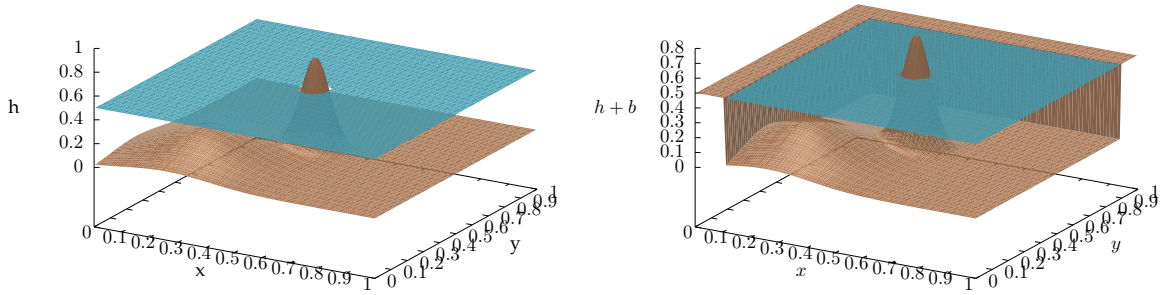
(a) HLL family

(b) Others

Figure 5.16.: Maximum deviation of hu over the full domain $[0,1]^2$ from $t = 0$ to $t = 1$ s for the test scenario for well-balancedness using ExaHyPE 2 with ADER-DG and cubic basis polynomials. Not every solver is capable of maintaining stability of the scheme, even though a logarithmic growth seem to happen in most solvers. HLL and Rusanov coincide as well as the exact and two rarefactions exact solver. Equation 5.4, on the other hand, results in balancedness for the George, Rusanov, and HLL Riemann solver.

5.1.4. Dry States and Discontinuous Bathymetry

Dealing with initial dry states and keeping well-balancedness at the same time is a challenging task for both an ADER-DG setting and its Riemann solver [35]. However, it is a crucial step towards tsunami simulations and real-life applications. Therefore, two test scenarios were developed throughout this work that help in preparing such settings. Figure 5.17a shows the first problem: It is an extension to Equation 5.3, where the smooth initial hump now involves dry regions but still has parts where the underwater topography is unequal to zero. 5.17b adds a discontinuous coastal region to the setup. Appendix C.1 states the initial values, which can be used as a reference for future work.



(a) Test problem 1

(b) Test problem 2

Figure 5.17.: Test problems for well-balancedness involving dry states and discontinuous coastal regions. Handling these settings is a demanding task for numerical simulations and Riemann solvers, especially when using Discontinuous Galerkin methods. A correct numerical solution is able to maintain the ocean at rest without spurious waves.

The demands on a numerical scheme solving these scenarios are obvious: No numerical waves are allowed to be added, or at least they should remain constant and not blow up the results over time. However, without modifications to the problem definition as well as the Riemann solvers, these waves might occur in the coastal regions, destabilize the result, and produce wrong solutions. Even worse, the discontinuous beach can immediately break the simulation. Initially, these problems were observed for all implemented Riemann solvers and any form of the shallow water equations. To make the simulation work again, several steps are involved:

- The formulation 3.44 without source term is implemented. Even though the assumption of a smooth topography is not fulfilled, this equation can be used if the transition from coast to sea in 5.17b is considered separately.
- Following the code of Breuer and Bader [4], the Riemann solver applies reflecting boundary conditions if the left or right side is dry (i.e., $h_l := h_r, hu_l = -hu_l, hv_l = -hv_l, b_l = b_r$ for dry left side). Initially, this excludes inundation scenarios.

- The dry regions are defined over a threshold (e.g., $h <= 0.01$ implies a dry node) and are aligned to cell interfaces before simulation start. That means that there can not arise semi-dry regions within a single cell. Or in other words, dry cells degenerate to Finite Volume cells. This idea is inspired by Reinartz et al. using ExaHyPE 1 [39]. Though this approach coarsens the resolution along the coast, a way out could be to refine the grid where it is necessary. This approach is visualized in Figure 5.18.
- Wet cells are allowed to dry out at runtime: If one quadrature node of a cell is below the defined threshold, the complete cell is set to $h = 0$. This can be seen as a correction step to regions that were initially almost dry. However, it should be noted that this is a unphyiscal step, as it violates the conservation of mass.

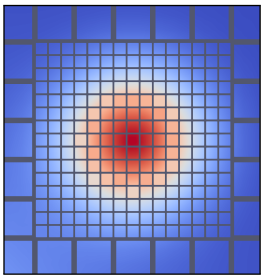
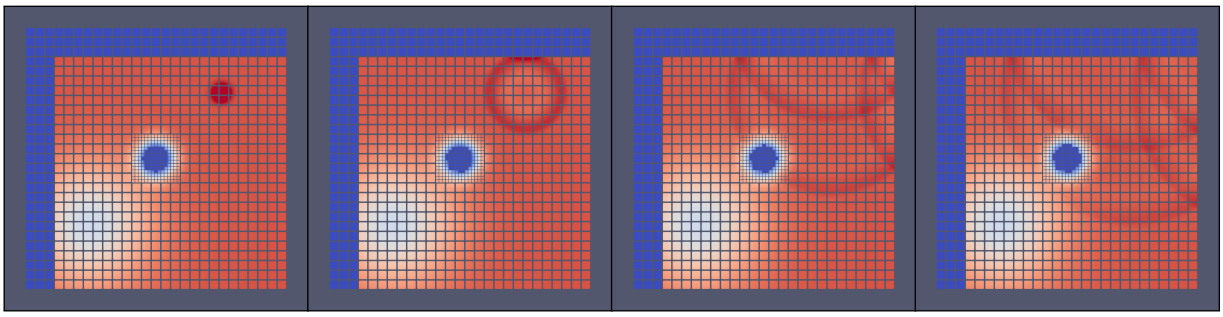


Figure 5.18: Bathymetry b for test problems 5.17 on a refined grid along the central coastal region. To enable a stable numerical simulation, cells are entirely set to dry throughout the simulation, if the water height falls below a defined threshold. This approach can restore the initial resolution of the bathymetry.

The approach runs successfully using the HLL-type Riemann solvers. Further complexity arises when a hump of water is added around $(0.75, 0.75)$ as Figure 5.19 shows. Here, the HLLE Riemann solver demonstrates functionality as well. However, due to the changed semi-discrete form and conserved quantities from Equation 3.44 compared to 2.68 it does not work without further modification to Riemann solvers, which are based on the eigendecomposition of the system. The experiments emphasize one advantage of the HLL-based solvers: they can be quickly adapted to different scenarios, while still providing reasonable results in a setting using adaptive mesh refinement.



(a) $t = 0$ (b) $t = 0.05$ (c) $t = 0.15$ (d) $t = 0.20$

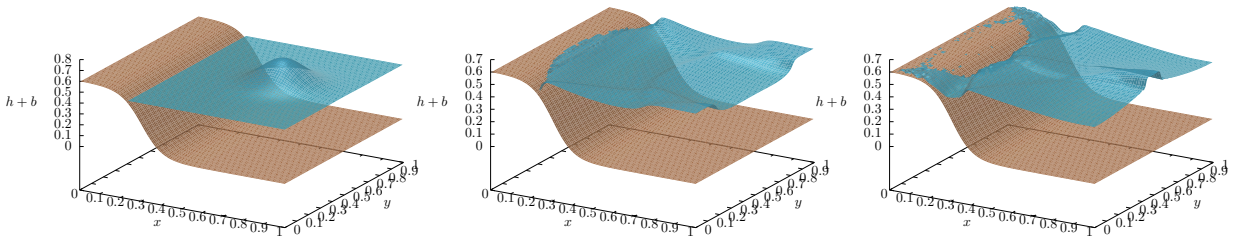
Figure 5.19.: Numerical solution of h for Problem 5.17b with an additional hump of water in the top right corner with reflecting boundary conditions in ExaHyPE 2 using ADER-DG, cubic polynomials, and the HLLE Riemann solver. The waves hit coastal regions, which are aligned to the cells of the simulations. The smooth landscape in the center of the domain is refined to preserve the resolution of the bathymetry.

