Master of Science Thesis

Numerical study of adaptive mesh refinement applied to a third order minimum truncation error Active Flux method

Jeroen Kunnen

July 13, 2018

Aerodynamics

Faculty of Aerospace Engineering

TU Delft

Delft University of Technology

Numerical study of adaptive mesh refinement applied to a third order minimum truncation error Active Flux method

Master of Science Thesis

For obtaining the degree of Master of Science in Aerospace Engineering
at Delft University of Technology

Jeroen Kunnen

July 13, 2018

Faculty of Aerospace Engineering · Delft University of Technology

Delft University of Technology

DELFT UNIVERSITY OF TECHNOLOGY
DEPARTMENT OF AERODYNAMICS

The undersigned hereby certify that they have read and recommend to the Faculty of Aerospace Engineering for acceptance the thesis entitled **"Numerical study of adaptive mesh refinement applied to a third order minimum truncation error Active Flux method"** by **Jeroen Kunnen** in fulfillment of the requirements for the degree of **Master of Science**.

Dated: July 13, 2018

Supervisors:

Dr.ir. M.I. Gerritsma

Prof.dr. S. Hickel

Dr.ir. M. Möller

Summary

Fluid flows are described mathematically by the Navier-Stokes equations. This system of equations is highly nonlinear, lacking an explicit analytic solution. For many fluid flow problems the range of length scales is rather broad, which causes obtaining a satisfying solution numerically to be an expensive and resource-intensive operation. Varying per situation, the Navier-Stokes equations can often be simplified by neglecting or linearizing certain terms. Even then, however, numerical approximation is in many cases found to be inevitable. Combined with the exponential growth of computing power, this explains the severe increase in research on numerical methods over the past few decades.

One challenging problem in this research area is that when flow velocity surpasses acoustic speeds, shock waves may form. Shock waves are minuscule layers which separate two fluid states, resulting in extreme gradients throughout the layer. Accurately representing such disruptive features on a discrete domain requires inordinately small mesh spacing. A too coarse mesh has been shown numerously to result in either a high dissipation rate or spurious oscillations, depending on the method used. Furthermore, fluid flow problems are frequently dominated by advection. It is therefore key that a numerical method succeeds in a correct translation of some initial state on the discrete domain. A model equation that is frequently used for validation of the shape-preserving capabilities of a numerical method is the linear advection equation, which describes pure advection of a specified initial state. Advection of a discontinuous initial state then provides an indication of the extent to which a method succeeds at preserving the shape of a shock wave.

Fairly recently, a new formulation was introduced by Timothy Eymann and Philip Roe under the name of Active Flux Schemes. This formulation extends a regular finite volume method by placing an additional variable on the interfaces between computational cells. The advantage of this is that the numerical stencil becomes denser such that higher orders can be reached without extending outside the physical domain of dependence. Also, the update method for the interface values need not be conservative as this is already satisfied through the underlying finite volume scheme.

In this thesis a third order Active Flux scheme is derived under the constraint of minimizing the truncation error of a Taylor series expansion. Furthermore, it is recognized that in order to accurately represent a continuous shape on a discrete mesh, mesh spacing should locally be coherent with the shape's spatial frequency. To accomplish this without the drastic increase in computational effort that results from uniform refinement, the method is extended to incorporate adaptive mesh refinement. The aim of adaptive mesh refinement is to algorithmically derive the required mesh resolution from local properties in order to close the gap between high accuracy and low computational load.

The adaptive Active Flux method is implemented for the linear advection equation and verified using various initial waveforms with different characteristics. It was found that, in line with expectation, adaptive mesh refinement is not beneficial for smooth waveforms, increasing both simulation time and error level compared to the simulation on an analogous uniform mesh. The results of triangle and sawtooth wave patterns reveal the benefits of adaptive mesh refinement in terms of a reduction in the number of cells while approaching the error development of a simulation on a full resolution uniform mesh. Finally, a high resolution simulation of Zalesak's waveform has been performed. This waveform consists of four individual waveforms, each of different characteristic shape, allowing verification of the adaption algorithm for all features at once. It is shown that the adaptive simulation reaches the same error level as a uniform simulation, while using ~9.6 times less computational cells.

For the non-smooth waveforms that were tested, the errors of the adaptive simulations were smaller than their uniform analogue, given that the refinement/coarsening criteria were strict enough. On the other hand, execution duration revealed that adaption in its current state is quite expensive computationally. However, it should be noted that the code has in no way been optimized, causing it to be impossible to make any definite statements concerning performance. Also, only one-dimensional problems have been tested in this thesis, while the benefits of adaptive mesh refinement become increasingly apparent for higher dimensional problems. For this reason, it is believed that adaptive mesh refinement is a necessity in the simulation of complex unsteady fluid flows.

Acknowledgements

First of all, I would like to take this opportunity to thank my direct supervisor, Marc Gerritsma, for his overall guidance throughout this thesis project. Computational Fluid Dynamics really has become a broad research area in which it is frighteningly easy to lose sense of direction and spend an embarrassingly amount of time wandering through this expanding thick forest of complex concepts and theories. Marc's valuable feedback has kept me from straying too far from the path that defines a successful graduation project, while leaving me the freedom to change heading every now and then. Although his mathematical background has resulted in me leaving his office dumbfounded after a discussion more than once, this forced me to think about concepts on a fundamental level to see if they made sense, not only mathematically but also physically.

Finally, I am grateful for the unconditional emotional support and understanding of my parents, which has provided me the momentum to keep going, especially during the periods when motivation was lacking.

Contents

1 Introduction

Since the early 1980's, research on supersonic flows started to flourish, raising the incentive for the CFD community to develop numerical solvers for such high-speed flows. Supersonic flows differ from subsonic flows in characteristics due to the fact that supersonic flows allow the formation of shock waves; extremely thin layers in which flow properties vary rapidly, approaching a theoretical discontinuity. Traditional numerical solution methods tended to show either oscillatory or excessively dissipative behavior around such features, resulting in a growing demand for new methods that would improve performance in terms of accuracy in the presence of discontinuities.

Consequently, a lot of effort continues to be invested in this specific subject, thereby contributing to the ever expanding collection of numerical schemes. Investigation of these schemes reveals that there is a general consensus that local variation of the numerical dissipation rate provides a way of suppressing oscillatory behavior near discontinuities while allowing the use of higher order schemes in smooth regions. A relatively new type of scheme has been introduced in 2011 by Eymann and Roe [24]; the Active Flux (AF) schemes. In this type of scheme, an extra degree of freedom is imposed on the cell interfaces of a finite volume grid. This approach enables the use of non-conservative update methods for these interface values, as conservation is automatically adhered to through the conservative update method of the cell integral values. The paper by Eymann and Roe describes an update method that relies on characteristic tracing, which is physically intuitive but not trivially extendable to the Euler equations as in that case the state variables cannot be traced along a single path. This thesis will investigate an update method that is ultimately based on minimization of the truncation error of a Taylor series expansion. Furthermore, a way of local modification of mesh resolution to reduce the error near discontinuities is implemented through means of an adaptive mesh refinement algorithm. The question in which these elements are centralized and this thesis aims to answer is stated as:

How does an Active Flux node update method, derived under the constraint of a minimum Taylor series expansion truncation error, compare to the method of characteristic tracing and how can adaptive mesh refinement be implemented in order to cohere to local mesh resolution requirements?

The report is structured as follows. First, Chapter 2 contains a review on the foundation of conservation laws, along with three examples of conservation laws commonly found in the field of aerodynamics; the linear advection equation, Burgers' equation and the Euler equations. Then, a thorough literature study on methods of local control over the dissipation rate is provided in Chapter 3. The Active Flux method implementation as proposed by Eymann and Roe is explored in more detail in Chapter 4. This chapter also introduces a two-step method based on minimum Taylor series expansion truncation error, which is then rewritten to a single step Active Flux form. Thereafter, Chapter 5 focuses on the implementation of mesh adaption for Active Flux schemes, including a detailed overview of the code structure. Simulations have been run for various waveforms and several parameter configurations, for which a summary of relevant findings can be found in Chapter 6. From these results several conclusions are drawn, which are presented in Chapter 7. Finally, Chapter 8 provides some recommendations and points of attention for possible future research.

2

Background: Conservation laws

Conservation is one of the fundamental laws of nature and often serves as a basis for the understanding and simulation of all kinds of phenomena. It recurs in many scientific fields in various forms. This broad scope has resulted in the concept being discussed on a large scale, from different perspectives. The mathematical foundation is well documented, see e.g. [14, 15, 27]. Conservation is necessary to retain the physical validity of a simulation and is for this reason the underlying principle of many numerical methods. Therefore, this chapter provides a brief recapitulation of the basics, starting with the derivation in Section 2.1. Section 2.2 introduces the concept of characteristics. Finally, three well-known conservation laws are treated concisely; the linear advection equation (Section 2.3), Burgers' equation (Section 2.4) and the one-dimensional Euler equations (Section 2.5).

2.1. Derivation

Considering a space Ω enclosed by a boundary $\partial\Omega$, conservation requires that, in the absence of sources and sinks, the total amount of a quantity u in Ω can only change by a non-zero average flux of u through $\partial\Omega$. This requirement translates to the following mathematical expression:

$$\int_\Omega u(\xi, t_1)\,\mathrm{d}\mathcal{V} = \int_\Omega u(\xi, t_0)\,\mathrm{d}\mathcal{V} - \int_{t_0}^{t_1} \oint_{\partial\Omega} \boldsymbol{f}(u)\cdot\boldsymbol{n}\,\mathrm{d}\mathcal{S}\mathrm{d}t \qquad \text{(integral form)} \tag{2.1a}$$

$$\int_\Omega u(\xi, t_1) - u(\xi, t_0)\,\mathrm{d}\mathcal{V} = -\int_{t_0}^{t_1} \oint_{\partial\Omega} \boldsymbol{f}(u)\cdot\boldsymbol{n}\,\mathrm{d}\mathcal{S}\mathrm{d}t \tag{2.1b}$$

$$\int_{t_0}^{t_1}\int_\Omega u_t\,\mathrm{d}\mathcal{V}\mathrm{d}t = -\int_{t_0}^{t_1}\int_\Omega \boldsymbol{\nabla}\cdot\boldsymbol{f}(u)\,\mathrm{d}\mathcal{V}\mathrm{d}t \tag{2.1c}$$

$$\int_{t_0}^{t_1}\int_\Omega u_t + \boldsymbol{\nabla}\cdot\boldsymbol{f}(u)\,\mathrm{d}\mathcal{V}\mathrm{d}t = 0 \tag{2.1d}$$

$$u_t + \boldsymbol{\nabla}\cdot\boldsymbol{f}(u) = 0 \qquad \text{(differential form)} \tag{2.1e}$$

In these equations, $\boldsymbol{f}$ is a vector field representing the flux of quantity u. In (2.1a) and (2.1b), ξ is a convenient shorthand notation for the collection of spatial parameters, the amount of which is equal to the dimension of Ω. It should also be observed that the use of the divergence theorem in (2.1c) is only valid when the vector field $\boldsymbol{\nabla}\cdot\boldsymbol{f}$ is integrable on Ω. When multiple quantities are to be conserved, (2.1e) can be written as follows, where the gradient vector acts on the vector field column-wise:

$$\boldsymbol{u}_t + \boldsymbol{\nabla}\cdot\boldsymbol{f}(\boldsymbol{u}) = \boldsymbol{0}. \tag{2.2}$$

2.2. Characteristics

In one spatial dimension, (2.2) reduces to:

$$\boldsymbol{u}_t + \boldsymbol{f}(\boldsymbol{u})_x = \boldsymbol{0}. \tag{2.3}$$

When both the flux function $\boldsymbol{f}(\boldsymbol{u})$ and the vector of conserved variables $\boldsymbol{u}$ are differentiable, the Jacobian $\boldsymbol{J} = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}}$ allows this equation to be written as:

$$\boldsymbol{u}_t + \boldsymbol{J}\boldsymbol{u}_x = \boldsymbol{0}. \tag{2.4}$$

In case the matrix $\boldsymbol{J}$ contains off-diagonal non-zero entries, interaction exists between the entries of $\boldsymbol{u}$, complicating the behaviour of the system. In order to simplify the analysis of such a system, an attempt to decouple the system can be made by applying a change of variables. Letting this new set of variables be contained in the vector $\boldsymbol{v}$, the desired form becomes:

$$\boldsymbol{v}_t + \boldsymbol{\Lambda}\boldsymbol{v}_x = \boldsymbol{0}, \tag{2.5}$$

where $\boldsymbol{\Lambda}$ is a diagonal matrix. Introducing a second Jacobian $\boldsymbol{L} = \frac{\partial \boldsymbol{v}}{\partial \boldsymbol{u}}$, this can be rewritten to:

$$\frac{\partial \boldsymbol{v}}{\partial \boldsymbol{u}}\boldsymbol{u}_t + \boldsymbol{\Lambda}\frac{\partial \boldsymbol{v}}{\partial \boldsymbol{u}}\boldsymbol{u}_x = \boldsymbol{0}, \tag{2.6}$$

$$\boldsymbol{L}\boldsymbol{u}_t + \boldsymbol{\Lambda}\boldsymbol{L}\boldsymbol{u}_x = \boldsymbol{0}. \tag{2.7}$$

This shows that a decoupled system results for the following change of variables:

$$\mathrm{d}\boldsymbol{v} = \boldsymbol{L}\mathrm{d}\boldsymbol{u}. \tag{2.8}$$

For linear systems, this simply becomes the map $\boldsymbol{v} = \boldsymbol{L}\boldsymbol{u}$. In this case, for each variable v_i in $\boldsymbol{v}$, a path in $x-t$ space exists along which v_i is constant. This path is defined by $\frac{\mathrm{d}x}{\mathrm{d}t} = \lambda_i$, where λ_i is the i^{th} entry of the diagonal of $\boldsymbol{\Lambda}$. The v_i are commonly referred to as characteristic variables or Riemann invariants. The system of equations in (2.5) is called the system of characteristic equations. The paths along which the characteristic variables are constant, have been termed the characteristics. Premultiplying (2.7) by the inverse of $\boldsymbol{L}$ yields:

$$\boldsymbol{u}_t + \boldsymbol{L}^{-1}\boldsymbol{\Lambda}\boldsymbol{L}\boldsymbol{u}_x = \boldsymbol{0}. \tag{2.9}$$

Comparing this to (2.4) reveals that $\boldsymbol{L}^{-1}\boldsymbol{\Lambda}\boldsymbol{L}$ is in fact the diagonalization of the Jacobian $\boldsymbol{J}$, where the matrix $\boldsymbol{L}$ thus consists of the left eigenvectors of $\boldsymbol{J}$, and $\boldsymbol{\Lambda}$ contains $\boldsymbol{J}$'s eigenvalues. Unfortunately, for the Euler equations, (2.8) cannot be directly integrated to explicitly yield $\boldsymbol{v}$ in terms of $\boldsymbol{u}$. Therefore, only a differential form of the characteristic variables is available.

For each point (x,t), the characteristics intersecting that point can be drawn. From this, two regions can be inferred; the domain of dependence (DOD) and the domain of influence (DOI). The domain of dependence consists of the collection of points able to influence the state at (x,t). Similarly, the domain of influence comprises all points whose states can be influenced by the point (x,t). For a single conserved variable (or a system with equal characteristic velocities), both domains converge to a single line on top of the characteristics. For a system with distinct characteristic velocities, the domains are bounded by the fastest and slowest moving characteristics, see Figure 2.1. Thus, for schemes to correctly mimic physical phenomena, it should be kept in mind that information travels at a finite velocity, and the state at some point in space-time can only be affected by points in its DOD. This poses some constraints on the numerical solution method. In Figure 2.1 for example, a non-directionally-biased scheme would be an inappropriate choice as the characteristic velocity extrema are of the same sign.