5.1.5. Inundation Scenarios

In the previous problems, reflecting boundary conditions and aligning dry regions with cell boundaries prevented areas from being flooded during the simulation. A major problem with the latter types of simulations are the underlying assumptions of the shallow water equations: For example, the assumption of a small vertical length scale compared to the horizontal one is not necessarily satisfied along the coast, as Ortleb et al. point out [35]. The same is true for the presence of a hydrostatic pressure distribution, even though the equations can still capture many underlying physical effects well enough. Therefore, wetting and drying scenarios are outlined in this section using the analytical dry state solution for all Riemann solvers from Section 3.1.5. However, it should be noted that George's Riemann solver has been successfully tested for wetting and drying scenarios not employing this approach as well [17] and there are also modifications to the wave estimates used in the HLLC solver that enables these processes [23].

Here, the thin-layer approach that was already sketched in the previous section is updated to realize these ideas together with Equation 2.68: A threshold for the water height marks dry or wet regions. However, instead of now applying reflecting boundary conditions in the Riemann solver for dry/wet side combinations, the Riemann problem is solved analytically with the restriction that flux of momentum is not allowed to dry regions. In the implementation, FL[i] or FR[i] is set to zero, respectively. Furthermore, the preprocessing step is added to ExaHyPE 2's kernels that corrects negative water heights to $h = 0$, which may arise due to the created numerical scheme.

To test the implementation, Equation C.3 from the appendix is considered first: A hump of water is placed close to a smooth sloping beach on the domain $\Omega = [0, 1]^2$. Results are plotted in Figure 5.20 for $\eta = h + b$: The coast is flooded over time and dried out again in the simulation. It should be noted that this experiment is not checked for correctness, but rather for stable and reasonable results. Furthermore, is has to be added that several changes are applied that break the properties of a conservation law, which have to be kept in mind: For example, the conservation of momentum is clearly violated, as the flux is constrained.



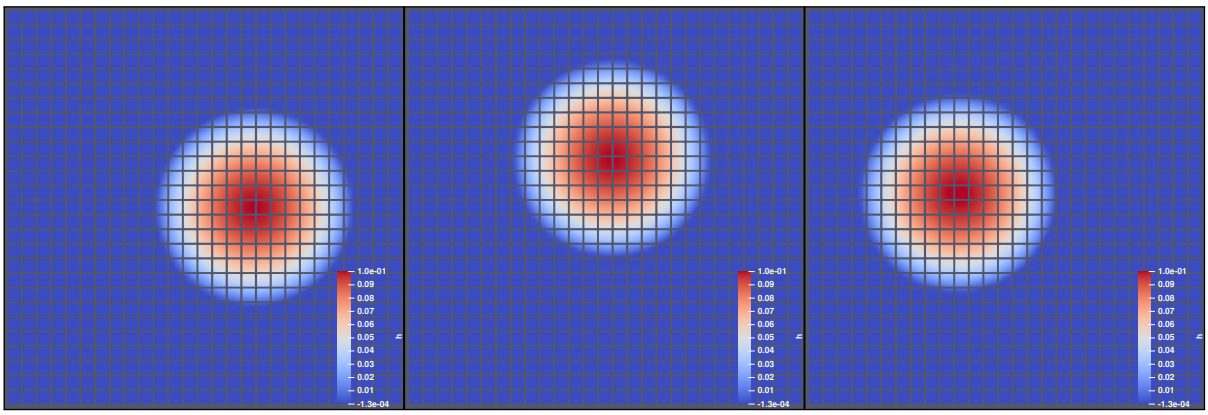
(a) $t = 0$ (b) $t = 0.25$ (c) $t = 0.4$

Figure 5.20.: Numerical results of problem C.3 using ExaHyPE 2 with ADER-DG, cubic basis polynomials, and the exact Riemann solver based on a thin-layer approach. A wave is flooding a smooth coastal region. ExaHyPE 2's structure is flexible enough to create a stable simulation for wetting and drying scenarios.

A scenario that allows for verification of the approach as well as evaluation of the Riemann solver involving wetting and drying is the so-called oscillating lake scenario taken from Reinartz et al. [39] and Rannabauer [38] based on previous work [41][1]. Here, a droplet of water is released in a dry basin in $\Omega = [-2, 2]^2$ and is expected to travel circularly in it. The initial conditions together with the analytical solution are given in Equation 5.5.

$$q(x, y, t) = \begin{pmatrix} h \\ hu \\ hv \\ b \end{pmatrix} = \begin{pmatrix} \max(0, 0.1(x\cos(\omega t) + y\sin(\omega t) + \frac{3}{4}) - b) \\ \frac{1}{2}\omega\sin(\omega t)h \\ \frac{1}{2}\omega\cos(\omega t)h \\ \frac{1}{10}(x^2 + y^2) \end{pmatrix}, \tag{5.5}$$

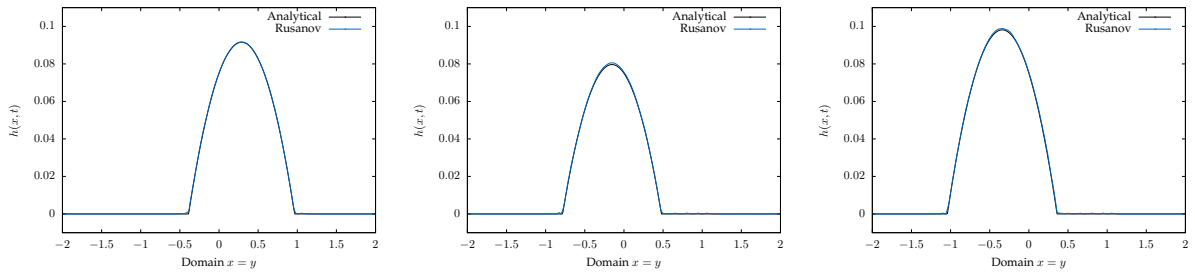
where $\omega = \sqrt{0.2g}$ and $g = 9.81$.The simulation using the HLLE Riemann solver is plotted in Figure 5.21: The initial lake oscillates through the basin and tends towards a steady state in the center. The simulation parameters are ADER-DG using cubic polynomials, a mesh size of 0.3, and a threshold of 0.0005 for dry regions. Not visible is the small layer of water that naturally remains due to the numerical scheme when traversing.



(a) $t = 0$ (b) $t = 1$ (c) $t = 2$

Figure 5.21.: Oscillating lake scenario 5.5 simulated in ExaHyPE 2 using ADER-DG, cubic polynomials, and the HLLE Riemann solvers at different points in time. The constant drying and wetting is a challenging problem for numerical schemes and involves several difficulties: Here, a thin-layer approach is employed with $h = 0.0005$ as a threshold to ensure the stability of the scheme.

Out of all tested solvers, only HLL, Rusanov, and HLLE converged at $t = 3$. Roe and George operated until 0.2 and 0.12 before diverging, respectively. All other solvers diverged almost immediately after the start ($t = 0.02$), revealing that they are not stable in the created scheme. The analytical solution over $x = y$ is plotted in Figure 5.22 together with the numerical solution using the Rusanov solver: Differences are barely visible in the plot due to the high accuracy. However, oscillations around the wet/dry font arise and strengthen throughout the simulation (compare, for example, $x = 1$ in $t = 1$ and $x = 0.5$ in $t = 3$). A zoomed variant can be found in Appendix C.2, supporting this statement.



(a) $t = 1$ (b) $t = 2$ (c) $t = 3$

Figure 5.22.: Analytical and numerical solution along $x = y$ for Problem 5.5 for selected points time. The simulation with ExaHyPE 2 employs ADER-DG, cubic polynomials, a fixed mesh of size 0.3, and the Rusanov Riemann solver. Differences between both solutions are barely notable, but oscillations start to spread around $t = 3$.

HLLE and HLL have similar accuracy, which is relatable to the similarity of the three solvers. This claim is emphasized using Figure 5.23, showing the relative error for $x = y$ over the full domain. The values are computed point wise with a non-equidistant grid, similar to the previous sections. Nevertheless, HLLE performs slightly better than all other solvers at all points in time, but $t = 1.5$. Rusanov and HLL alternate regarding the error: HLL results in the higher values for $t \in \{0.5, 1, 3\}$.

The following conclusions may be drawn from the results: HLL-type solvers are capable of realizing flooding scenarios in ExaHyPE 2 without complex modifications to the created numerical scheme with reasonable accuracy. Other solvers, such as exact, linearized, or augmented ones, require additional adjustments. For example, in ExaHyPE 1, Finite Volume limiting together with adaptive mesh refinement was used in cells with low water height [39] [38]. However, by the time of this thesis, these features are not yet available and should be addressed in future studies once the features are available.

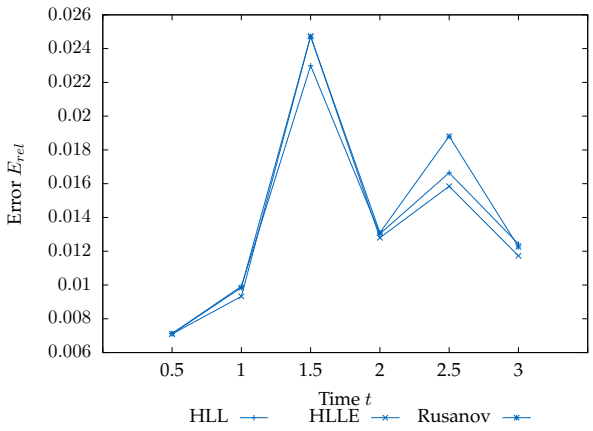
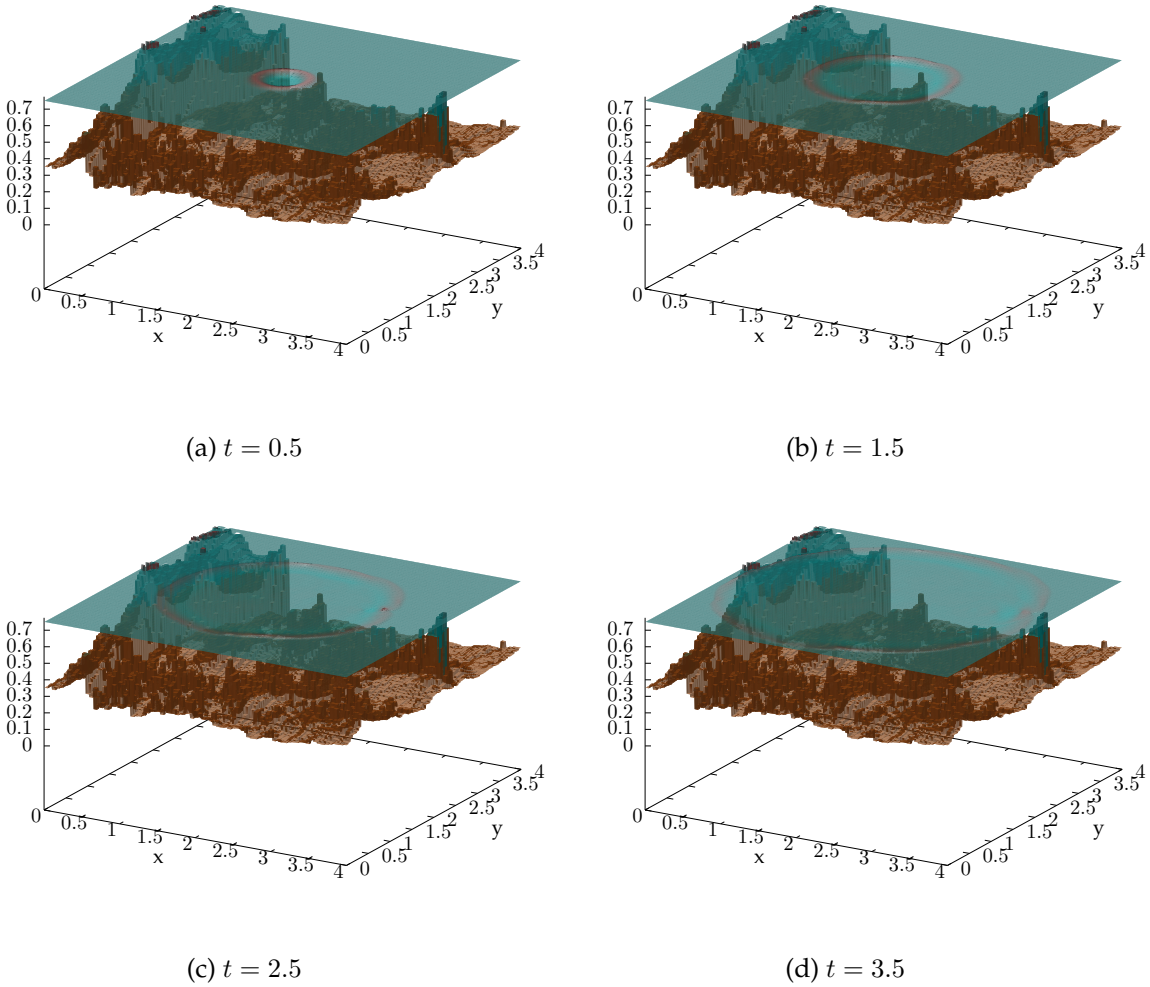


Figure 5.23.: Relative error for Problem 5.5 calculated over $x = y$ in $\Omega = [-2, 2]^2$ for selected points time using ExaHyPE 2 with ADER-DG, cubic polynomials, a fixed mesh of size 0.3, and various Riemann solvers.

Together with the previous section, ExaHyPE 2 and the HLL Riemann solvers are theoretically suited to perform tsunami simulations. A starting point for this can be found in Figure 5.24: Here, the hump of water is released into the domain of the Tohoku earthquake from 2011. The HLLE solver with reflecting boundary conditions from Section 5.1.4 was taken as implementation. The results show, on the one hand, that a well-balanced scheme with complex topography is obtained and, on the other hand, that a solution to the Riemann problem with initial dry states is possible. A following task could be to implement and verify the actual tsunami waves. However, since this exceeds the scope of this paper, it is considered as a desirable addition for future work.



(a) $t = 0.5$

(b) $t = 1.5$

(c) $t = 2.5$

(d) $t = 3.5$

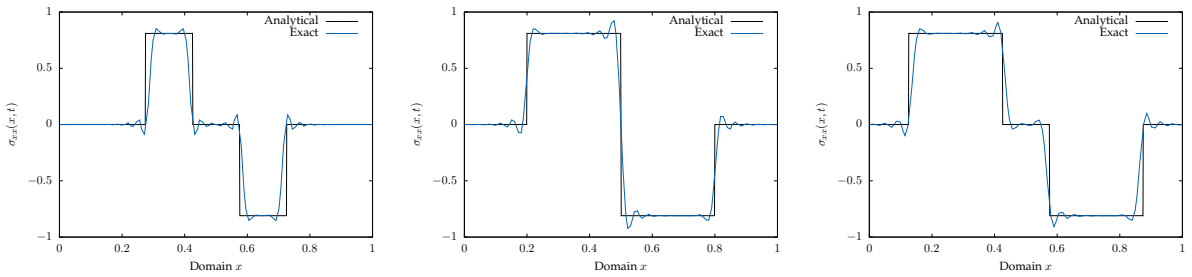
Figure 5.24.: Numerical results of an artificial tsunami simulation based on the 2011 Tohoku earthquake in the east of Japan using the results from Section 5.1.4 and 5.1.5. ADER-DG with cubic polynomials and the HLLE solver was used. The results show that the Riemann solvers together with the modifications on the numerical scheme can produce well-balanced stable simulation on larger domains with complex bathymetry.