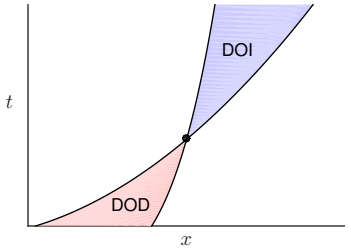


Figure 2.1: The domain of influence (DOI) and domain of dependence (DOD) for a point (x,t) bounded by the minimum and maximum characteristic velocity.

2.3. Linear advection equation

For the case of just one variable, (2.4) becomes:

$$u_t + f(u)_x = u_t + \lambda(u)u_x = 0. \tag{2.10}$$

The simplest form of this equation occurs for a constant characteristic velocity, $\lambda(u) = a$. This form is well-known as the linear advection equation. Since the characteristic velocity is invariant throughout the whole domain, the characteristics are all parallel and straight, their slopes dependent on the value of a. The following figure shows some characteristics for the possible values of a.

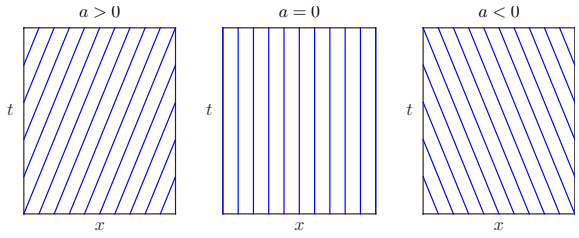


Figure 2.2: Characteristics of the linear advection equation for various values of the wave speed, a.

The characteristic pattern indicates that the linear advection equation performs nothing more than a shift of the initial conditions with velocity a, preserving the shape. A constant value is tied to each characteristic, which implies that as long as a value is specified somewhere in the $x-t$ domain for each characteristic, the solution in the whole domain is known. Therefore, the complete solution becomes:

$$u(x,t) = u(x - a(t - t_s), t_s), \tag{2.11}$$

where t_s is the time at which the solution is known. Because the linear advection equation simply transports a shape without deformation, it serves as a good measure of a scheme's shape preserving capability.

2.4. Burgers' equation

Another interesting equation is found by defining the characteristic velocity at a certain location as the solution value at that point, or $\lambda(u) = u$. This equation known as Burgers' equation and can be written in advection/characteristic form and non-conservative form:

$$u_t + uu_x = 0 \quad \text{(advection/characteristic form)}, \tag{2.12}$$

$$u_t + \left(\frac{u^2}{2}\right)_x = 0 \quad \text{(conservative form)}. \tag{2.13}$$

Mathematically, these equations differ only by the fact that the advection form is more constraining; cases (e.g. $u = |x|$) exist for which $\left(\frac{u^2}{2}\right)_x$ is continuous but u_x is not. Comparing (2.13) to (2.3), the conserved variable is u itself, and the flux expression equals $\frac{u^2}{2}$.

Because the characteristic velocity is equal to u, and the quantity that is constant along these characteristics is also u, the characteristics are represented by straight lines in (x,t)-space. However, as the characteristic velocity is no longer constant, characteristics may converge and diverge. The flux function is convex, implying that the characteristic velocity increases as u increases. Therefore, regions for which $u_x < 0$ and thus $\lambda_x < 0$ correspond to converging characteristics, and regions where $u_x > 0$ correspond to diverging characteristics, see respectively Figures 2.3 and 2.4.

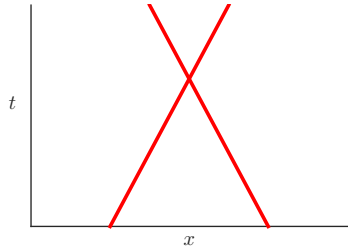


Figure 2.3: Characteristics converging in time as a result of a negative $\frac{d\lambda}{dx}$.

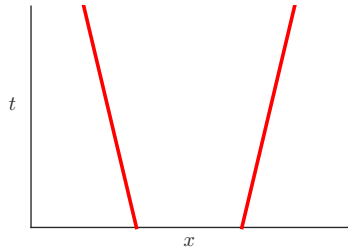


Figure 2.4: Characteristics diverging in time as a result of a positive $\frac{d\lambda}{dx}$.

2.4.1. Converging characteristics

For Burgers' equation, converging characteristics result in shape compression, also known as wave steepening, see Figure 2.5.

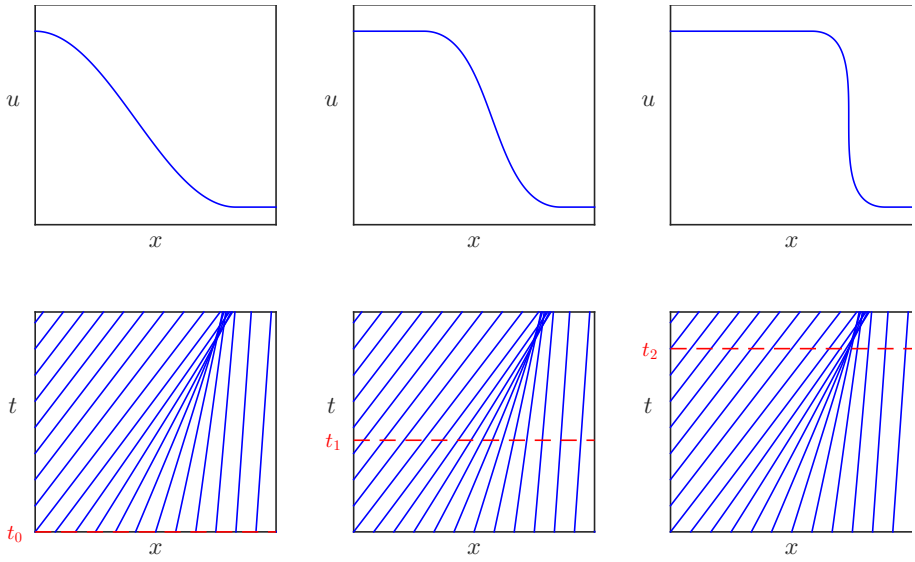


Figure 2.5: A visualization of the steepening process caused by converging characteristics.

As can be seen in this figure, at some point, converging characteristics will intersect. From there on the solution can be traced back to two distinct origins, yielding multivalued points and thus for many cases an unphysical solution. It is also at this point, that $\frac{\partial u}{\partial x}$ reaches negative infinity. Due to the use of the divergence theorem in the derivation of the conservation law, Burgers' equation is only valid in regions where the flux expression is differentiable. So, to find the complete solution in the presence of discontinuities, an alternative formulation is required to relate the states on both sides of such discontinuities. To derive this expression, observe the case of a moving discontinuity with constant states on either side, see Figure 2.6.

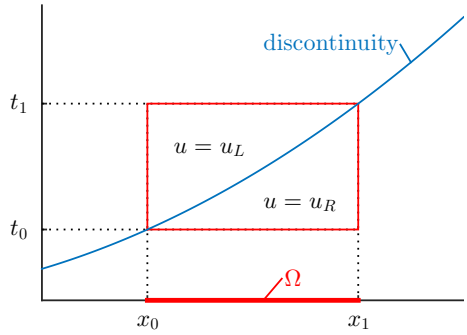


Figure 2.6: The bounding box used to derive the jump condition.

A rectangular bounding box can be formed using two points on the discontinuity path as opposing vertices. Since the integral form of the conservation law is still valid in the presence of discontinuities, it can be applied to the domain created by the projection of the bounding box as follows:

$$\int_\Omega u(\xi, t_1) - u(\xi, t_0)\, \mathrm{d}\mathcal{V} = -\int_{t_0}^{t_1} \oint_{\partial\Omega} \boldsymbol{f}(u) \cdot \boldsymbol{n}\, \mathrm{d}\mathcal{S}\mathrm{d}t, \tag{2.14a}$$

$$\int_{x_0}^{x_1} u(x, t_1) - u(x, t_0)\, \mathrm{d}x = -\int_{t_0}^{t_1} \left[f(u(x,t))\right]_{x_0}^{x_1} \mathrm{d}t, \tag{2.14b}$$

$$u_L\,(x_1 - x_0) - u_R\,(x_1 - x_0) = -\int_{t_0}^{t_1} f\,(u_R) - f\,(u_L)\ \mathrm{d}t, \tag{2.14c}$$

$$(u_L - u_R)\,(x_1 - x_0) = -\big(f\,(u_R) - f\,(u_L)\big)\,(t_1 - t_0)\,, \tag{2.14d}$$

$$\frac{x_1 - x_0}{t_1 - t_0} = \frac{f\,(u_L) - f\,(u_R)}{u_L - u_R}, \tag{2.14e}$$

$$\lim_{(t_1 - t_0)\to 0^+} \frac{x_1 - x_0}{t_1 - t_0} = \frac{\mathrm{d}x}{\mathrm{d}t} = v_d = \frac{f\,(u_L) - f\,(u_R)}{u_L - u_R}. \tag{2.14f}$$

This final sought-after expression is known as the jump condition as it relates the states on both sides of a discontinuity through the velocity of the discontinuity. Substitution of the flux function for Burgers' equation into (2.14f) leads to the following expression for the discontinuity speed:

$$\begin{aligned} v_d &= \frac{f\,(u_L) - f\,(u_R)}{u_L - u_R} \\ &= \frac{u_L^2 - u_R^2}{2(u_L - u_R)} \\ &= \frac{(u_L + u_R)(u_L - u_R)}{2(u_L - u_R)} \\ &= \frac{u_L + u_R}{2}. \end{aligned} \tag{2.15}$$

So, for Burgers' equation, the velocity of the discontinuity is simply the arithmetic average of the surrounding characteristic speeds. For the case of two distinct constant states, the complete characteristic pattern thus looks as shown in Figure 2.7.

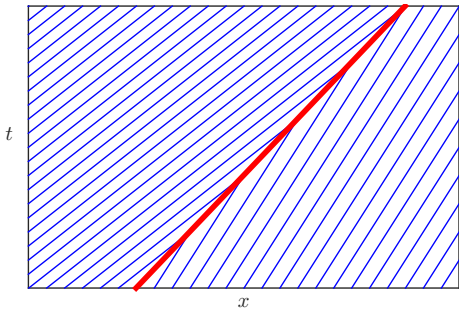


Figure 2.7: The characteristic pattern including the shock path on which the characteristics terminate.

2.4.2. Diverging characteristics

For diverging characteristics, the opposite takes place. The solution is expanded, smoothing the gradients as time progresses. An example of this process is shown in Figure 2.8.

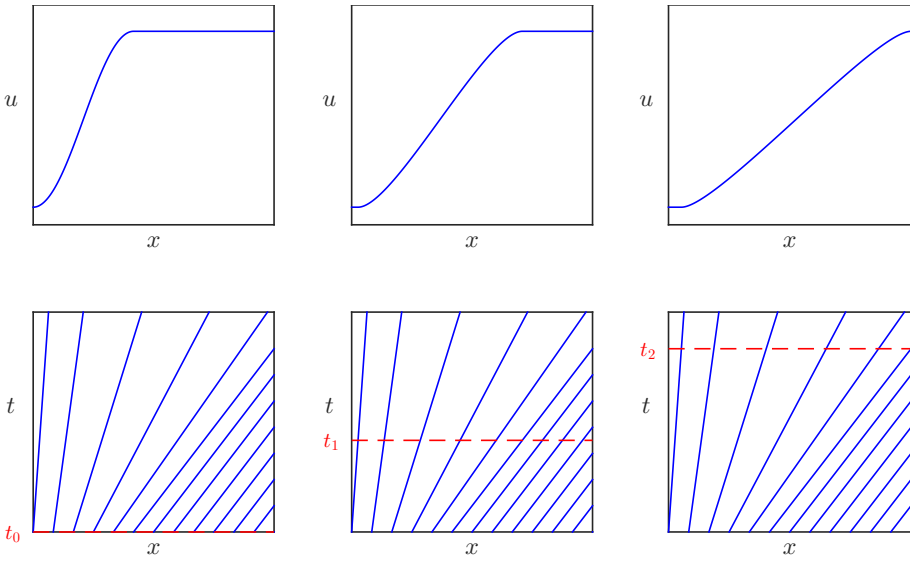
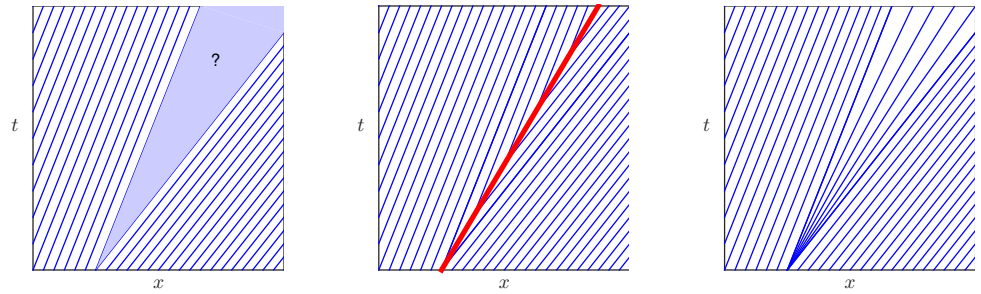


Figure 2.8: A figure showing shape manipulation due to expanding characteristics.

When the initial solution is smooth, for purely diverging characteristics, the solution throughout the domain can easily be found by tracing the characteristics. Analogous to the discussion on converging characteristics, an interesting case to look at is two distinct constant states, separated by a discontinuity. In this case, the characteristic speed has to increase with increasing x, implying that $u_R > u_L$. Observe that (2.14f) is still valid, and is a symmetric expression in the sense that interchanging u_L and u_R does not alter the resulting discontinuity speed. When applying this equation to an expansion, the characteristic pattern shown in Figure 2.9b is found. This solution is often referred to as an expansion shock, and is a valid solution of the inviscid Burgers' equation. For the viscous Burgers' equation with vanishing viscosity, the pattern shown in Figure 2.9c is found instead. Since truly inviscid processes are merely theoretical, we will find the latter solution to be more useful than the expansion shock when modelling real physics. We should however be aware of this extra solution branch when doing numerics, as convergence to the wrong branch can cause significant deviations from the true flow structure. Since Burgers' equation allows the formation of discontinuities and has an exact analytic solution, it is an ideal tool to observe how well a scheme handles nonlinear effects.