5.2. Elastic Wave Equations

This section briefly discusses the three-dimensional linear elastic wave equations using two constant-coefficient test problems. In three dimensions, a call of riemannSolver(...) in ExaHyPE 2 needs to solve all Riemann problems between a left and a right face (and not a left and a right edge as it was required when considering the shallow water equations). In addition, there are three dimensional-split formulations that have to be taken into account in the implementation. Equation 5.6 shows the first setting in a cubical domain $\Omega = [0, 1]^3$, resembling a two-sided dam break problem. The Lamé parameters are set constant.

$$\begin{aligned} \sigma_{xx} = \sigma_{yy} = \sigma_{zz} = \sigma_{xy} = \sigma_{xz} = \sigma_{yz} = v = w = 0, \\ u = \begin{cases} 0.1 & \text{, if } |x - 0.5| < 0.15 \\ 0 & \text{otherwise} \end{cases}, \\ \rho = 2.7, c_s = 3.464, c_p = 6.0. \end{aligned} \tag{5.6}$$

The problem is examined using the Rusanov, HLL, and an exact Riemann solver following Section 2.2.1 using the eigendecomposition presented in 2.5.2 for each dimensional-split formulation. Similar to the previous section, the comparison is drawn along the x - axis. Here, Figure 5.25 shows the results for σ_{xx} at different points in time using ADER-DG and cubic polynomials using the exact Riemann solver and for u in Figure 5.26 with HLL. The analytical result is plotted for comparison as well. Gibb's type oscillations occur, similar to what was observed in Section 5.1.1.



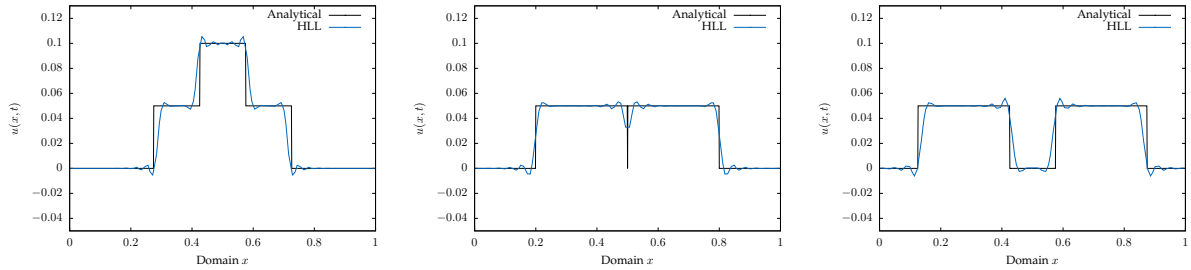
(a) $t = 0.0125$ (b) $t = 0.0250$ (c) $t = 0.0375$

Figure 5.25.: Analytical and approximated solution of σ_{xx} for Equation 5.6. The numerical simulation uses the exact Riemann solver. Results are shown for fixed $y = 0$ along $x \in [-1, 1]$ calculated with ExaHyPE 2 employing ADER-DG and cubic polynomials.

However, a point-wise error analysis at $t = 0.1$ over the full axis similar to the previous sections reveals vanishing differences of 10^{-6} among all three solvers with respect to the analytical solution. Thus, this type of problem does not permit further insights regarding the evaluation of the solvers but indicates correctness in more than two dimensions and the universal applicability of Rusanov and HLL. A more sophisticated version of 5.6, which creates changes in all three directions, is obtained by setting the initial values to the Form 5.7.

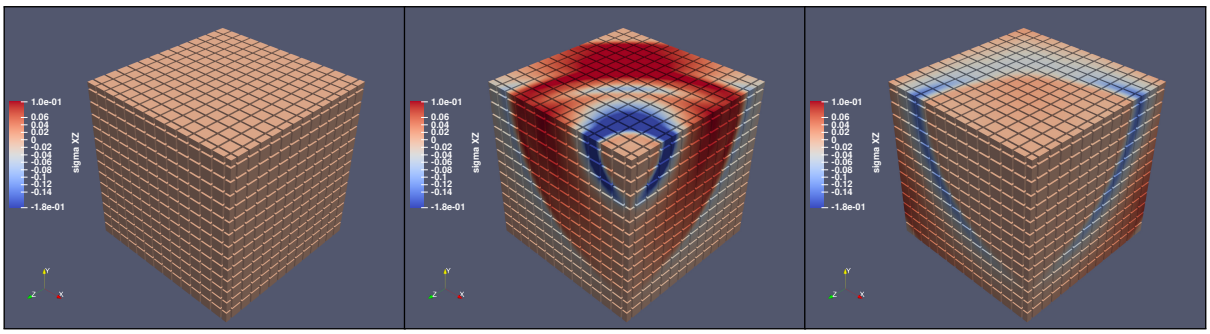
$$\begin{pmatrix} u(x,y,z,0) \\ v(x,y,z,0) \\ w(x,y,z,0) \end{pmatrix} = \begin{pmatrix} 0.1 \\ 0.1 \\ 0.1 \end{pmatrix}, \text{if } || \begin{pmatrix} x-0.5 \\ y-0.5 \\ z-0.5 \end{pmatrix} ||_2 < 0.15. \tag{5.7}$$

Numerical solutions for σ_{xz} are shown in a clipped three-dimensional domain in Figure 5.27 using the Rusanov, exact, and HLL Riemann solver. Even though a comparison using the analytical solution was not performed, the similarity of all solutions for all points in time indicates that the implemented solvers produce correct results. This confirms that the implementation strategy outlined in Section 4.2 works correctly in three dimensions as well. However, a significantly increased runtime compared to the shallow water equations was observed during execution of the simulation, confirming the assumption made in Section 4.2 regarding the scaling of the performance. Some general notes about the complexity of the solvers are outlined in the next section.



(a) $t = 0$ (b) $t = 0.05$ (c) $t = 0.10$

Figure 5.26.: Analytical and approximated solution of u for Equation 5.6. The numerical simulation uses the HLL Riemann solver. Results are shown for fixed $y = 0$ along $x \in [-1, 1]$ calculated with ExaHyPE 2 employing ADER-DG and cubic polynomials.



(a) $t = 0$ (Rusanov) (b) $t = 0.075$ (HLL) (c) $t = 0.15$ (Exact)

Figure 5.27.: Numerical solution of σ_{xz} for Problem 5.7 in ExaHyPE 2 using ADER-DG, cubic polynomials, and different Riemann solvers. The visual similarity of all simulations indicates a correct implementation for those solvers in three dimensions.

5.3. Notes on Performance

The comparisons among all Riemann solvers so far concerned their accuracy or qualitative properties, such as the ability to maintain well-balancedness or stability in complex scenarios. Also, it has been shown that a transition to three dimensions is feasible and yields valid results. Here, the differences in many solvers were frequently small due to the high convergence order of the ADER-DG method. Therefore, it might be incorrectly concluded that exact or complete solvers are generally preferable to the others since they resolve the wave structure better in cases of doubt. However, one aspect that has not been considered so far is the performance concerning memory and runtime of the solvers. As indicated in the introduction of this thesis, a Riemann solver is called extensively in an ADER-DG or Finite Volume setting. Thus, this property should be briefly discussed in the present section as well. A fair comparison in ExaHyPE 2 is therefore hardly possible, as the overall runtime of the generated code is determined by numerous degrees of freedom: Multithreading using CPUs and/or GPUs, linear vs. non-linear kernels, adaptive time stepping, and load balancing are only a subset of all possible parameters that have to be taken into account in performance analysis. Thus, the following discussion considering a one-dimensional Riemann problem with m equations is a descriptive complexity analysis, starting with complete solvers:

- In general, it can be noted that complete solvers tend to a linear complexity $\mathcal{O}(m \cdot c(m))$, as they scale with the size of the underlying conservation law. For each wave, an additional function $c(m)$ summarizes the effort for each of these waves. This function should be independent of m, i.e., $c(m) = const$.
- For a constant-coefficient equation, a matrix-vector product as shown in Equation 2.13 needs to be calculated, when the Riemann problem should be solved exactly. The same holds for linearized solvers such as the Roe Riemann solver, or the f-Wave approach. Depending on the sparsity of the eigendecomposition, this can usually be implemented efficiently. However, in general, the product is of quadratic complexity, i.e., $\mathcal{O}(m^2)$, which scales poorly with bigger systems when considering memory and runtime.
- A non-linear exact solver, such as derived for the shallow water equations, might involve applying a nonlinear root finder to a scalar or even vector-valued function g. Here, Newton's method is typically used, which is locally quadratically convergent for single roots, but still requires multiple evaluations of g and g', which can include large systems.
- Incomplete solvers share the encouraging property of only considering a fixed subset of waves. This implies that their complexity is independent of the wave structure, i.e. $\mathcal{O}(c)$. Two-Wave HLL solvers of the Form 3.23 and 3.26, such as HLL, HLLE, and Rusanov, enable the estimate $c \approx 1$. The latter equation is even to be preferred for use in ExaHyPE 2, as FL and FR are pre-computed before the Riemann solver is called. Together with parallelization, a very efficient scheme can be obtained. On the contrary, an open question is how the wave speeds s_1, s_2 are calculated. While Rusanov only needs one estimate based on the system's maximum eigenvalues, HLLE, for example, requires the Roe averages in addition, which might add further complexity by expensive function calls like taking the square root.

- Augmented solvers might include more than m waves for eigendecomposition. George's Riemann solver, for example, considers $m + 1$ waves for the shallow water equations, other approaches for theoretical analyses are even based on $2m$ waves [33]. This can slow down the required eigendecomposition in addition.

In summary, it can be speculated that the performance of the implemented solvers roughly behaves as follows in a sequential application (left: fastest, right: slowest):

$$\text{Rusanov} \leq \text{HLL} \leq \text{Two-Rfw} \leq \text{HLLE} \leq \text{HLLC} \leq f - \text{Wave} \leq \text{Roe} \leq \text{George} \leq \text{Exact} \quad (5.8)$$

The f-Wave solver was therefore preferred over the Roe solver as FL and FR are available on function call. Here, several aspects have been left out, including, for example, memory efficiency. A discussion of the ability to parallelize the code for GPUs and CPUs was only briefly touched, which can in practice favor worse complexities over better ones. Here, for example, the exact solver initially appears unfavorable due to the iterative root finder. At the same time, it should be noted that the experiments in the previous sections indicate that incomplete solvers can yield strong results in the experiments. These considerations are taken into account in the next chapter for the final conclusion.

6. Conclusion

ExaHyPE 2 (An Exascale Hyperbolic PDE Engine) is an open-source software for generating code, which solves hyperbolic partial differential equations (PDEs) using Finite Volumes and Discontinuous Galerkin methods while hiding numerical and implementational details. These equations enable, for example, the simulation of geophysical phenomena such as earthquakes and tsunamis, which helps in understanding and mitigating the consequences of these catastrophes. An essential part of both numerical techniques is the Riemann solver, a component that solves a PDE with a step function as initial value, the so-called Riemann problem. It is called at least once per cell in every spatial direction and, thus, influences both the performance and accuracy of the results. However, their impact in the context of ExaHyPE 2 has not yet been investigated.

This thesis implemented HLL-type Riemann solvers together with exact, augmented, and linearized ones on the example of the shallow water equations with and without bathymetry in ExaHyPE 2 and verified the correctness of the code using the linearized form of the equations. Their accuracy was discussed using a one-rarefaction, one-shock dam break scenario, showing that differences in the solvers are present but are less significant, especially when considering higher-order schemes. In smooth solutions such as the two-rarefaction dambreak scenario, where less Gibb's type oscillations occur without limiters, the variation in the error values between the solvers was tiny. Thus, computationally expensive solvers such as the exact Riemann solver do not necessarily provide better results.

Well-balancedness was investigated using different formulations of the shallow water equations with a lake at rest problem: Here, no solver could initially keep the momenta constant. However, reformulating the created scheme could preserve it for some of the solvers, showing that the choice of Riemann solver influences this essential property. Another test for well-balancedness was performed using a problem with initial dry states, both smooth and discontinuous. Several steps were involved to keep the created scheme balanced. The most impactful step was to align the wet/dry font to cells and set reflecting boundary conditions, degenerating coastal regions to finite volume grids, which could be mitigated by adaptive mesh refinement. The HLL-type solver could have been successfully employed in this setting.

Inundation scenarios were initially tested using the exact Riemann solver, showing that the dry-state Riemann problem is correctly implemented. Verification was performed using the oscillating lake scenario: Here, only the two-wave solvers showed stable results for the full simulated time while producing results with high accuracy, although the followed thin-layer approach violated assumptions on conservation. Due to the similarity of the schemes, they did not differ highly within the solvers, even though HLLE tended towards a slightly decreased error. All these ideas were finally moved together in a tsunami-like simulation, where stability and well-balancedness were shown in a setting with complex bathymetry and dry regions. Lastly, three of the solvers were adapted to the linear elastic wave equations,

showing that the implementation strategy works in three dimensions as well.

As limiters were not available for the main part of this work, their impact on the results could not be evaluated. Therefore, oscillations disturbed the investigations in this work with discontinuous solutions. For the same reason, dynamic adaptive mesh refinement could not be employed and verified. This may constitute the object of future studies. In addition, the author suggests examining strategically the HLL-type solvers, as they fit best into a generic code generation tool such as ExaHyPE. Furthermore, performance portability and potential performance optimizations should be carried out in the future and tested using tsunami simulations with real-life data.

Appendices

A. Part I

Here are the other eigenvectors for the three-dimensional elastic wave equations defined in 2.77. For matrix B they read:

$$R_{(y)} = \begin{pmatrix} \lambda & 0 & 0 & 1 & 0 & 0 & 0 & 0 & \lambda \\ \lambda+2\mu & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \lambda+2\mu \\ \lambda & 0 & 0 & 0 & 1 & 0 & 0 & 0 & \lambda \\ 0 & \mu & 0 & 0 & 0 & 0 & \mu & 0 & 0 \\ 0 & 0 & \mu & 0 & 0 & 0 & 0 & \mu & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & c_s & 0 & 0 & 0 & 0 & -c_s & 0 & 0 \\ c_p & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -c_p \\ 0 & 0 & c_s & 0 & 0 & 0 & 0 & -c_s & 0 \end{pmatrix}, \tag{A.1}$$

$$R_{(y)}^{-1} = \begin{pmatrix} 0 & \frac{1}{2(\lambda+2\mu)} & 0 & 0 & 0 & 0 & 0 & \frac{1}{2c_p} & 0 \\ 0 & 0 & 0 & \frac{1}{2\mu} & 0 & 0 & \frac{1}{2c_s} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2\mu} & 0 & 0 & 0 & \frac{1}{2c_s} \\ 1 & -\frac{\lambda}{\lambda+2\mu} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -\frac{\lambda}{\lambda+2\mu} & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2\mu} & 0 & 0 & -\frac{1}{2c_s} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2\mu} & 0 & 0 & 0 & -\frac{1}{2c_s} \\ 0 & \frac{1}{2(\lambda+2\mu)} & 0 & 0 & 0 & 0 & 0 & -\frac{1}{2c_p} & 0 \end{pmatrix}. \tag{A.2}$$