2.5. Euler equations

The motion and state of fluids is completely described by the Navier-Stokes equations, which are a mathematical translation of three physical principles: the conservation of mass, momentum and energy. While these equations are powerful, they are highly nonlinear and difficult to solve. Simplifications



(a) (b) (c)

Figure 2.9: Characteristic patterns in the presence of a centered expansion. (b) A possible characteristic pattern of the inviscid Burgers' equation. (c) The characteristic pattern found for the viscous Burgers' equation with vanishing viscosity.

of the Navier-Stokes equations often allow identification of the large-scale flow structure, or modelling of flows with improved efficiency and no significant accuracy loss. One of these simplifications is the omission of viscous and gravitational terms, which results in the Euler equations. The integral formulation of the Euler equations is as follows:

$$\frac{\partial}{\partial t}\int_\Omega \rho \,\mathrm{d}\mathcal{V} + \int_{\partial\Omega} \rho \boldsymbol{v}\cdot\boldsymbol{n}\,\mathrm{d}\mathcal{S} = 0, \tag{2.16a}$$

$$\frac{\partial}{\partial t}\int_\Omega \rho \boldsymbol{v}\,\mathrm{d}\mathcal{V} + \int_{\partial\Omega} \left(\rho \boldsymbol{v}\otimes\boldsymbol{v}\right)\boldsymbol{n}\,\mathrm{d}\mathcal{S} + \int_{\partial\Omega} p\boldsymbol{n}\,\mathrm{d}\mathcal{S} = 0, \tag{2.16b}$$

$$\frac{\partial}{\partial t}\int_\Omega \rho E\,\mathrm{d}\mathcal{V} + \int_{\partial\Omega} \rho E\boldsymbol{v}\cdot\boldsymbol{n}\,\mathrm{d}\mathcal{S} + \int_{\partial\Omega} p\boldsymbol{v}\cdot\boldsymbol{n}\,\mathrm{d}\mathcal{S} = 0. \tag{2.16c}$$

In these equations, ρ is the mass density, $\boldsymbol{v}$ is the velocity vector and p is the pressure. Also, the energy E comprises both the internal and kinetic energy:

$$E = e + \frac{\boldsymbol{v}\cdot\boldsymbol{v}}{2}. \tag{2.17}$$

By applying the integral equations to an infinitesimal volume, the differential formulation of the Euler equations can be found:

$$\frac{\partial \rho}{\partial t} + \nabla\cdot\left(\rho\boldsymbol{v}\right) = 0, \tag{2.18a}$$

$$\frac{\partial \rho\boldsymbol{v}}{\partial t} + \nabla\cdot\left(\rho\boldsymbol{v}\otimes\boldsymbol{v}\right) + \nabla p = \boldsymbol{0}, \tag{2.18b}$$

$$\frac{\partial \rho E}{\partial t} + \nabla\cdot\left(\rho E\boldsymbol{v}\right) + \nabla\cdot\left(p\boldsymbol{v}\right) = 0. \tag{2.18c}$$

In one dimension, these reduce to:

$$\frac{\partial \rho}{\partial t} + \frac{\partial \rho u}{\partial x} = 0, \tag{2.19a}$$

$$\frac{\partial \rho u}{\partial t} + \frac{\partial \rho u^2 + p}{\partial x} = 0, \tag{2.19b}$$

$$\frac{\partial \rho E}{\partial t} + \frac{\partial \rho E u + p u}{\partial x} = 0. \tag{2.19c}$$

This system of equations can be rewritten to vector format:

$$\boldsymbol{q}_t + \boldsymbol{f}(\boldsymbol{q})_x = \boldsymbol{0}, \tag{2.20}$$

where

$$\boldsymbol{q} = \begin{bmatrix} \rho \\ \rho u \\ \rho E \end{bmatrix} \quad \boldsymbol{f}(\boldsymbol{q}) = \begin{bmatrix} \rho u \\ \rho u^2 + p \\ \rho E u + p u \end{bmatrix}. \tag{2.21}$$

The three equations (2.19a-2.19c) contain four independent variables; density, velocity, energy and pressure. This makes it an underdetermined system. To close the system, an expression of state is required. Limiting to calorically perfect gasses, the ideal gas law can be used:

$$\begin{aligned} p &= \rho(\gamma-1)e \\ &= \rho(\gamma-1)\left(E - \frac{u^2}{2}\right). \end{aligned} \tag{2.22}$$

2.5.1. Invariants

Since (2.20) is in the same form as (2.3), the same approach can be used to determine the characteristic patterns. To do this, the first step is to find the Jacobian $J = \frac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{q}}$. Reformulate $\boldsymbol{q}$ and $\boldsymbol{f}(\boldsymbol{q})$ as:

$$\boldsymbol{q} = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \end{bmatrix}, \quad \boldsymbol{f}(\boldsymbol{q}) = \begin{bmatrix} q_2 \\ (\gamma-1)q_3 + (3-\gamma)\frac{q_2^2}{2q_1} \\ \gamma\frac{q_2 q_3}{q_1} + \frac{1-\gamma}{2}\frac{q_2^3}{q_1^2} \end{bmatrix}. \tag{2.23}$$

The Jacobian then equals:

$$\frac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{q}} = \begin{bmatrix} 0 & 1 & 0 \\ \frac{\gamma-3}{2}\left(\frac{q_2}{q_1}\right)^2 & (3-\gamma)\frac{q_2}{q_1} & \gamma-1 \\ -\gamma\frac{q_2 q_3}{q_1^2} + (\gamma-1)\left(\frac{q_2}{q_1}\right)^3 & \gamma\frac{q_3}{q_1} + 3\frac{1-\gamma}{2}\left(\frac{q_2}{q_1}\right)^2 & \gamma\frac{q_2}{q_1} \end{bmatrix}. \tag{2.24}$$

The eigenvalues of this matrix are $\lambda_1 = \frac{q_2}{q_1}$, $\lambda_2 = \frac{q_2}{q_1} + \sqrt{\gamma(\gamma-1)\left(\frac{q_3}{q_1} - \frac{1}{2}\left(\frac{q_2}{q_1}\right)^2\right)}$ and $\lambda_3 = \frac{q_2}{q_1} - \sqrt{\gamma(\gamma-1)\left(\frac{q_3}{q_1} - \frac{1}{2}\left(\frac{q_2}{q_1}\right)^2\right)}$.
Expressed in non-conserved variables, the eigenvalues correspond to $\lambda_1 = u$, $\lambda_2 = u + a$ and $\lambda_3 = u - a$, where u is the flow velocity and a the speed of sound. The extrema of these eigenvalues define the boundaries of the domains of dependence and influence.

2.5.2. Jump conditions

Similar to what was done for Burgers' equation, for each equation in (2.20) a jump condition can be found by applying (2.14f).

$$v_d = \frac{\boldsymbol{f}(\boldsymbol{u}_L) - \boldsymbol{f}(\boldsymbol{u}_R)}{\boldsymbol{u}_L - \boldsymbol{u}_R}, \tag{2.25a}$$

$$v_d\,(\boldsymbol{u}_L - \boldsymbol{u}_R) = \boldsymbol{f}(\boldsymbol{u}_L) - \boldsymbol{f}(\boldsymbol{u}_R), \tag{2.25b}$$

$$v_d \begin{bmatrix} \rho_L - \rho_R \\ \rho_L u_L - \rho_R u_R \\ \rho_L E_L - \rho_R E_R \end{bmatrix} = \begin{bmatrix} \rho_L u_L - \rho_R u_R \\ \rho_L u_L^2 + p_L - \rho_R u_R^2 - p_R \\ \rho_L E_L u_L + p_L u_L - \rho_R E_R u_R - p_R u_R \end{bmatrix}. \tag{2.25c}$$

In case of a steady shock, v_d vanishes and we arrive at the Rankine-Hugoniot relations for a steady normal shock:

$$\rho_L u_L = \rho_R u_R, \tag{2.26a}$$

$$\rho_L u_L^2 + p_L = \rho_R u_R^2 + p_R, \tag{2.26b}$$

$$\rho_L E_L u_L + p_L u_L = \rho_R E_R u_R + p_R u_R. \tag{2.26c}$$

3

Literature study

Due to the scientific and commercial importance of accurately solving the Euler equations, a lot of research has been performed on the subject. A consequence of the combination of nonlinear behavior and absence of viscosity in the Euler equations is that it allows solutions to become discontinuous. Because the large majority of numeric schemes approximate the true solution by discretization on a finite grid, such discontinuities pose a challenge as they cannot be represented exactly on these grids. In general, first order methods show monotonic behavior, so these at least result in a consistent representation of a discontinuity, be it smeared over multiple cells. The downside of first order methods is that their inherent high dissipativity is reflected by their poor shape-preserving capabilities. On the other hand, higher order methods are less dissipative, but not monotonicity preserving unless additional measures are taken. This results in (sometimes unstable) oscillatory behavior near discontinuities or other non-smooth regions in the solution, known as Gibb's phenomenon. These oscillations spread throughout the domain, thereby polluting the solution. Over the past few decades, a lot of effort has been put into research aiming to find new numeric methods that would provide decent accuracy for a broad range of flow conditions. During this research, several problems proven difficult to solve have been discovered, which therefore often serve as validation cases for new methods. There is a general consensus that local control of the dissipation rate is key in the success of the used method [16, 22, 26, 32]. Thus, the majority of methods can be categorized by their type of approach in accomplishing this dissipation control. Four main types will be discussed in this chapter: Hybrid methods (Section 3.1), flux limiters (Section 3.2), direct manipulation of the dissipation rate (Section 3.3) and finally mesh adaption methods (Section 3.4).

3.1. Hybrid methods

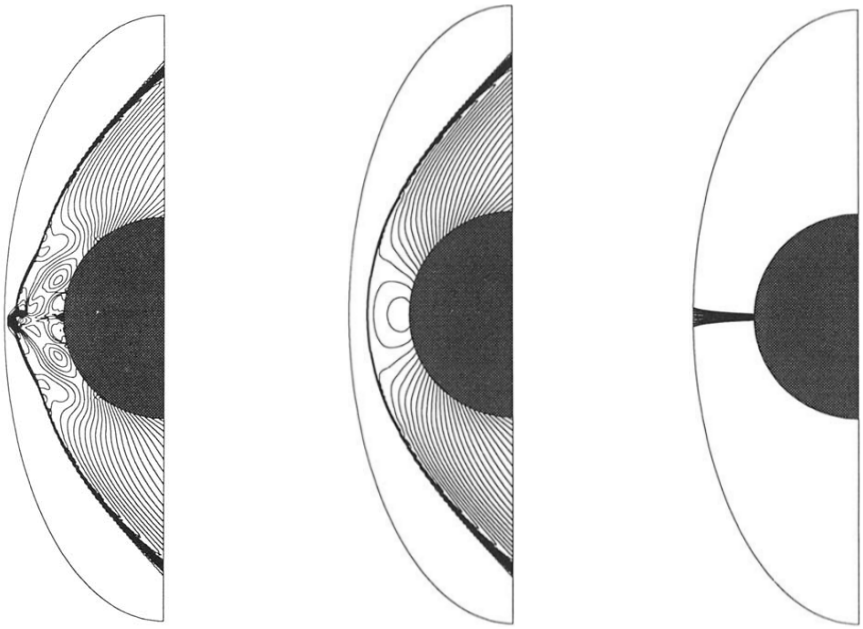
The idea of hybrid methods originates from the observation that monotonicity preserving first order methods are fairly capable at resolving discontinuities, in general spreading them over a few grid cells, where higher order methods perform better in smooth regions. Higher order methods are less dissipative and are thus better at preserving (smooth) shapes, but show oscillatory behavior near discontinuities. Hybrid solvers tackle this problem by incorporating two or more schemes, locally switching between these schemes to only increase dissipation in regions where it is required. Of course, the two important aspects of a hybrid method are the scheme selection and the switching function, which determines the scheme that is to be used. In short, a switching function attempts to value a scheme's performance using the local solution state, after which the scheme that is expected to yield the best result is selected to advance the solution. The switching function can be discontinuous, meaning that for each computational element exactly one scheme is selected. In other cases, a blend of schemes is used to smoothen the transitions between regions of different schemes, for example by using a weighted average.

A disadvantage of hybrid schemes is that the combination of individual schemes makes them hard to analyze and validate. For complex flow interactions, it is difficult to predict in which regions switching functions will be active and how they affect the flow structure. Also, the switching functions are often found empirically and involves tunable parameters. However, as Quirk states in [22], this is pre-

ferred over the addition of artificial dissipation as it is easier to determine *where* dissipation should be increased opposed to *how much* dissipation is required. Several examples of hybrid schemes are concisely described in Sections 3.1.1-3.1.4.

3.1.1. Adaptive Riemann solver

A classic example of a hybrid scheme is J.J. Quirk's adaptive Riemann solver [22]. He primarily used Roe's approximate Riemann solver, switching to the more dissipative HLLE scheme in regions where Roe's scheme is known to experience difficulties. One of the failings that occurs when using Roe's approximate Riemann solver is known as the carbuncle phenomenon. This phenomenon, see Figure 3.1a, occurs when simulating a hypersonic flow over a blunt object. The solution converges to a state where a bulge arises on top of the bow shock near the center line. Although it being a valid solution of the Euler equations, this particular behavior is not reflected by experiments. Figure 3.1b shows how Quirk's hybrid method successfully avoids the carbuncle phenomenon when solving a hypersonic blunt body flow.



(a) (b) (c)

Figure 3.1: A figure showing the impact of locally replacing Roe's approximate Riemann solver by the HLLE scheme. Taken from [22]. (a) Solution using only Roe's scheme. Carbuncle is clearly visible. (b) Solution using hybrid scheme. No carbuncle can be observed. (c) Visual representation of region where the switching function is active.

3.1.2. HLLCM scheme

A more recent example of a hybrid scheme is proposed by Shen [33]. He introduces a modification to the HLLC scheme, naming it HLLCM. He finds that the HLLCM performs well in terms of numerical stability, but is unable to preserve steady shear waves. Since the original HLLC method has no problem resolving these shear waves, he proposes a linear switching function based on the detection of the presence of a shear wave, producing accurate results.

3.1.3. Rotated-hybrid Riemann solver

A promising class of hybrid approaches comprises the so-called rotated-hybrid Riemann solvers, first proposed by H. Nishikawa and K. Kitamura in 2008 [11]. In two dimensional simulations, instead of solving the one-dimensional Euler equations normal to a cell interface, the solver of Nishikawa and Kitamura solves the two-dimensional Euler equations, applying a different solver to directions normal and parallel to a cell interface. A more dissipative solver is used in the normal direction to suppress the development of carbuncle-like phenomena, while accurately resolving the shear layer by applying a less dissipative solver in the parallel direction. The Rotated-RHLL solver combines the Roe and HLLE

flux formulations, yielding very accurate solutions. Similar to the HLLC-HLLCM method, this solver is able to capture shear waves, which causes it to be able to accurately resolve boundary layers as well. An advantage of rotated-hybrid solvers over regular hybrid methods is that it avoids the implementation of a switching function and the inevitable empirical tuning of such a function.

3.1.4. (W)ENO schemes

The Essentially Non-Oscillatory (ENO) schemes is a class of schemes introduced by Harten, Engquist, Osher and Chakravarthy [6] in 1987. ENO schemes make use of an adaptive stencil to allow higher order interpolation methods without violating the requirement of diminishing total variation. The idea is to construct a collection of different stencils with the same order of accuracy in smooth regions, of which one is chosen to advance the solution with, based on the local smoothness of the solution. This avoids Gibb's phenomenon near discontinuities without having to resort to first order methods, like many other schemes do. The ENO approach can be seen as a hybrid method, as it involves multiple numerical methods and a switching function. A significant improvement in terms of accuracy and computational efficiency of the ENO schemes has been made by the introduction of the Weighted Essentially Non-Oscillatory (WENO) schemes [31]. Instead of selecting one stencil on which the interpolation is performed, the interpolations of all considered stencils are combined by weighted averaging, where the weights are based on local solution smoothness. This approach results in more gradual transitions and increases the order of accuracy by one.