And for matrix C:

$$R_{(z)} = \begin{pmatrix} \lambda & 0 & 0 & 1 & 0 & 0 & 0 & 0 & \lambda \\ \lambda & 0 & 0 & 0 & 1 & 0 & 0 & 0 & \lambda \\ \lambda+2\mu & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \lambda+2\mu \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & \mu & 0 & 0 & 0 & 0 & \mu & 0 \\ 0 & \mu & 0 & 0 & 0 & 0 & \mu & 0 & 0 \\ 0 & c_s & 0 & 0 & 0 & 0 & -c_s & 0 & 0 \\ 0 & 0 & c_s & 0 & 0 & 0 & 0 & -c_s & 0 \\ c_p & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -c_p \end{pmatrix}, \tag{A.3}$$

$$R_{(z)}^{-1} = \begin{pmatrix} 0 & 0 & \frac{1}{2(\lambda+2\mu)} & 0 & 0 & 0 & 0 & 0 & \frac{1}{2c_p} \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{2\mu} & \frac{1}{2c_s} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2\mu} & 0 & 0 & \frac{1}{2c_s} & 0 \\ 1 & 0 & -\frac{\lambda}{\lambda+2\mu} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -\frac{\lambda}{\lambda+2\mu} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{2\mu} & -\frac{1}{2c_s} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2\mu} & 0 & 0 & -\frac{1}{2c_s} & 0 \\ 0 & 0 & \frac{1}{2(\lambda+2\mu)} & 0 & 0 & 0 & 0 & 0 & -\frac{1}{2c_p} \end{pmatrix}. \tag{A.4}$$

For future work, the two-dimensional form is given as well. The equation reads

$$q_t + A(q)q_x + B(q)q_y = 0, \tag{A.5}$$

with

$$A = \begin{pmatrix} 0 & 0 & 0 & -(\lambda+2\mu) & 0 \\ 0 & 0 & 0 & -\lambda & 0 \\ 0 & 0 & 0 & 0 & -\mu \\ -\frac{1}{\rho} & 0 & 0 & 0 & 0 \\ 0 & 0 & -\frac{1}{\rho} & 0 & 0 \end{pmatrix}, B = \begin{pmatrix} 0 & 0 & 0 & 0 & -\lambda \\ 0 & 0 & 0 & 0 & -(\lambda+2\mu) \\ 0 & 0 & 0 & -\mu & 0 \\ 0 & 0 & -\frac{1}{\rho} & 0 & 0 \\ 0 & -\frac{1}{\rho} & 0 & 0 & 0 \end{pmatrix}. \tag{A.6}$$

.

The eigenvalues are given as

$$\lambda \in \{-c_p, -c_s, 0, c_s, c_p\} \tag{A.7}$$

and the eigenvectors

$$R_{(x)} = \begin{pmatrix} \lambda+2\mu & 0 & 0 & 0 & \lambda+2\mu \\ \lambda & 0 & 1 & 0 & \lambda \\ 0 & \mu & 0 & \mu & 0 \\ c_p & 0 & 0 & 0 & -c_p \\ 0 & c_s & 0 & -c_s & 0 \end{pmatrix}, R_{(x)}^{-1} = \begin{pmatrix} \frac{1}{2(\lambda+2\mu)} & 0 & 0 & \frac{1}{2c_p} & 0 \\ 0 & 0 & \frac{1}{2\mu} & 0 & \frac{1}{2c_s} \\ -\frac{\lambda}{\lambda+2\mu} & 1 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2\mu} & 0 & -\frac{1}{2c_s} \\ \frac{1}{2(\lambda+2\mu)} & 0 & 0 & -\frac{1}{2c_p} & 0 \end{pmatrix}, \tag{A.8}$$

$$R_{(y)} = \begin{pmatrix} \lambda & 0 & 1 & 0 & \lambda \\ \lambda+2\mu & 0 & 0 & 0 & \lambda+2\mu \\ 0 & \mu & 0 & \mu & 0 \\ 0 & c_s & 0 & -c_s & 0 \\ c_p & 0 & 0 & 0 & -c_p \end{pmatrix}, R_{(y)}^{-1} = \begin{pmatrix} 0 & \frac{1}{2(\lambda+2\mu)} & 0 & 0 & \frac{1}{2c_p} \\ 0 & 0 & \frac{1}{2\mu} & \frac{1}{2c_s} & 0 \\ 1 & -\frac{\lambda}{\lambda+2\mu} & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2\mu} & -\frac{1}{c_s} & 0 \\ 0 & \frac{1}{2(\lambda+2\mu)} & 0 & 0 & -\frac{1}{2c_p}. \end{pmatrix}. \tag{A.9}$$

B. Part II

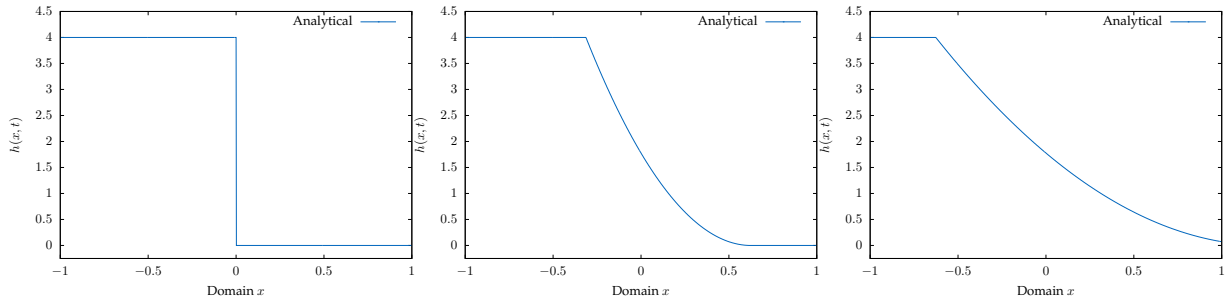
Example file for ExaHyPE 2 used in Section 4.2. Note that this file is valid at the time of writing this work and may be the object of future changes.

```
1
2  import peano4
3  import exahype2
4
5  project = exahype2.Project( ["examples", "exahype2", "tutorial"], "out", ".",
       executable="OUT" )
6
7  thesolver = exahype2.solvers.aderdg.rusanov.GlobalAdaptiveTimeStep(
8    "out",
9    order=3,
10   unknowns=4, auxiliary_variables=0,
11   min_cell_h=0.1, max_cell_h=0.1,
12   time_step_relaxation = 0.9
13 )
14
15 thesolver.set_implementation(refinement_criterion=exahype2.solvers.fv.PDETerms.
       User_Defined_Implementation)
16 project.add_solver( thesolver )
17
18 project.set_global_simulation_parameters(
19   dimensions = 2,
20   offset = [0,0],
21   size = [1,1],
22   min_end_time = 1.0,
23   max_end_time = 1.0,
24   first_plot_time_stamp = 0.0,
25   time_in_between_plots = 0.005,
26   periodic_BC = [False, False, False]
27 )
28
29 build_mode = peano4.output.CompileMode.Release
30 project.set_load_balancing( "toolbox::loadbalancing::RecursiveBipartition", "new
       ::exahype2::LoadBalancingConfiguration()" )
31 project.set_Peano4_installation( "../../", build_mode )
32 peano4_project = project.generate_Peano4_project(False)
33
34 peano4_project.build(make_clean_first=True, number_of_parallel_builds=4)
```

Figure B.1.: Example problem for ExaHyPE 2.

Derivation of an alternative form of the shallow water equations including bathymetry in homogeneous form.

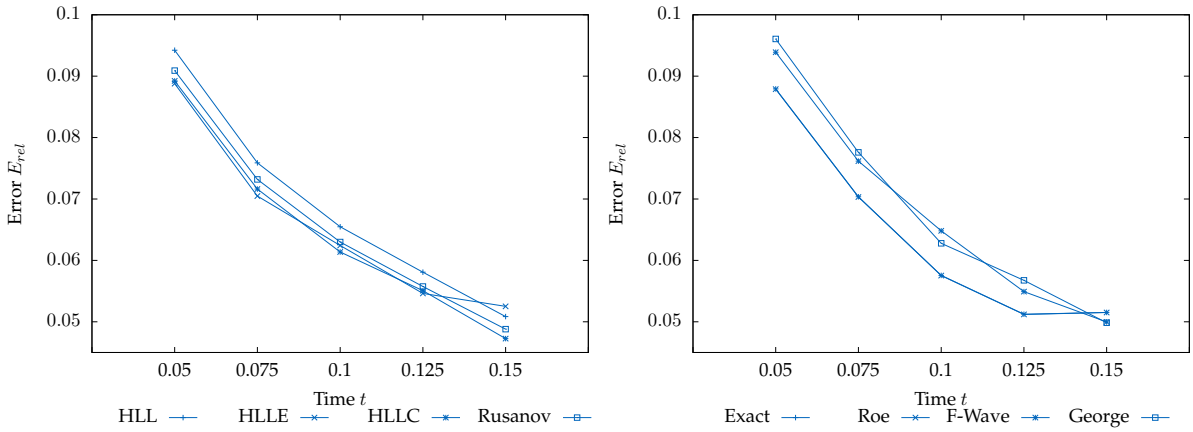
$$
\begin{aligned}
& \partial_t(hu) = +\partial_x(hu^2 + \frac{1}{2}gh^2) = -gh\partial_x(b) \\
\overset{\text{chain rule}}{\Rightarrow} \quad & \partial_t(h)u + h\partial_t(u) + \partial_x(hu^2) + \partial_x(\frac{1}{2}gh^2) = -gh\partial_x(b) \\
\overset{\text{Insert original eq}}{\Rightarrow} \quad & -\partial_x(hu)u + h\partial_t(u) + \partial_x(hu^2) + \partial_x(\frac{1}{2}gh^2) = -gh\partial_x(b) \\
\overset{\text{chain rule}}{\Rightarrow} \quad & h\partial_t(u) - \partial_x(h)u^2 - hu\partial_x(u) + \partial_x(h)u^2 + 2hu\partial_x(u) + g\partial_x(h)h = -gh\partial_x(b) \\
\overset{\text{cancel terms}}{\Rightarrow} \quad & h\partial_t(u) + hu\partial_x(u) + g\partial_x(h)h = -gh\partial_x(b) \\
\overset{\text{divide by h, rearrange}}{\Rightarrow} \quad & \partial_t(u) + u\partial_x(u) + g\partial_x(h) + g\partial_x(b) = 0 \\
\overset{\text{rewrite}}{\Rightarrow} \quad & \partial_t(u) + \partial_x(\frac{1}{2}u^2) + g\partial_x(h) + g\partial_x(b) = 0 \\
\overset{\text{linearity of derivative}}{\Rightarrow} \quad & \partial_t(u) + \partial_x(\frac{1}{2}u^2 + g(h+b)) = 0
\end{aligned}
\tag{B.1}
$$



(a) $t = 0$ (b) $t = 0.05$ (c) $t = 0.10$

Figure B.2.: Analytical solution to the dam break Riemann problem for the one-dimensional shallow water equations with dry initial states.

C. Part III



(a) HLL family

(b) Other solvers

Figure C.1.: Error of hu for the one-rarefaction one-shock scenario 5.1.2 without $t = 0.025$ to make the differences more visible.

Test scenario for well-balancedness with dry states:

$$b(x,y) = 0.75e^{-250((x-0.5)^2+(y-0.5)^2)} + 0.25e^{-20((x-0.75)^2+(y-0.75)^2)}$$

$$q(x,y,0) = \begin{cases} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0.5 \end{pmatrix}, & \text{if } x \leq 0.1 \ \vee \ y \geq 0.9 \\ \begin{pmatrix} \max(0.5-b,0) \\ 0 \\ 0 \\ b(x,y) \end{pmatrix}, & \text{otherwise} \end{cases} \tag{C.1}$$

The hump of water added in the subsequent problem is described by:

$$w = 0.25e^{-1000((x-0.75)^2+(y-0.75)^2)}. \tag{C.2}$$

h is then expressed by $\max(0.5 + w - b, 0)$.

Test scenario for inundation:

$$\begin{aligned} \tilde{b}(x,y) &:= -\frac{0.6}{1+e^{-20(x-0.3)}} + 0.6, \\ \tilde{w}(x,y) &:= \frac{1}{4} e^{-50((x-0.7)^2+(y-0.5)^2)} + 0.5, \\ q(x,y,0) &= \begin{pmatrix} \min(\tilde{w}(x,y) - \tilde{b}(x,y), 0) \\ 0 \\ 0 \\ \tilde{b}(x,y) \end{pmatrix}. \end{aligned} \tag{C.3}$$

Results for the Oscillating Lake scenario at $t = 3$ along $[0.3, 2]$:

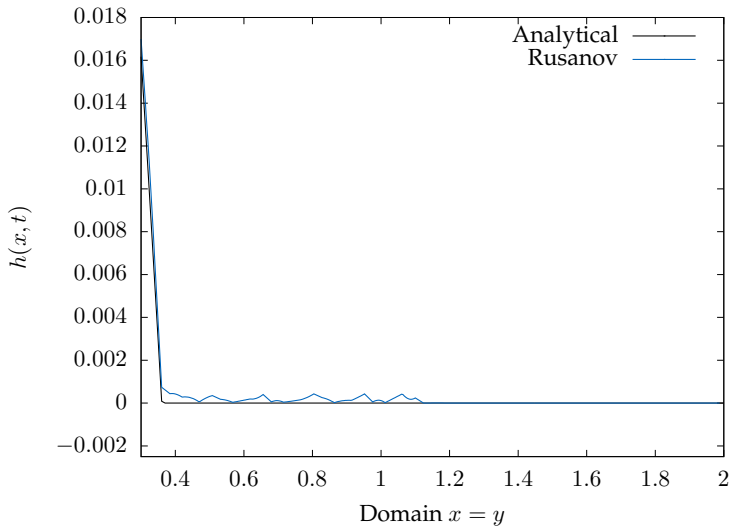


Figure C.2.: Zoomed result for unknown h in oscillating lake scenario at time $t = 3$ along $[0.3, 2]$.

Bibliography

[1] Emmanuel Audusse, François Bouchut, Marie Odile Bristeau, Rupert Klein, and Benoît Perthame. A fast and stable well-balanced scheme with hydrostatic reconstruction for shallow water flows. *SIAM Journal on Scientific Computing*, 25(6):2050–2065, 2004.

[2] Derek S. Bale, Randall J. Leveque, Sorin Mitran, and James A. Rossmanith. A wave propagation method for conservation laws and balance laws with spatially varying flux functions. *SIAM Journal on Scientific Computing*, 24(3):955–978, 2003.

[3] Onno Bokhove. Flooding and drying in discontinuous galerkin finite-element discretizations of shallow-water equations. part 1: One dimension. *Journal of Scientific Computing*, 22:47–82, 1 2005.

[4] Alexander Breuer and Michael Bader. Teaching parallel programming models on a shallow-water code. In *2012 11th International Symposium on Parallel and Distributed Computing*, pages 301–308, 2012.

[5] Bernardo Cockburn, Bo Dong, Johnny Guzmán, and Jianliang Qian. Optimal convergence of the original dg method on special meshes for variable transport velocity. *SIAM Journal on Numerical Analysis*, 48(1):133–146, 2010.

[6] Bernardo Cockburn, George E Karniadakis, and Chi-Wang Shu. The development of discontinuous galerkin methods. In *Discontinuous Galerkin Methods*, pages 3–50. Springer Berlin Heidelberg, 2000.

[7] Rosa Donat and Antonio Marquina. Capturing shock reflections: An improved flux formula. *Journal of Computational Physics*, 125(1):42–58, 1996.

[8] Michael Dumbser and S Balsara. A new efficient formulation of the hllem riemann solver for general conservative and non-conservative hyperbolic systems. *Journal of Computational Physics*, 304(1):275–319, 2016.