3.2. Flux limiters

As stated in the introduction of this chapter, linear methods of an order higher than one suffer from spurious oscillatory behavior near discontinuities and sharp changes in the solution. Forcing the solution to be monotone is a way to remove these oscillations. Godunov's theorem states that such monotonicity preserving linear schemes can be of at most order one. Although the monotonicity property is highly desirable as the solution will not display spurious behavior, first order accuracy heavily smears the solution due to the high dissipation rate. There was thus a need to venture into the domain of nonlinear methods in order to find monotonicity preserving schemes of order higher than one. A quantity that indicates whether a scheme is able to introduce spurious wiggles is the solution's total variation (TV), defined by Harten in 1983 [8]. In continuous and discrete form, this quantity is defined as:

$$TV = \int \left| \frac{\partial u}{\partial x} \right| \mathrm{d}x \qquad \text{(Continuous)}, \tag{3.1a}$$

$$= \sum_i |u_{i+1} - u_i| \qquad \text{(Discrete)}. \tag{3.1b}$$

A scheme is said to be total variation diminishing (TVD) if the total variation is monotonically decreasing for increasing time $\left(TV^{n+1} < TV^n\right)$. An important observation made by Harten was that TVD schemes are monotonicity preserving. This means that if higher order linear schemes could be modified in order to become TVD, spurious wiggles could be avoided while simultaneously maintaining a high order of accuracy. This is exactly what flux limiters aim to do. Flux limiters define the flux term in finite volume formulations as a weighted combination of flux terms of a low order monotonicity preserving scheme and a higher order scheme. The total flux then becomes:

$$\boldsymbol{F} = \boldsymbol{F}_L + \phi(r)\left(\boldsymbol{F}_H - \boldsymbol{F}_L\right), \tag{3.2}$$

where the subscripts L and H represent low and high order scheme flux terms, respectively. Also, r is a smoothness indicator, commonly defined as the ratio between two subsequent spatial undivided differences,

$$r = \frac{u_{i+1} - u_i}{u_i - u_{i-1}}, \tag{3.3}$$

where r is forced to be ≥ 0. A function $\phi(r)$ defines how to balance the low and high order flux terms based on the smoothness parameter r. In order for the method to be stable and TVD, $\phi(r)$ is limited by the following restrictions:

- $\phi(r) \geq 0$,
- $\phi(1) = 1$,
- $\phi(r)$ lies completely in the TVD region of the high order scheme.

Van Leer [29] implemented a flux limiter by setting bounds to the solution slopes such that they would not create new local extrema. This approach effectively forces the solution to stay monotone. Many limiter functions $\phi(r)$ have been created over the past few decades with varying levels of success. Flux limiting has been adopted in schemes made by among others Roe [23], Chakravarthy and Osher [19] and Harten [8]. The disadvantage of flux limiters is that they tend to wrongly activate at extrema in smooth regions of the solution, introducing an inordinate amount of dissipation at these locations and quickly flattening smooth peaks. Examples of this flattening effect are shown in Figure 3.2 for several limiter functions.

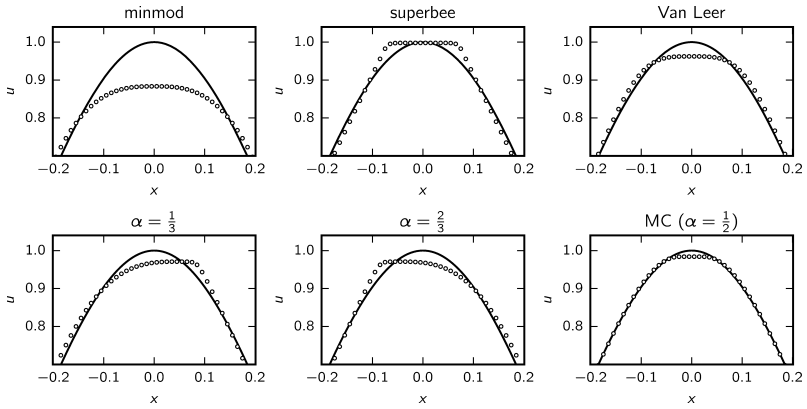


Figure 3.2: Flattening of smooth extrema due to the usage of flux limiters. Taken from [13].

3.3. Direct dissipation manipulation

An obvious way to control dissipation rate is to use a method that allows direct control on numerical dissipation through some adjustable parameter. An example of such a method can be found within the class of Conservation Element Solution Element (CESE) schemes. The original formulation of a CESE scheme designed by Chang [7] is the $a-\mu$ scheme. This scheme uses a mesh staggered in both space and time, marching two variables u and u_x independently. The scheme is symmetric in space and time, causing it to have no numerical dissipation and be reversible. Chang recognized that such a method cannot be used reliably for the inviscid Euler equations due to the presence of non-reversible phenomena such as shock waves. For this reason, he altered the original $a-\mu$ scheme to the $a-\epsilon$ scheme, which incorporates a controllable amount of numerical dissipation proportional to the adjustable parameter $0 \leq \epsilon \leq 1$, where $\epsilon = 0$ corresponds to no dissipation and $\epsilon = 1$ indicates a high dissipation rate. Although flexible, this method requires the user to define both *where* and *how much* dissipation should be applied, which is found on a trial and error basis.

3.4. Mesh adaption

The leading term of the truncation error for a numerical method is of the form $\mathcal{O}(\Delta x^n)$, where Δx designates mesh spacing and n the order of the method. Thus, besides increasing the order, decreasing the mesh spacing also results in reduced numerical dissipation. However, decreasing the mesh spacing globally may drastically increase computational effort. In order to ensure stability by satisfying the CFL condition, smaller mesh spacing consequences in smaller time steps. Thus, halving the mesh spacing means that the amount of calculations multiply by a factor of 2^{N+1}, where N is the number of spatial dimensions. Because of this exponential growth of computational effort, a research field exists that focuses on nonuniform mesh adaption, often referred to as adaptive mesh refinement (AMR). It is an active research area that aims at handling the temporally and spatially varying solution resolution requirement that is present in many numerical problems. Many of such problems vary in complexity throughout the computational domain, and some regions within this domain thus are of a higher information density

than others. In these cases, the use of a uniform grid is a compromise between an unnecessary waste of resources in smooth regions when using a fine grid, or a deteriorated solution accuracy in complex regions in case of a coarse grid. A locally refined grid allows the discrete domain to mimic variations in resolution that occur in the continuous domain. A refined static mesh provides a good approach for increased local resolution while avoiding the bookkeeping and increased computational load of an adaptive method, but requires a priori information on the converged solution structure and is for this reason often unfeasible for time-dependent problems. A logical train of thought is then to somehow let the solution method vary such that it can adapt to the local environment. Besides computational fluid dynamics, mesh adaption is frequently used in astrophysical simulations [2, 12, 20], which are often tied to extreme variability of time and/or length scales.

3.4.1. Types of adaption methods

In general, methods of adapting solution resolution can be categorized in the following three approaches:

- ***p*-refinement** Does not alter grid geometry, but controls accuracy by varying the order of the solution method;
- ***r*-refinement** Involves moving and/or stretching of the grid, see Figure 3.3;
- ***h*-refinement** Alters grid resolution without distortion by introducing or removing grid points hierarchically, see Figure 3.4.

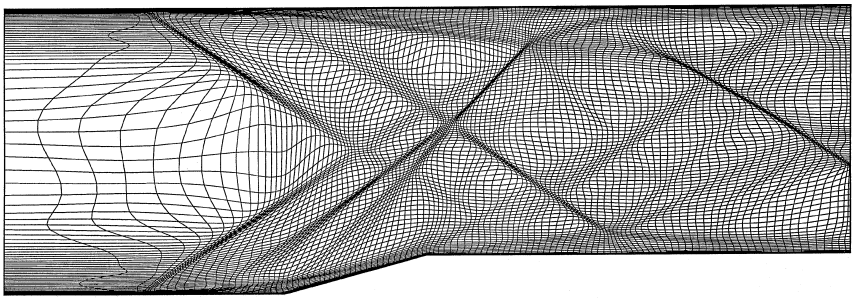


Figure 3.3: An example of *r*-refinement on a shock wave interaction problem. Taken from [17].

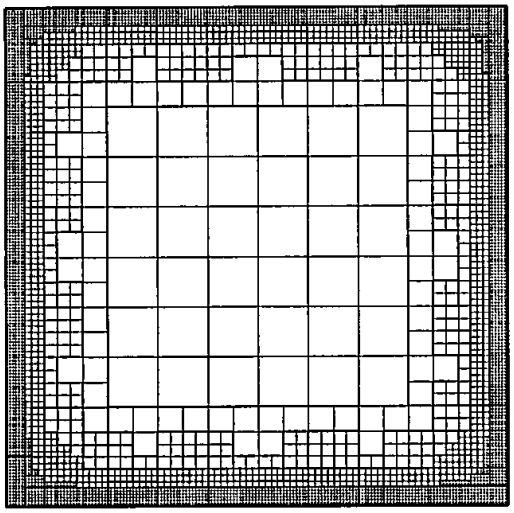


Figure 3.4: An example of *h*-refinement. Taken from [9].

Although *p*-refinement techniques have proven accurate in smooth fluid problems, they are generally less successful when applied in the presence of discontinuities [1]. Higher order methods applied

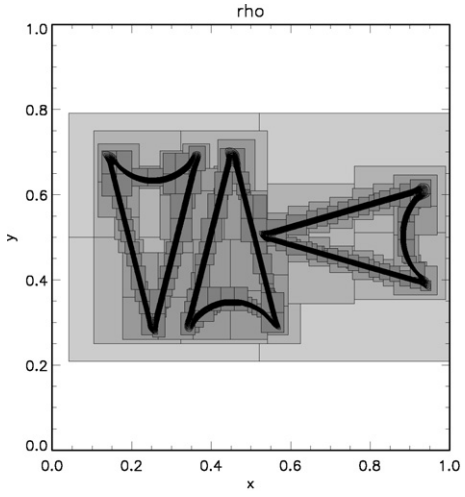
to high-frequency phenomena tend to stimulate nonphysical oscillations and overshoots through the Gibbs phenomenon.

A disadvantage of *r*-refinement is that the amount of grid points is constant, and thus requires a prediction on how the complexity of the solution will evolve over time to guarantee a good enough solution accuracy during the complete simulation [4]. The formation of a shock wave is an example where a smooth solution transitions to a more complex structure and, depending on the implementation, this may cause a pileup of grid points at the shock location and near-vacuum grid densities elsewhere, resulting in a highly deformed grid. The method of *h*-refinement is in this sense more flexible and less coupled in that alterations of the grid structure in some region do not affect the remainder of the domain, which is one of the reasons for its popularity in fluid problems [5, 25, 30]. Since shock development and propagation problems are of interest in this thesis, *h*-refinement is the most suitable approach for controlling the local solution resolution and is therefore focused on here. The method of *h*-refinement can be further categorized in patch based and cell-based methods, see Figure 3.5.

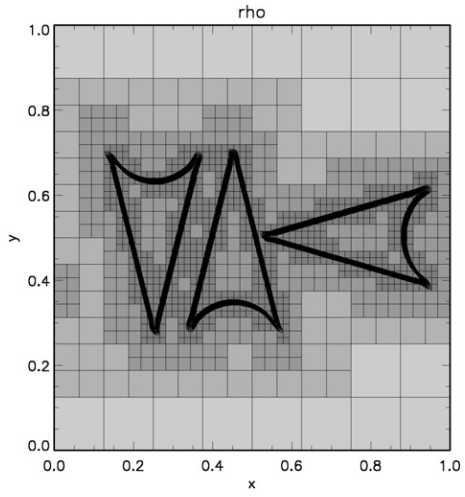
In cell-based refinement, flagged cells are recursively refined individually. On the other hand, patch-based methods first create clusters of flagged cells and enclose each cluster by a rectangular patch in which all cells are refined. This method allows for easier parallelization and can thus be better optimized compared to cell-based methods. A disadvantage is that the code has to incorporate additional algorithms for the clustering step.

3.4.2. Adaption criteria

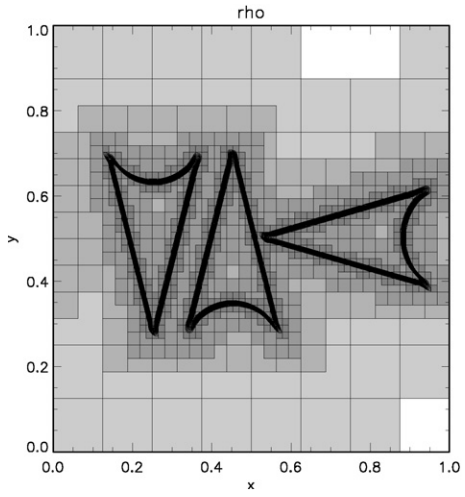
Besides the bookkeeping that is often the source of headaches in h-refinement code design, another hard part is the decision making that is involved in the adaption step. In literature, often the concept of equal distribution of error is used to decide where to coarsen or refine the mesh [1, 3, 21]. Of course, the exact solution has to be known in order to determine the error exactly, so a method to estimate the error is required. One such method is Richardson extrapolation, where an approximation of the first truncation error term of the numerical method is retrieved by applying the numerical scheme to overlapping meshes with different levels of uniform refinement. Thus, this error estimator is ideal for patch-based codes, but less intuitive to implement in cell-based codes. Another error indicator that is more applicable to individual cells is referred to as feature detection, and is used in other research fields such as image and signal processing as well. Feature detection aims at identifying relatively complex regions in data, often through analysis of first or higher order derivatives of variables of interest [1, 10]. Feature detection is also an important aspect of hybrid methods, commonly used as switching function.



(a)



(b)



(c)

Figure 3.5: Three types of *h*-refinement, taken from [28]. (a) Patch-based refinement. (b) Cell-based refinement. (c) Hybrid tree-patch refinement.

4

Active Flux

A relatively new class of numerical approaches for solving hyperbolic conservation laws are Active Flux schemes. The term Active Flux was introduced in 2011 in a conference paper by Eymann and Roe [24]. The core concept of these schemes is an implementation of van Leer's train of thought of increasing a scheme's order of accuracy by introducing additional degrees of freedom intra-cell rather than extra-cell. This prevents extending the computational stencil beyond the physical domain of influence. In an Active Flux method, this is accomplished through modification of a finite volume method by adding an extra variable on the interfaces separating the computational cells. This creates a method where interface values and cell average values are updated differently. Updating the cell average values in a conservative way leaves the cell interface values to be updated through a method that need not be conservative. This is the same as a regular finite volume method, where now the flux function is to be expressed in terms of cell interface values and cell average values explicitly. With $u_{i+\frac{1}{2}}$ representing the interface value at $x_{i+\frac{1}{2}}$, and $\overline{u}_i$ the average integral value at x_i, the finite volume update method equals:

$$\overline{u}_i^{n+1} = \overline{u}_i^n - \frac{1}{\Delta x}\int_{t_n}^{t_{n+1}} f_{i+\frac{1}{2}}(t) - f_{i-\frac{1}{2}}(t)\mathrm{d}t, \tag{4.1}$$

where the flux f is thus a function of u and $\overline{u}$. What remains is defining $f(t)$, and choosing a method to update the interface values in time. Using e.g. an exact Riemann solver with $\overline{u}$ on both sides of the interfaces as input states, one arrives at a form of Godunov's first order method.

4.1. Eymann and Roe's implementation

Eymann and Roe use the properties of characteristics to find the interface values at the next time level. The advantage of this is that this method inherently preserves directionality in the flow, thereby adhering to physical constraints. Assume a scalar conservation law in one spatial dimension, see (2.10). We have seen that u is the characteristic variable that is advected with velocity $\frac{\partial f(u)}{\partial u}$. Thus, if the characteristic pattern is known, it is possible to determine the solution values on the cell interfaces at an elevated time level by tracing along the characteristics to a time level at which the solution is known, see Figure 4.1.

$$\frac{\partial u}{\partial t} + \frac{\partial f(u)}{\partial x} = 0. \tag{2.10}$$

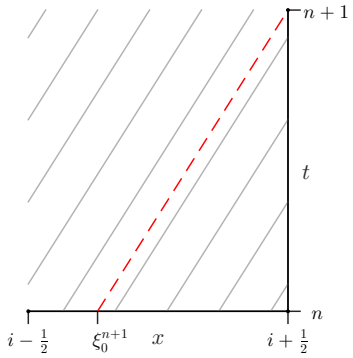


Figure 4.1: Visualization of how character tracing is used to find interface values at an elevated time level.