[9] Michael Dumbser, Cedric Enaux, and Eleuterio F. Toro. Finite volume schemes of very high order of accuracy for stiff hyperbolic balance laws. *Journal of Computational Physics*, 227(8):3971–4001, 4 2008.

[10] Michael Dumbser, Francesco Fambri, Maurizio Tavelli, Michael Bader, and Tobias Weinzierl. Efficient implementation of ader discontinuous galerkin schemes for a scalable hyperbolic pde engine. *Axioms*, 7(3), 8 2018.

[11] Michael Dumbser and Martin Käser. An arbitrary high-order discontinuous galerkin method for elastic waves on unstructured meshes - ii. the three-dimensional isotropic case. *Geophysical Journal International*, 167(1):319–336, 10 2006.

[12] Michael Dumbser and Eleuterio F. Toro. A simple extension of the osher riemann solver to non-conservative hyperbolic systems. *Journal of Scientific Computing*, 48(1):70–88, 7 2011.

[13] Bernd Einfeldt. On godunov-type methods for gas dynamics. *SIAM Journal on Numerical Analysis*, 25(2):294–318, 4 1988.

[14] Bernd Einfeldt, Claus-Dieter Munz, Philip Roe, and Björn Sjögreen. On godunov-type methods near low densities. *Journal of Computational Physics*, 92(2):213–295, 1991.

[15] Björn Engquist and Stanley Osher. One-sided difference approximations for nonlinear conservation laws. *Mathematics of Computation*, 36(154):321–351, 1981.

[16] David L. George. Numerical approximation of the nonlinear shallow water equations with topography and dry beds: A godunov-type scheme. Master's thesis, 2004.

[17] David L. George. Augmented riemann solvers for the shallow water equations over variable topography with steady states and inundation. *Journal of Computational Physics*, 227(6):3089–3113, 3 2008.

[18] S. K. Godunov. A difference scheme for numerical solution of discontinuous solution of hydrodynamic equations. *Mat. Sb. (N.S.)*, 47:271–306, 1958.

[19] David Gottlieb and Chi-Wang Shu. On the gibbs phenomenon and its resolution. *Society for Industrial and Applied Mathematic*, 39(4):644–668, 1997.

[20] Amiram Harten, Peter D. Lax, and Bram van Leer. On upstream differencing and godunov-type schemes for hyperbolic conservation laws. In *Upwind and High-Resolution Schemes*, pages 53–79. Springer Berlin Heidelberg, 1997.

[21] Jan S. Hesthaven and Tim Warburton. *Nodal discontinuous Galerkin methods: algorithms, analysis, and applications*. Springer Science and Business Media, 2007.

[22] Thomas Y Hou and Philippe G Le Floch. Why nonconservative schemes converge to wrong solutions: Error analysis. *Mathematics of Computation*, 62(206):497–530, 1994.

[23] Yuxin Huang, Ningchuan Zhang, and Yuguo Pei. Well-balanced finite volume scheme for shallow water flooding and drying over arbitrary topography. *Engineering Applications of Computational Fluid Mechanics*, 7(1):40–54, 2013.

[24] Patrick Jenny and Bernhard Müller. Rankine-hugoniot-riemann solver considering source terms and multidimensional effects. *Journal of Computational Physics*, 145(2):575–610, 1998.

[25] David I. Ketcheson, Randall J. LeVeque, and Mauricio J. Del Razo. *Riemann problems and Jupyter solutions*, volume 16. SIAM, 2020.

[26] Martin Kronbichler. Efficient high-order discretizations for computational fluid dynamics. In Martin Kronbichler and Per-Olof Persson, editors, *The Discontinuous Galerkin Method: Derivation and Properties*, pages 1–55. Springer International Publishing, 2021.

[27] Stig Larsson and Vidar Thomée. *Partial Differential Equations with Numerical Methods*, volume 45. Springer, 2003.

[28] Peter Lax and Burton Wendroff. Systems of conservation laws. Technical report, Los Alamos National Lab (LANL), 1958.

[29] Philippe G. Lefloch and Mai Duc Thanh. The riemann problem for the shallow water equations with discontinuous topography. *Communications in Mathematical Sciences*, 5(4):865–885, 2007.

[30] Randall J. LeVeque. *Finite Volume Methods for Hyperbolic Problems*, volume 31. Cambridge University Press.

[31] Randall J. Leveque. Balancing source terms and flux gradients in high-resolution godunov methods: The quasi-steady wave-propagation algorithm. *Journal of computational physics*, 146(1):346–365, 1998.

[32] Randall J. Leveque, David L. George, and Marsha J. Berger. Tsunami modelling with adaptively refined finite volume methods. *Acta Numerica*, 20:211–289, 4 2011.

[33] Randall J. LeVeque and Marcia Pelanti. A class of approximate riemann solvers and their relation to relaxation schemes. *Journal of Computational Physics*, 172(2):572–591, 2001.

[34] Takahiro Miyoshi and Kanya Kusano. A multi-state hll approximate riemann solver for ideal magnetohydrodynamics. *Journal of Computational Physics*, 208(1):315–344, 9 2005.

[35] Sigrun Ortleb, Jonathan Lambrechts, and Tuomas Kärnä. Wetting and drying procedures for shallow water simulations. In Henk Schuttelaars, Arnold Heemink, and Eric Deleersnijder, editors, *The Mathematics of Marine Modelling: Water, Solute and Particle Dynamics in Estuaries and Shallow Seas*, pages 287–314. Springer International Publishing, 2022.

[36] Feng Qu, Jiaojiao Chen, Di Sun, Junqiang Bai, and Chao Yan. A new all-speed flux scheme for the euler equations. *Computers and Mathematics with Applications*, 77(4):1216–1231, 2 2019.

[37] Feng Qu, Di Sun, Qingsong Liu, and Junqiang Bai. A review of riemann solvers for hypersonic flows. *Archives of Computational Methods in Engineering*, 29(3):1771–1800, 5 2022.

[38] Leonhard Andreas Rannabauer. *Earthquake and Tsunami Simulation with high-order ADER-DG methods*. PhD thesis, 2022.

[39] Anne Reinarz, Dominic E Charrier, Michael Bader, Luke Bovard, Michael Dumbser, Kenneth Duru, Francesco Fambri, Alice-Agnes Gabriel, Jean-Matthieu Gallard, Sven Köppel, Lukas Krenz, Leonhard Rannabauer, Luciano Rezzolla, Philipp Samfass, Maurizio Tavelli, and Tobias Weinzierl. Exahype: An engine for parallel dynamically adaptive simulations of wave problems. *Computer Physics Communications*, 254:107251, 2020.

[40] Philip L. Roe. Approximate riemann solvers, parameter vectors, and difference schemes. *Journal of Computational Physics*, 43(2):357–372, 1981.

[41] William Carlisle Thacker. Some exact solutions to the nonlinear shallow-water wave equations. *Journal of Fluid Mechanics*, 107:499–508, 1981.

[42] Eleuterio F. Toro. Riemann problems and the waf method for solving the two-dimensional shallow water equations. *Royal Society of London Philosophical Transactions Series A*, 338, 01 1992.

[43] Eleuterio F. Toro. *Riemann solvers and numerical methods for fluid dynamics: a practical introduction*. Springer Berlin Heidelberg, 2013.

[44] Eleuterio F. Toro. The hllc riemann solver. *Shock Waves*, 29(8):1065–1082, 11 2019.

[45] Eleuterio F. Toro, Cristòbal E. Castro, Davide Vanzo, and Annunziato Siviglia. A flux-vector splitting scheme for the shallow water equations extended to high-order on unstructured meshes. *International Journal for Numerical Methods in Fluids*, 94:1679–1705, 10 2022.

[46] Eleuterio F. Toro, R. C. Millington, and L. A. M. Nejad. Towards verys high order godunov schemes. In *Godunov methods: theory and applications*, pages 907–940, 2001.