As can be seen in this figure, the interface value $u_{i+1/2}$ at time level $n+1$ is equal to the solution value $u\left(\xi_0^{n+1}\right)$ at time level n. Because the solution is only known at discrete points, a local interpolation of the solution on each cell's interior domain is required to estimate the value of $u(\xi_0)$. To keep the stencil compact, only the information available within each cell is used to construct the interpolation. For k knowns, a sum of k weighted basis functions can be used as interpolation:

$$u(\xi) = \sum_k c_k \phi_k(\xi), \tag{4.2}$$

where $\xi = \frac{x - x_{i-1/2}}{\Delta x}$ is a local scaling parameter such that $\xi = 0$ and $\xi = 1$ match the left and right cell interface respectively. In this case, one average cell integral value $\overline{u}$ and two interface values u are known for each cell, so k ranges from 1 to 3. Eymann and Roe use second order polynomials as basis functions, meaning that the interpolation $u(\xi)$ also takes the form of a second order polynomial:

$$u(\xi) = \alpha\xi^2 + \beta\xi + \gamma. \tag{4.3}$$

The coefficients α, β, and γ are determined by the following constraints:

- $u(0) = u_{i-\frac{1}{2}}$ (left interface),
- $u(1) = u_{i+\frac{1}{2}}$ (right interface),
- $\frac{1}{\Delta x}\int_0^{\Delta x} u(x)\mathrm{d}x = \int_0^1 u(\xi)\mathrm{d}\xi = \overline{u}_i$.

This results in the following system:

$$\begin{bmatrix} 0 & 0 & 1 \\ 1 & 1 & 1 \\ \frac{1}{3} & \frac{1}{2} & 1 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \\ \gamma \end{bmatrix} = \begin{bmatrix} u_{i-\frac{1}{2}} \\ u_{i+\frac{1}{2}} \\ \overline{u}_i \end{bmatrix}, \tag{4.4}$$

the solution of which provides the values of α, β and γ:

$$\begin{bmatrix} \alpha \\ \beta \\ \gamma \end{bmatrix} = \begin{bmatrix} 3u_{i-\frac{1}{2}} - 6\overline{u}_i + 3u_{i+\frac{1}{2}} \\ -4u_{i-\frac{1}{2}} + 6\overline{u}_i - 2u_{i+\frac{1}{2}} \\ u_{i-\frac{1}{2}} \end{bmatrix}. \tag{4.5}$$

Substitution into (4.3) yields:

$$\begin{aligned} u(\xi) &= \alpha\xi^2 + \beta\xi + \gamma && \text{(4.6a)} \\ &= \left(3u_{i-\frac{1}{2}} - 6\overline{u}_i + 3u_{i+\frac{1}{2}}\right)\xi^2 + \left(-4u_{i-\frac{1}{2}} + 6\overline{u}_i - 2u_{i+\frac{1}{2}}\right)\xi + u_{i-\frac{1}{2}} && \text{(4.6b)} \\ &= \left(3\xi^2 - 4\xi + 1\right)u_{i-\frac{1}{2}} + \left(-6\xi^2 + 6\xi\right)\overline{u}_i + \left(3\xi^2 - 2\xi\right)u_{i+\frac{1}{2}}. && \text{(4.6c)} \end{aligned}$$

This final expression contains three quadratic equations that give a more intuitive view on the construction of the interpolation. Visualized in Figure 4.2, their properties are as follows:

$\phi(\xi)$	$\phi(0)$	$\int_0^1 \phi(\xi)\,\mathrm{d}\xi$	$\phi(1)$
$3\xi^2-4\xi+1$	1	0	0
$-6\xi^2+6\xi$	0	1	0
$3\xi^2-2\xi$	0	0	1

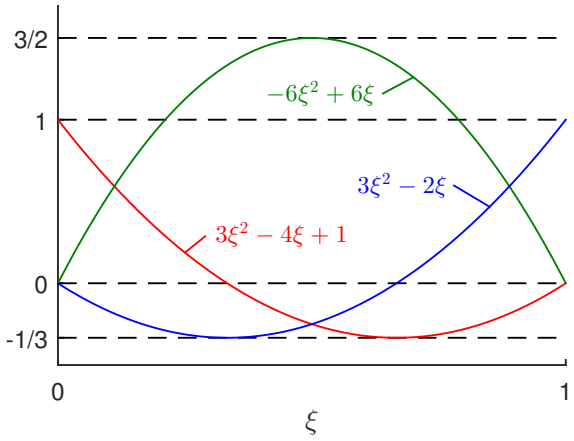


Figure 4.2: The three polynomial basis functions used to reconstruct the solution.

The final step requires a method to find the characteristic origin ξ_0 given the characteristic pattern. Although the direction of the characteristics completely depends on $\frac{\partial f}{\partial u}$, we know that the characteristics are straight as u is constant along $\frac{\mathrm{d}x}{\mathrm{d}t} = f(u)$, and thus $\frac{\mathrm{d}x}{\mathrm{d}t}$ is constant. It then becomes trivial to select the cell from which the characteristic originates and the goal of finding its origin is simplified to solving the following implicit problem:

$$\xi_0 = \begin{cases} 1 - \left.\frac{\partial f}{\partial u}\right|_{\xi_0} \frac{\Delta t}{\Delta x} & \text{if } \left.\frac{\partial f}{\partial u}\right|_{\xi_0} > 0 \quad \text{(cell left of interface)} \\ \left.\frac{\partial f}{\partial u}\right|_{\xi_0} \frac{\Delta t}{\Delta x} & \text{if } \left.\frac{\partial f}{\partial u}\right|_{\xi_0} < 0 \quad \text{(cell right of interface)} \\ 0 & \text{if } \left.\frac{\partial f}{\partial u}\right|_{\xi_0} = 0. \end{cases} \tag{4.7}$$

The expression for the flux integral is now known as well:

$$F_{i+\frac{1}{2}} = \int_{t^n}^{t^{n+1}} f_{i+\frac{1}{2}}(t)\mathrm{d}t \tag{4.8a}$$

$$= \int_{t^n}^{t^{n+1}} f_{i+\frac{1}{2}}\left(u_{i+\frac{1}{2}}(t)\right)\mathrm{d}t \tag{4.8b}$$

$$= \int_{t^n}^{t^{n+1}} f_{i+\frac{1}{2}}\left(u^n(\xi_0(t))\right)\mathrm{d}t. \tag{4.8c}$$

Because the flux is derived from u, whose approximation is quadratic and thus second order accurate, a second order approximation of the integral in (4.8c) is sufficient; higher orders will not increase accuracy. Eymann suggests to use Simpson's rule, for which an additional flux value at intermediate time level $t^{n+\frac{1}{2}}$ is required, meaning that a second characteristic origin has to be found. The total amount of u that has crossed the interface between time levels t^n and t^{n+1}, F, is then evaluated by:

$$F_{i+\frac{1}{2}} = \frac{\Delta t}{6}\left(f^n_{i+\frac{1}{2}} + 4f^{n+\frac{1}{2}}_{i+\frac{1}{2}} + f^{n+1}_{i+\frac{1}{2}}\right). \tag{4.9}$$

Adopting the notation ξ^n_0 as the characteristic origin of the characteristic that intersects the interface

at time level n, as already used in Figure 4.1, the flux terms become:

$$f^n_{i+\frac{1}{2}} = f_{i+\frac{1}{2}}\left(u^n\left(\xi^0_0\right)\right) = f_{i+\frac{1}{2}}\left(u^n_{i+\frac{1}{2}}\right), \tag{4.10a}$$

$$f^{n+\frac{1}{2}}_{i+\frac{1}{2}} = f_{i+\frac{1}{2}}\left(u^n\left(\xi^{n+\frac{1}{2}}_0\right)\right), \tag{4.10b}$$

$$f^{n+1}_{i+\frac{1}{2}} = f_{i+\frac{1}{2}}\left(u^n\left(\xi^{n+1}_0\right)\right). \tag{4.10c}$$

4.1.1. Linear advection

For the linear advection equation, $\frac{\partial f}{\partial u}$ is constant throughout the whole domain, which means that (4.7) becomes explicit and the characteristic origin is easily found by substituting $\frac{\partial f}{\partial u} = a$. Assuming $a > 0$:

$$\xi^{n+1}_0 = 1 - \frac{a\Delta t}{\Delta x}. \tag{4.11}$$

Or, in terms of the Courant number, denoted by ν:

$$\xi^{n+1}_0 = 1 - \nu. \tag{4.12}$$

Using symmetry, inserting this into (4.6c) results in the following expression for u at the characteristic origin:

$$u^n\left(\xi^{n+1}_0\right) = \left(3\nu^2 - 2\nu\right)u^n_{i-\frac{1}{2}} + \left(-6\nu^2 + 6\nu\right)\overline{u}^n_i + \left(3\nu^2 - 4\nu + 1\right)u^n_{i+\frac{1}{2}}. \tag{4.13}$$

This equation is also used to update the interface values to the next time level. In the previous section we have seen that the characteristic origin halfway the time step is required to find a second order accurate representation of the integral flux F. Replacing Δt by $\frac{\Delta t}{2}$ in (4.11) yields:

$$\begin{aligned}\xi^{n+\frac{1}{2}}_0 &= 1 - \frac{a\Delta t}{2\Delta x}\\ &= 1 - \frac{\nu}{2}.\end{aligned} \tag{4.14}$$

And thus:

$$u^n\left(\xi^{n+\frac{1}{2}}_0\right) = \left(\frac{3}{4}\nu^2 - \nu\right)u^n_{i-\frac{1}{2}} + \left(-\frac{3}{2}\nu^2 + 3\nu\right)\overline{u}^n_i + \left(\frac{3}{4}\nu^2 - 2\nu + 1\right)u^n_{i+\frac{1}{2}}. \tag{4.15}$$

Since for the linear advection equation the flux function is $f = au$, the integral flux value is found by substituting (4.13) and (4.15) into (4.9):

$$\begin{aligned}F_{i+\frac{1}{2}} &= \frac{\Delta t}{6}\left(f^n_{i+\frac{1}{2}} + 4f^{n+\frac{1}{2}}_{i+\frac{1}{2}} + f^{n+1}_{i+\frac{1}{2}}\right)\\ &= \frac{a\Delta t}{6}\left(u^n_{i+\frac{1}{2}} + 4u^{n+\frac{1}{2}}_{i+\frac{1}{2}} + u^{n+1}_{i+\frac{1}{2}}\right)\\ &= \frac{a\Delta t}{6}\left(u^n_{i+\frac{1}{2}} + 4u^n\left(\xi^{n+\frac{1}{2}}_0\right) + u^n\left(\xi^{n+1}_0\right)\right)\\ &= a\Delta t\left(\left(\nu^2 - \nu\right)u^n_{i-\frac{1}{2}} + \left(-2\nu^2 + 3\nu\right)\overline{u}^n_i + \left(\nu^2 - 2\nu + 1\right)u^n_{i+\frac{1}{2}}\right).\end{aligned} \tag{4.16}$$

After calculating the integral flux values at each cell interface, the average integral cell values can be updated through:

$$\begin{aligned}\overline{u}^{n+1}_i &= \overline{u}^n_i - \frac{1}{\Delta x}\left(F_{i+\frac{1}{2}} - F_{i-\frac{1}{2}}\right)\\ &= \overline{u}^n_i - \left(-2\nu^3 + 3\nu^2\right)\left(\overline{u}^n_i - \overline{u}^n_{i-1}\right) + (\nu - 1)\nu\left(\nu u^n_{i-\frac{3}{2}} + (1-\nu)u^n_{i+\frac{1}{2}} - u^n_{i-\frac{1}{2}}\right).\end{aligned} \tag{4.17}$$

4.1.2. Analysis

Although the Active Flux method essentially doubles the degrees of freedom by introducing an additional variable, this form where point updates are isolated from integral updates makes it trivial to adhere to conservation laws. However, due to this compound formulation of the scheme, standard analysis techniques cannot be applied directly. Therefore, the scheme is rewritten to a single multistep form that can be applied to a standard uniform grid. This reformulation is derived in Appendix A, where the grid point update expression is found to be:

$$\begin{aligned} u_i^{n+1} =& 2\nu\left(-1+3\nu-\nu^2\right)u_{i-1}^n \\ &+2\left(1-\nu\right)\left(1-\nu-\nu^2\right)u_i^n \\ &-\nu^4 u_{i-2}^{n-1} \\ &+2\nu(1-\nu)(1+\nu-\nu^2)u_{i-1}^{n-1} \\ &-\left(1-\nu\right)^4 u_i^{n-1}. \end{aligned} \tag{4.18}$$

This form shows that the scheme uses a stencil as shown in Figure 4.3, where each grid point value is multiplied by a constant. The sum of these constants equals one, so the new value essentially is a weighted average of the five grid point values. The constants are polynomial functions of the Courant number, and it is not hard to see that for Courant numbers of 0 and 1, (4.18) reduces to a first order upwind scheme along the characteristics, which is exact if the characteristic variable is constant.

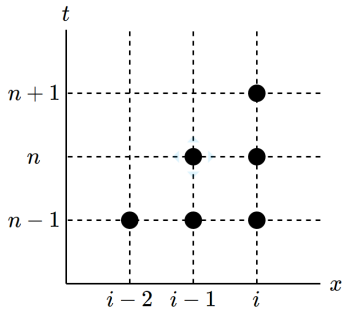


Figure 4.3: Stencil used for Eymann's Active Flux scheme.

The form presented in (4.18) lends itself to a von Neumann stability analysis which reveals some of the error development characteristics. Since this is a multistep method, the amplification factor is obtained in the form of a matrix of size $N \times N$, where N is the amount of time levels over which the used grid points are distributed, which in this case equals 2 (n and $n-1$). Stability then requires that the magnitudes of all N eigenvalues retrieved from this matrix are ≤ 1. For (4.18), the amplification matrix is as follows:

$$\boldsymbol{G} = \begin{bmatrix} 2\left(1-\nu\right)\left(1-\nu-\nu^2\right)+2\nu\left(-1+3\nu-\nu^2\right)e^{-i\phi} & 2\nu(1-\nu)(1+\nu-\nu^2)e^{-i\phi}-(1-\nu)^4-\nu^4 e^{-2i\phi} \\ 1 & 0 \end{bmatrix}. \tag{4.19}$$

The characteristic polynomial for this matrix reads:

$$\begin{aligned} -\lambda\Big(2\left(1-\nu\right)\left(1-\nu-\nu^2\right)+2\nu\left(-1+3\nu-\nu^2\right)e^{-i\phi}-\lambda\Big) \\ -2\nu(1-\nu)(1+\nu-\nu^2)e^{-i\phi}+(1-\nu)^4+\nu^4 e^{-2i\phi}=0. \end{aligned} \tag{4.20}$$

Solving for the two eigenvalues $\lambda_{1,2}$ yields:

$$\lambda_1 = -(e^{-i\phi}-1)\nu^3 + 3e^{-i\phi}\nu^2 - (e^{-i\phi}+2)\nu + 1 + (\nu-1)\nu\sqrt{e^{-2i\phi}(\nu^2-4\nu+1) - 2e^{-i\phi}(\nu^2-\nu-5) + \nu^2 + 2\nu - 2}\,, \tag{4.21}$$

$$\lambda_2 = -(e^{-i\phi}-1)\nu^3 + 3e^{-i\phi}\nu^2 - (e^{-i\phi}+2)\nu + 1 - (\nu-1)\nu\sqrt{e^{-2i\phi}(\nu^2-4\nu+1) - 2e^{-i\phi}(\nu^2-\nu-5) + \nu^2 + 2\nu - 2}\,. \tag{4.22}$$

Both eigenvalues are visualized in Figure 4.4, from which can be derived that the method is stable for Courant numbers up to one.

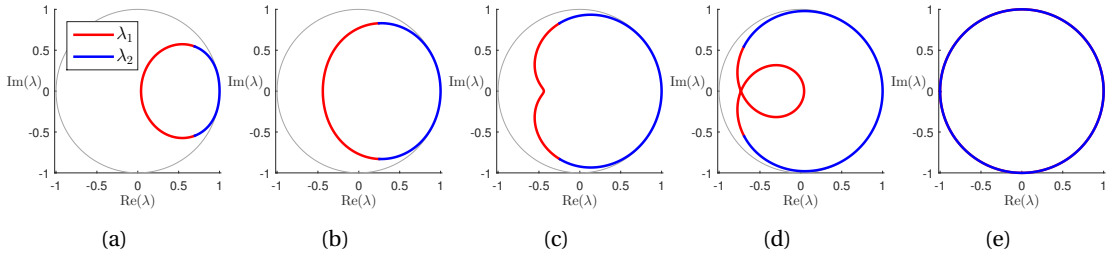


Figure 4.4: Eigenvalues for Courant numbers of 0.2 (a), 0.4 (b), 0.6 (c), 0.8 (d) and 1.0 (e).

4.2. Multistep scheme generation

As shown in the previous section, the interface update method of Eymann and Roe's Active Flux scheme can be expressed in the form of a weighted average. In general, a linear update method can be constructed by choosing which grid points to use and which weights to assign to these points. This is equivalent to using finite differences to approximate derivatives to a certain degree of accuracy. One way to distribute the weights is by minimizing the Taylor series expansion truncation error (TSETE). This constraint allows us to generate update methods of arbitrary order based solely on a selection of grid points. On a uniform grid, a two dimensional Taylor series expansion can be used to relate the values of two distinct grid nodes in time and space. If two nodes are separated by i spatial intervals and j temporal intervals, as shown in Figure 4.5, the Taylor series expansion is equal to:

$$u\left(x+i\Delta x, t+j\Delta t\right) = \sum_{k=0}^{\infty} \frac{1}{k!}\left(i\Delta x\frac{\partial}{\partial x} + j\Delta t\frac{\partial}{\partial t}\right)^k u(x,t). \tag{4.23}$$

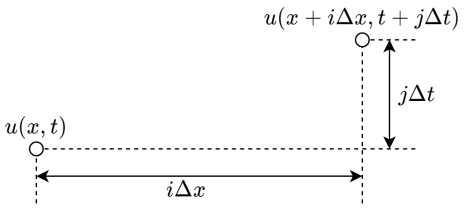


Figure 4.5: Auxiliary visualization on the relation between two nodes through Taylor series expansion.

Since a uniform grid is assumed here, we can shorten the notation by using indices when referring to grid points. Suppose the unknown grid point value at the next time step is designated by u_m^{n+1}. The Taylor expansion with respect to a node at $\left(m+i, n+1+j\right)$ then equals:

$$u_{m+i}^{n+1+j} = \sum_{k=0}^{\infty} \frac{1}{k!}\left(i\Delta x\frac{\partial}{\partial x} + j\Delta t\frac{\partial}{\partial t}\right)^k u_m^{n+1}. \tag{4.24}$$

For the linear advection equation, we know that $\frac{\partial u}{\partial t} = -a\frac{\partial u}{\partial x}$, and this expression can thus be rewritten to:

$$u_{m+i}^{n+1+j} = \sum_{k=0}^{\infty} \frac{1}{k!} \left(i\Delta x - aj\Delta t\right)^k \frac{\partial^k u_m^{n+1}}{\partial x^k}. \tag{4.25}$$

A linear combination of the Taylor series expansions with respect to N nodes allows a value estimation with a truncation error on the order of $O\left(\Delta x^N + \Delta t^N\right)$. This information can be used to generate optimal (in a minimum truncation error sense) update methods, by solely specifying which nodes to use. The general form of the update method is expressed as:

$$u_m^{n+1} = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} c_i^j u_{m+i}^{n+j}, \tag{4.26}$$

which, considering only explicit schemes, reduces to:

$$u_m^{n+1} \approx \sum_{i=-\infty}^{\infty} \sum_{j=0}^{\infty} c_i^j u_{m+i}^{n-j}. \tag{4.27}$$

The coefficients c_i^j are found by solving the system

$$\boldsymbol{A}^{\boldsymbol{T}}\boldsymbol{c} = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix},$$

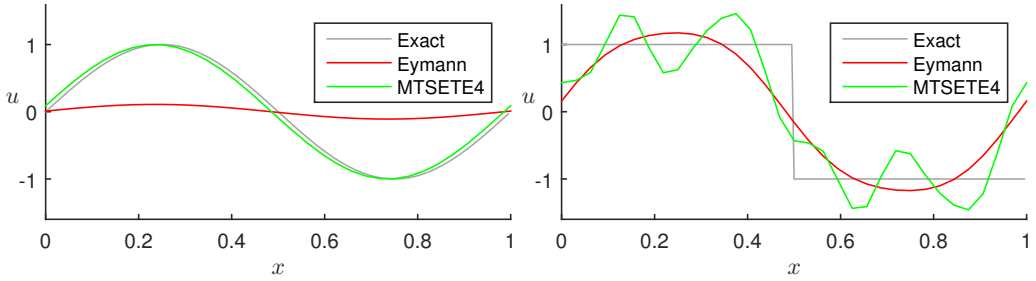
where $\boldsymbol{A}$ contains the expansion coefficients:

$$\boldsymbol{A} = \begin{bmatrix} 1 & i_1\Delta x - aj_1\Delta t & \dots & \frac{(i_1\Delta x - aj_1\Delta t)^{N-1}}{(N-1)!} \\ 1 & i_2\Delta x - aj_2\Delta t & \dots & \frac{(i_2\Delta x - aj_2\Delta t)^{N-1}}{(N-1)!} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & i_N\Delta x - aj_N\Delta t & \dots & \frac{(i_N\Delta x - aj_N\Delta t)^{N-1}}{(N-1)!} \end{bmatrix}.$$

When applying this procedure to the stencil shown in Figure 4.3, we end up with the following update method, which will be referred to as MTSETE4:

$$\begin{aligned} u_i^{n+1} = &-4\nu\frac{1-2\nu}{\nu+1}u_{i-1}^n \\ &-4\frac{(2\nu-1)(\nu-1)}{\nu-2}u_i^n \\ &-\nu^2\frac{1-2\nu}{\nu-2}u_{i-2}^{n-1} \\ &+4\nu(1-\nu)u_{i-1}^{n-1} \\ &+\frac{(\nu-1)^2(2\nu-1)}{\nu+1}u_i^{n-1}. \end{aligned} \tag{4.28}$$

This method is 4^{th} order accurate. Unlike Eymann and Roe's method, which is exact for Courant numbers of 0 and 1, this method is also exact for a Courant number of $\frac{1}{2}$. Since its order of accuracy is one higher, it has a lower dissipation rate and instead dispersive errors are expected to be more apparent, see Figure 4.6a. Near discontinuities however, this method suffers from highly oscillatory behaviour,



(a) (b)

Figure 4.6: Solutions of a smooth and discontinuous waveform on a 32 node grid and a Courant number of 0.8. (a) Solutions of a sine wave after 20000 domain traversals. (b) Solutions of a square wave after 200 domain traversals.

while Eymann and Roe's Active Flux method dampens out such oscillations over time, see Figure 4.6b. It appears that an order of accuracy is exchanged for reduced spurious oscillatory behavior.

To determine the stability of this scheme, we can again set up an amplification matrix and find the eigenvalues of this matrix. The amplification matrix equals:

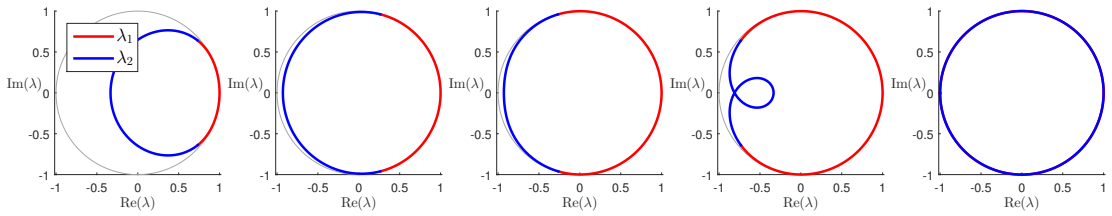
$$\boldsymbol{G} = \begin{bmatrix} -4\frac{(2\nu-1)(\nu-1)}{\nu-2} - 4\nu\frac{1-2\nu}{\nu+1}e^{-i\phi} & 4\nu(1-\nu)e^{-i\phi} + \frac{(\nu-1)^2(2\nu-1)}{\nu+1} - \nu^2\frac{1-2\nu}{\nu-2}e^{-2i\phi} \\ 1 & 0 \end{bmatrix}, \tag{4.29}$$

whose eigenvalues are:

$$\begin{aligned} \lambda_1 = &\frac{4\nu^3(e^{-i\phi}-1)+\nu^2(2-10e^{-i\phi})+4\nu(e^{-i\phi}+1)-2}{(\nu-2)(\nu+1)} \\ &+ \frac{3(\nu-1)\nu\sqrt{2\nu^2(e^{-i\phi}-1)^2+\nu(-5e^{-2i\phi}+4e^{-i\phi}+1)+2e^{-2i\phi}+8e^{-i\phi}-1}}{(\nu-2)(\nu+1)}, \end{aligned} \tag{4.30}$$

$$\begin{aligned} \lambda_2 = &\frac{4\nu^3(e^{-i\phi}-1)+\nu^2(2-10e^{-i\phi})+4\nu(e^{-i\phi}+1)-2}{(\nu-2)(\nu+1)} \\ &- \frac{3(\nu-1)\nu\sqrt{2\nu^2(e^{-i\phi}-1)^2+\nu(-5e^{-2i\phi}+4e^{-i\phi}+1)+2e^{-2i\phi}+8e^{-i\phi}-1}}{(\nu-2)(\nu+1)}. \end{aligned} \tag{4.31}$$

Plots of both eigenvalues, see Figure 4.7, shows that this method is stable up to $\nu = 1$.



(a) (b) (c) (d) (e)

Figure 4.7: Eigenvalues for the fourth order minimum error scheme. Courant numbers of 0.2 (a), 0.4 (b), 0.6 (c), 0.8 (d) and 1.0 (e).

Using the TSETE as parameter to minimize, each unique stencil yields one scheme. Stability and accuracy are however not guaranteed by this method, so not every stencil results in a useful scheme.

4.3. Third order multistep scheme

A stencil whose corresponding update method is of the same order as the Active Flux method by Eymann and Roe is displayed in Figure 4.8. Although the order is the same, the stencil is more compact.

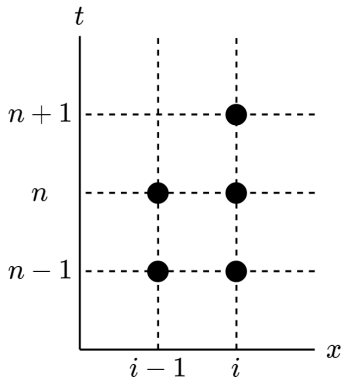


Figure 4.8: Stencil resulting in a third order accurate update method.

The update method for this stencil reads:

$$\begin{aligned} u_i^{n+1} = &-2\nu\frac{1-2\nu}{\nu+1}u_{i-1}^n \\ &+2(1-2\nu)u_i^n \\ &+2\nu u_{i-1}^{n-1} \\ &-\frac{(2\nu-1)(\nu-1)}{\nu+1}u_i^{n-1}. \end{aligned} \tag{4.32}$$

The amplification matrix for this method equals:

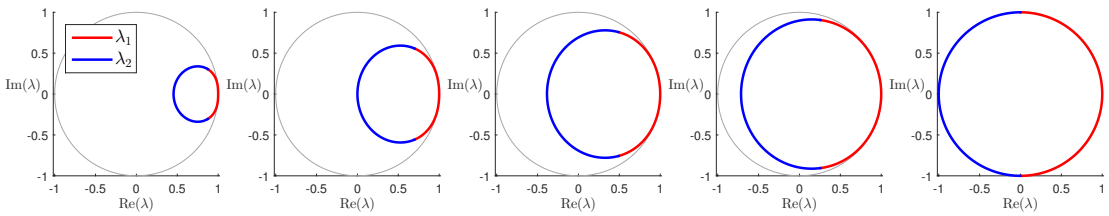
$$\boldsymbol{G} = \begin{bmatrix} -2\nu\frac{1-2\nu}{\nu+1}e^{-i\phi}+2(1-2\nu) & 2\nu e^{-i\phi}-\frac{(2\nu-1)(\nu-1)}{\nu+1} \\ 1 & 0 \end{bmatrix}, \tag{4.33}$$

whose eigenvalues equal:

$$\begin{aligned} \lambda_1 =&(2\nu-1)\frac{\nu\left(e^{-i\phi}-1\right)-1}{\nu+1} \\ &+\frac{\nu\sqrt{(2\nu-1)^2e^{-2i\phi}+2(-4\nu+5)(\nu+1)e^{-i\phi}+2(\nu+1)(2\nu-1)}}{\nu+1}, \end{aligned} \tag{4.34}$$

$$\begin{aligned} \lambda_2 =&(2\nu-1)\frac{\nu\left(e^{-i\phi}-1\right)-1}{\nu+1} \\ &-\frac{\nu\sqrt{(2\nu-1)^2e^{-2i\phi}+2(-4\nu+5)(\nu+1)e^{-i\phi}+2(\nu+1)(2\nu-1)}}{\nu+1}. \end{aligned} \tag{4.35}$$

In Figure 4.9 the eigenvalues are visualized, in which can be seen that this method is only stable up to a Courant number of 0.5.



(a) (b) (c) (d) (e)

Figure 4.9: Eigenvalues for the third order minimum error scheme. Courant numbers of 0.1 (a), 0.2 (b), 0.3 (c), 0.4 (d) and 0.5 (e).

Similar to how (4.18) was derived in Appendix A, this process can be reversed to find the Active Flux form of the above update method, see Appendix B. This results in the following:

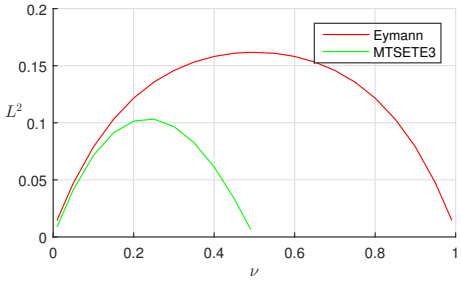
$$u_{i+\frac{1}{2}}^{n+1} = u_{i+\frac{1}{2}}^{n} - 2\nu\left(\frac{u_{i-\frac{1}{2}}^{n} - 3\overline{u}_{i}^{n} + 2u_{i+\frac{1}{2}}^{n}}{\nu+1} + \nu\frac{u_{i-\frac{1}{2}}^{n} - u_{i+\frac{1}{2}}^{n}}{\nu+1}\right) \quad \text{(Node update).} \tag{4.36}$$

$$\overline{u}_{i}^{n+1} = \overline{u}_{i}^{n} - \nu\left(u_{i+\frac{1}{2}}^{n+1} - u_{i-\frac{1}{2}}^{n+1}\right) \quad \text{(Integral update).} \tag{4.37}$$

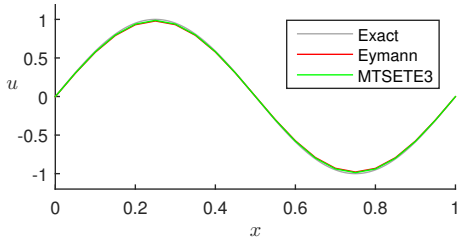
This method, from here on referred to as MTSETE3, being of the same order as Eymann's Active Flux method, shows similar characteristics. In order to compare the two methods in terms of accuracy, the L^2 error is used, which is calculated through:

$$L^2 = \sqrt{\frac{\sum_{i=1}^{N}\left(u_i - u_{i,exact}\right)^2}{N}}, \tag{4.38}$$

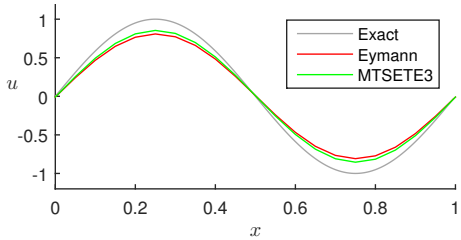
in which N designates the amount of nodes. Figure 4.10a shows how the L^2 errors of a simulation of a sine wave vary with the Courant number. For lower Courant numbers, the methods behave almost equally, as can be seen in Figure 4.10b. As the Courant number increases, the errors diverge in favor of the MTSETE3 method. The lower dissipation rate and increased accuracy for higher Courant numbers can be observed in Figure 4.10d. For a square wave, similar error patterns are obtained, as summarized in Figure 4.11.



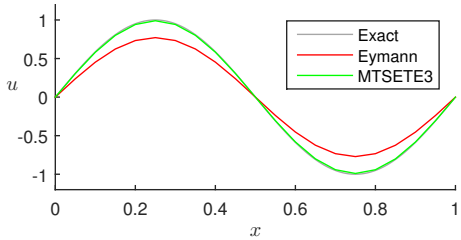
(a)



(b)



(c)



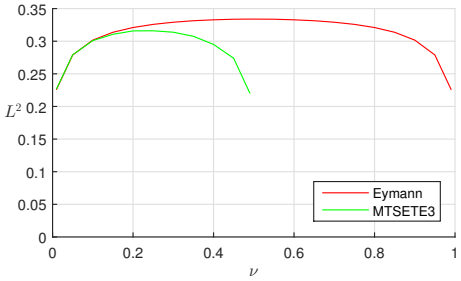
(d)

Figure 4.10: Results of advecting a sine wave on a 20 node grid for 10000 time steps. (a) A plot showing the L^2 error for varying Courant number. The other figures show the actual solution at (b) $\nu = 0.01$, (c) $\nu = 0.25$ and (d) $\nu = 0.49$.

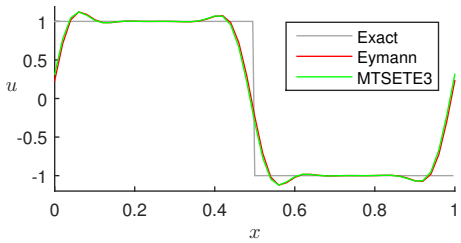
An important observation was made when simultaneously plotting the node and average integral values separately, see Figure 4.12. This figure contains the results of the same simulation as shown in Figure 4.10d, with the exception that u and $\overline{u}$ are plotted separately. According to Figure 4.12b, there is a clear constant spatial disagreement, δ, between both plots, where the plot of u appears to lag behind. After doing multiple simulations with varying Courant number and mesh spacing, δ was found to be consistent with:

$$\begin{aligned} \delta &= \frac{\nu \Delta x}{2} \\ &= \frac{a \Delta t}{2}, \end{aligned} \tag{4.39}$$

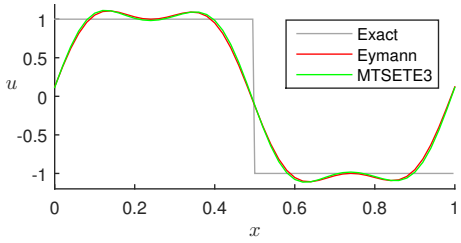
showing that the node values thus lag behind by $\frac{\Delta t}{2}$. This reveals that the node and average integral values are staggered in time, and (4.36) and (4.37) are rewritten to:



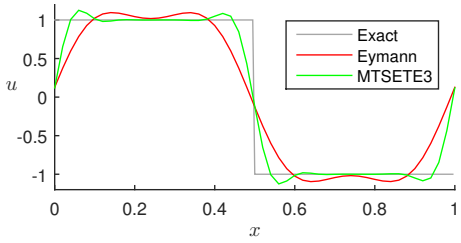
(a)



(b)



(c)



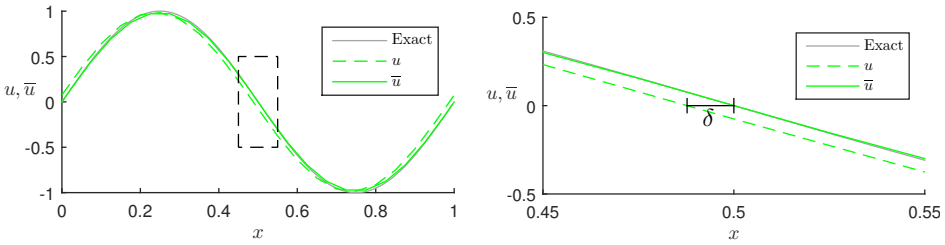
(d)

Figure 4.11: Results of advecting a square wave on a 20 node grid for 10000 time steps. (a) A plot showing the L^2 error for varying Courant number. The other figures show the actual solution at (b) $\nu = 0.01$, (c) $\nu = 0.25$ and (d) $\nu = 0.49$.

$$u_{i+\frac{1}{2}}^{n+\frac{1}{2}} = u_{i+\frac{1}{2}}^{n-\frac{1}{2}} - 2\nu\left(\frac{u_{i-\frac{1}{2}}^{n-\frac{1}{2}} - 3\overline{u}_i^n + 2u_{i+\frac{1}{2}}^{n-\frac{1}{2}}}{\nu+1} + \nu\frac{u_{i-\frac{1}{2}}^{n-\frac{1}{2}} - u_{i+\frac{1}{2}}^{n-\frac{1}{2}}}{\nu+1}\right) \qquad \text{(Node update).} \tag{4.40}$$

$$\overline{u}_i^{n+1} = \overline{u}_i^n - \nu\left(u_{i+\frac{1}{2}}^{n+\frac{1}{2}} - u_{i-\frac{1}{2}}^{n+\frac{1}{2}}\right) \qquad \text{(Integral update).} \tag{4.41}$$

As the figures have shown and is to be expected, the discontinuities that are present in the square wave are troublesome to represent on a discrete grid. The tendency to smoothen these discontinuities causes the solution in the whole domain to significantly deviate from the exact solution. Also, the derivative at the locations of the discontinuities is increasingly underestimated over time. Although the exact solution is truly discontinuous and the derivative is thus infinite, the profile approximates the very steep gradients that occur within a shock wave. The question then arises whether it is realistic to try to represent such high frequency features on a relatively coarse mesh. The spurious behavior occurs due to the



(a) (b)

Figure 4.12: (a) Plots of u and $\overline{u}$ after advecting a sine wave on a 20 node grid for 10000 time steps, with $\nu = 0.49$. (b) Enhanced view of the region in (a), showing a spatial offset, δ.

local profile not being smooth enough relative to the mesh spacing at that specific location. Thus, it is believed that mesh spacing should be dependent on the smoothness of the profile. For this reason, the next chapter will introduce the concept of adaptive mesh refinement (AMR), which is a scientific field that aims at algorithmically conforming local mesh resolution to local smoothness requirements.

5

Adaptive mesh refinement

As the bandwidth of time and/or length scales in numerical simulations grows, uniform mesh structures become increasingly less efficient. As explained in Chapter 3, adaptive mesh refinement (AMR) provides a way of local control over the solution resolution in order to adhere to mesh spacing requirements resulting from these variations in time and/or length scales, while keeping computational effort within bounds. Although AMR is a broad term that encompasses multiple types of adaption, *h*-refinement was found to be most fitting for problems subject to discontinuities. This chapter will aim to provide a detailed overview of the implementation of *h*-refinement for the Active Flux method introduced in Section 4.2. Section 5.1 describes the internal data structures used in the code. Then, the procedure for performing a global time step is elaborated upon in Section 5.2. The adaptive part of AMR requires decision making on which cells to refine or coarsen. These criteria are presented in Section 5.3. In Section 5.4 a method of mapping continuous initial conditions onto the discrete domain is explained. Finally, Section 5.5 provides some ideas on how the method can be extended to multiple dimensions and nonlinear equations.

5.1. Computational architecture

Since for *h*-refinement the positions of the individual nodes are static and child nodes are always contained within their respective parent boundaries, a simple and natural way of representing such a mesh is through some hierarchical structure. Depending on the structure chosen, this poses some constraints to the creation and removal of nodes. Since the problems at hand have just one spatial dimension, a binary tree, see Figure 5.1, is a simple yet effective representation of the computational mesh. In general, a binary tree consists of one root node to which all other nodes are (in)directly attached and each node has between zero and two direct children. The 'distance' a node is separated from the root is referred to as depth. All higher depth nodes attached to a certain node make up the set of nodes labelled as descendants. Similarly, all nodes, including the root node, that provide the connection between a node and the root node are the ascendants. Further following tree terminology, nodes with at least one descendant are called branches, while childless nodes are referred to as leaves.

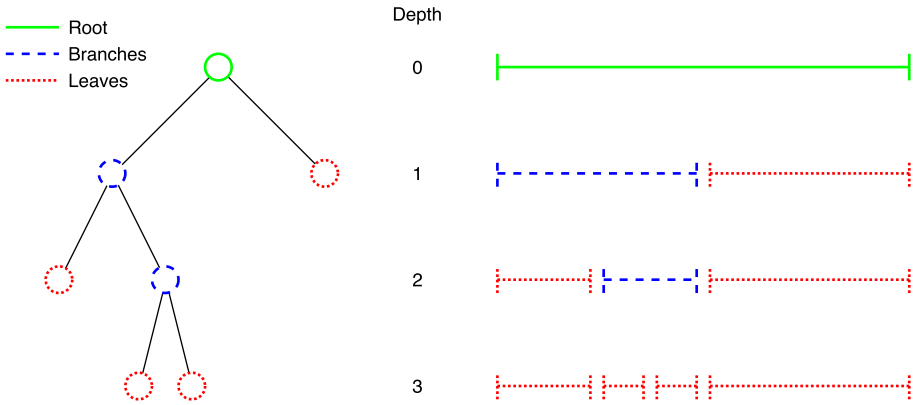


Figure 5.1: Two different binary tree representations. The representation on the right clarifies that the union of each nodes' descendants covers the same domain as the node itself.

In order to handle mesh adaption programmatically, we require a data structure for which one aspect is crucial: Easy access from one node to its surrounding nodes, both horizontally (direct neighbouring nodes) and vertically (parent and child nodes). One such structure that has been adopted by others [10, 18] in mesh refinement codes and is efficient in terms of memory management is called the fully threaded tree. Because we don't want gaps in our mesh, each node in the tree can not have just a single child, but always either zero or two children. It is this restriction on the general binary tree definition that allows the implementation of the fully threaded tree. As explained in [10], time-stepping of computational cells often requires access to neighboring nodes, which is not a trivial task for a regular binary tree. As shown in Figure 5.2, a fully threaded tree overcomes this problem by reusing the empty node references of the leaves as references to their respective neighbours.

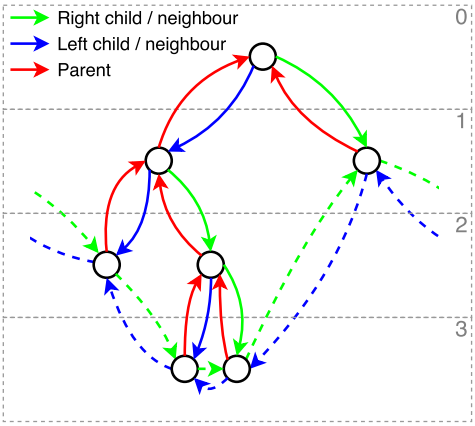


Figure 5.2: Example of a small fully threaded tree, showing how each leaves' empty child node references are reused to simplify access to neighbouring nodes. Top right corner values indicate depth level.

Each node in a fully threaded tree consists of the following properties:

- **Parent** A reference to this node's parent node.
- **Depth** Integer that represents the number of nodes between this node and the root node.
- **Left** Either a reference to the left child node (branches) or the neighbour to the left (leaves).
- **Right** Either a reference to the right child node (branches) or the neighbour to the right (leaves).
- **Data** The actual payload carried by the node.

For an Active Flux method, the data carried by a tree node equals a single computational cell in the mesh. Because the term 'node' is in numerical context used as a container for point quantities, from

here on we let the term 'cell' inherit all properties of a tree node such that we can drop all references to the term 'node' in tree context, avoiding confusion later on.

Although the spatial domain is one-dimensional, the time variable does add another dimension. For a constant advection velocity, the CFL condition poses a constraint on the maximum aspect ratio ($= \Delta t/\Delta x$) of the computational cells. To guarantee stability of the solution method in a global sense for a mesh built from cells of varying size, one could limit the time step size for all cells such that the spatially smallest cell satisfies the CFL condition. A more efficient yet more complex solution is to scale the time step according to the spatial refinement for each cell individually, resulting in a globally constant aspect ratio. Because the root cell spans the entire computational domain and the refinement factor is 2, cell size scales according to:

$$\Delta x_d = \frac{\Delta x_0}{2^d}, \tag{5.1}$$

where d designates cell depth. The constant aspect ratio ensures that local time step size scales equally:

$$\Delta t_d = \frac{\Delta t_0}{2^d}, \tag{5.2}$$

where Δt_0 is referred to as the global time step size. A global time step for the binary tree shown in Figure 5.1 can then be visualized as follows:

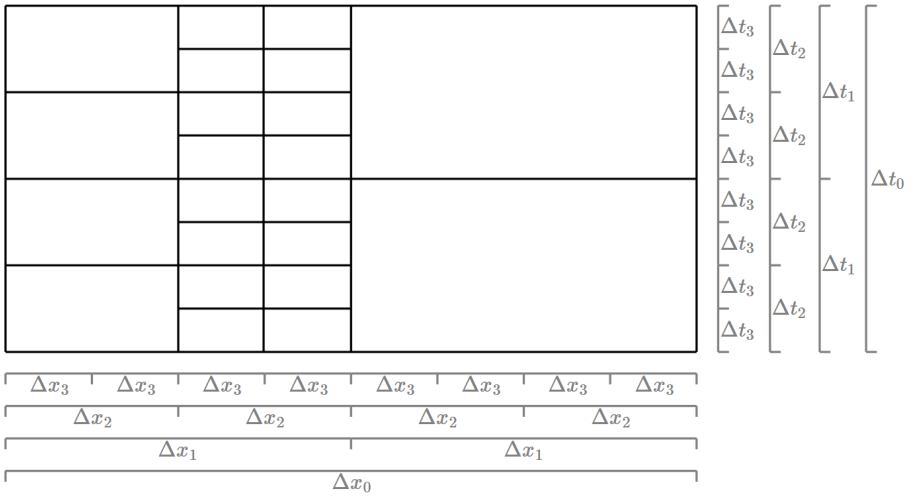


Figure 5.3: Example of a global time step for the tree shown in Figure 5.1.

In order to avoid arithmetic under-/overflow related issues and infinite refinement tendencies near discontinuities, a manually set parameter, d_{max}, is introduced here to provide an upper bound to cell depth. This means that the smallest possible time step size equals:

$$\Delta t_{min} = \frac{\Delta t_0}{2^{d_{max}}}. \tag{5.3}$$

As will become clear later, it is convenient to define a discrete integer time scale as:

$$\hat{t} = \left\lfloor \frac{t}{\Delta t_{min}} \right\rfloor \mod 2^{d_{max}}. \tag{5.4}$$

In Chapter 4 it was found that the nodes and volumes of the cells are staggered both spatially and temporally. This is visualized in Figure 5.4, replacing the cells in Figure 5.3 with the positions of the nodes and volumes. Assuming a maximum depth of 3, alongside the discrete time scale is shown.

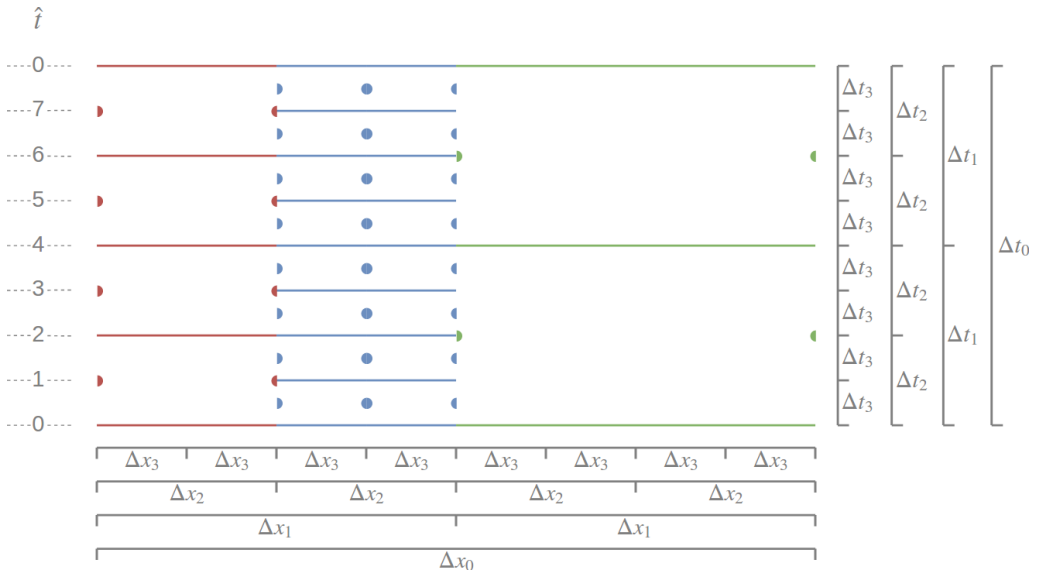


Figure 5.4: A visualization of the locations of node values and integral values. The colors are used to indicate distinctive depths and make clear which cell the nodes correspond to.

This figure reveals that the nodes on the shared interface between neighbouring cells of equal depth refer to the same value. Also observe that nodes of unequal depth cells never share the same time levels. In order to minimize the time level differences between neighbouring cells and simultaneously create a smooth cell depth graduation, the depth difference between two neighbouring cells is limited to at most one and the example mesh that has been used in Figures 5.1-5.4 is therefore invalid.

To account for the spatial and temporal nonuniformity of the mesh, several extra variables are introduced in a cell. To be able to concisely refer to these variables, the naming convention is expanded here. First, in order to adhere to the conservation constraint at interfaces separating cells of varying depth, the intermediate fluxes for the smaller cell need to be kept track of. This ensures that, when the larger cell completes its local time step, the sum of fluxes on both sides of the interface are equal. Secondly, refining/coarsening criteria are often based on the smoothness of the local solution. However, because within a cell the nodes and volumes are never aligned time wise, interpolation is required to determine the actual solution at a specific discrete time level. These two requirements result in the following layout of variables within a cell:

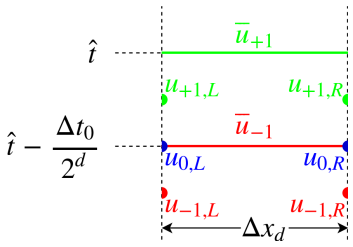


Figure 5.5: Distribution of variables within a cell.

As before, volume quantities are recognized by their overline. A subscript is added to distinguish the new value (+1) from the old value (−1). An extra subscript has been added to node quantities to indicate whether they lie on the left (L) or right (R) interface. Also, the subscript 0 is reserved for the interpolated volume-aligned nodes.

5.2. Global time step procedure

This section elaborates on the procedure that is performed to advance the solution a single global time step. Because the logical flow of the whole procedure is quite complex due to nested loops and the inevitable amount of bookkeeping, it is split up in several smaller routines, which will be discussed in order of appearance.

By definition, a global time step is equal to the local time step of the root cell. With each increasing depth level, the amount of steps in a global time step doubles, such that a cell at a maximum depth d_{max} completes $2^{d_{max}}$ steps every global time step. This forms the iteration for the main loop, shown in the flowchart in Figure 5.7. Since not every node or volume is advanced at each iteration, we need a way to determine which quantities require re-evaluation during a certain iteration. As can be observed from Figure 5.4, every smallest time step the nodes and volumes of the maximum depth cells are advanced. More generally, the volume of a cell of depth d is advanced every $2^{d_{max}-d}$ time steps. Furthermore, a cell's node values are advanced at the same rate, but at intermediate time levels. At this point another variable d_{lim} is introduced which provides the lower limit on the depth range of cells for which the volumes are advanced. Clearly, this variable is dependent on the discrete time level $\hat{t}$. If one manually determines d_{lim} for several values of $\hat{t}$, the result is a specific non-repeating sequence. Subtracted from the maximum depth d_{max}, one finds a sequence of the form (0,1,0,2,0,1,0,3,0,... etc.). In computer science fields the algorithm that produces this sequence goes under the name of 'count trailing zeros' (CTZ). As the name implies, given an input integer in binary form, this function returns the amount of trailing zeros after the least significant set bit. Thus, d_{lim} is defined as:

$$d_{lim} = d_{max} - CTZ\left(\hat{t}\right). \tag{5.5}$$

In Figure 5.6 an attempt is made at visually clarifying the use of this algorithm. From this figure it can also be deducted that at a certain $\hat{t}$, the nodes of cells of depth $d_{lim}-1$ are updated (given that $d_{lim} > 0$).

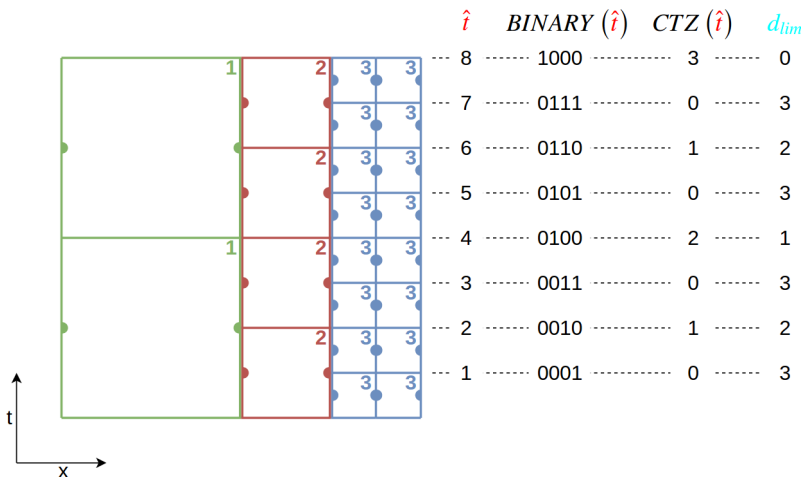


Figure 5.6: Example visual representation of the relation between the CTZ function and depth range. Upper right corner values indicate cell depth. Maximum depth equals 3.

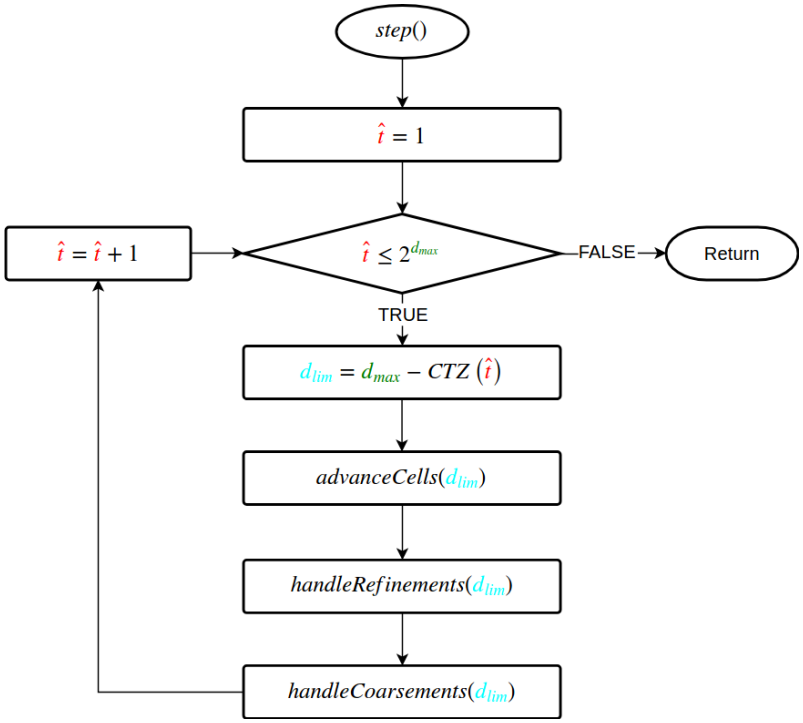


Figure 5.7: Flowchart of the main loop of a single global time step.

The main loop contains three functions, which are executed in succession:

- *advanceCells()* Responsible for advancing cell nodes and volumes.
- *handleRefinements()* Checks cell state and refines if necessary.
- *handleCoarsements()* Checks cell state and coarsens if possible.

In Figure 5.8 the function that advances the nodes and volumes is expanded. This shows that first the nodes of the maximum depth cells are advanced. Then the volumes of the cells whose depth is $\geq d_{lim}$ are advanced. Finally, if possible, the nodes of cells of depth $d_{lim} - 1$ are advanced.

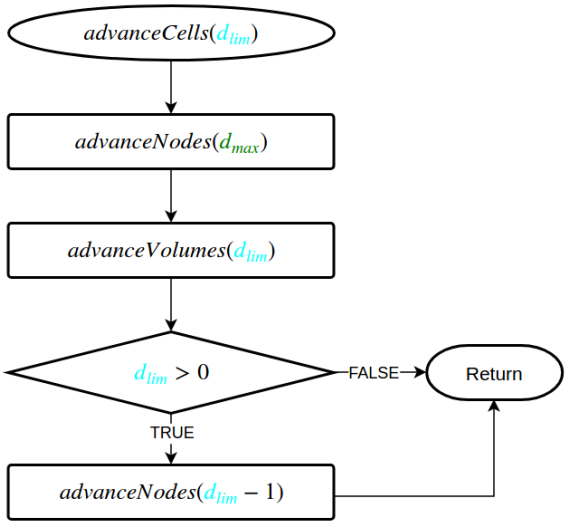


Figure 5.8: Flowchart of the procedure that handles advancing the nodes and volumes.

Figure 5.9 displays the procedure of advancing the nodes of all cells at a depth d. These cells are iterated over and their left node value is calculated. Then it is checked whether the neighbouring cell on

the right is of equal depth. If so, the nodes of both cells refer to the same value, meaning that this node will be advanced when the procedure is performed for the neighbouring cell and there is thus no need to do that here.

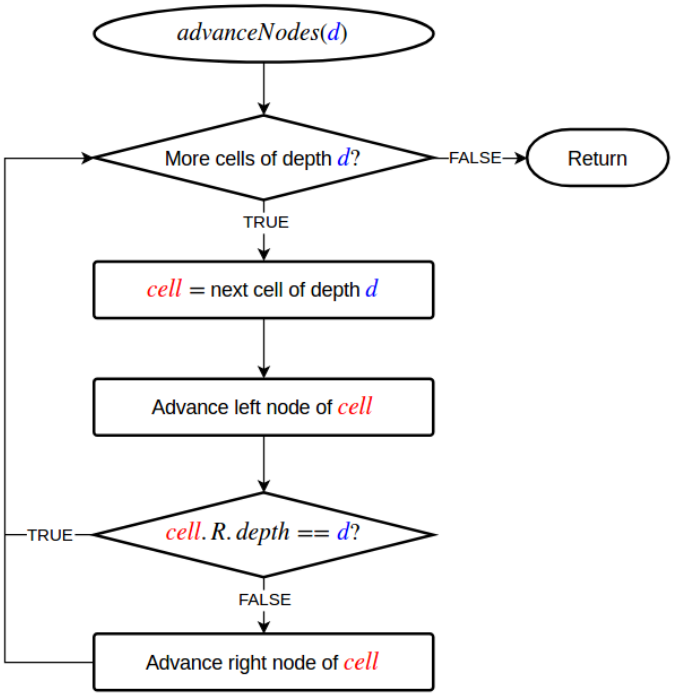


Figure 5.9: Flowchart of the procedure that advances the nodes.

The procedure for advancing the cell volumes is shown in Figure 5.10, displaying a simple iteration over all cells whose depth is in the interval $[d_{lim}, d_{max}]$.

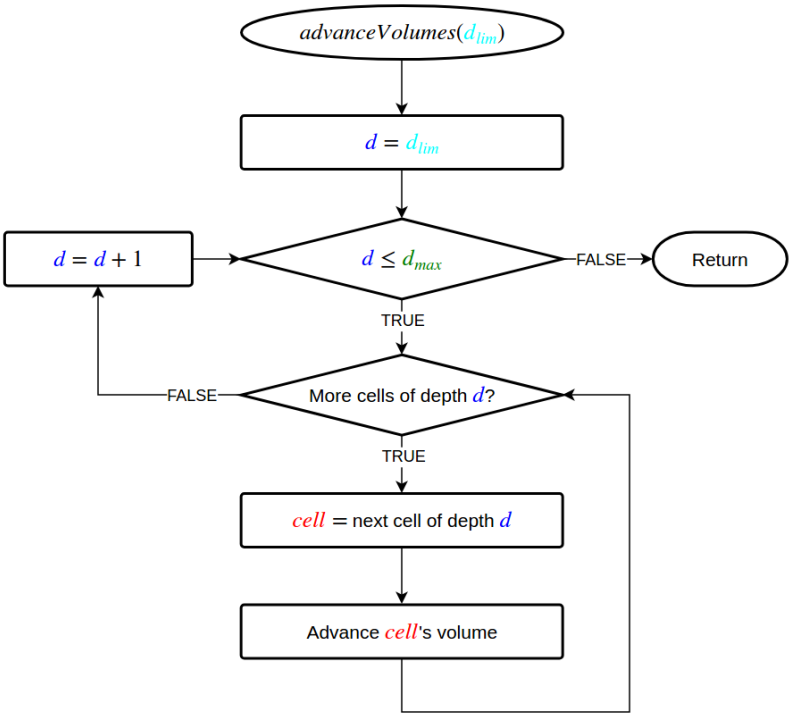


Figure 5.10: Flowchart of the procedure that advances the volumes.

The final part in the main loop as shown in Figure 5.7 is where the adaptive part of the method takes place. Here, for each eligible cell a decision is made whether to alter its structure through refinement or coarsening.

5.2.1. Refinement

The refinement step involves splitting a cell into two smaller sized cells, thereby essentially doubling the degrees of freedom in this domain, which then hopefully provide a more accurate display of the solution than the original cell. Here, the refinement step is explained in detail.

Due to the fact that earlier we posed the constraint that two neighbouring cells' depths may differ by at most one, the validity of this condition has to be checked for when refining a cell. However, because refinement is crucial in preserving and accurately propagating the solution, it is desirable to be able to split a cell when the algorithm determines it needs to, without the possibility of failing due to this constraint. This means that before a cell is refined, the algorithm will iteratively traverse the neighbours in a specific direction to check if these cells need to split in advance. The iteration direction depends on the original cell's position in the binary tree. An example of this refinement step is shown in Figure 5.11.

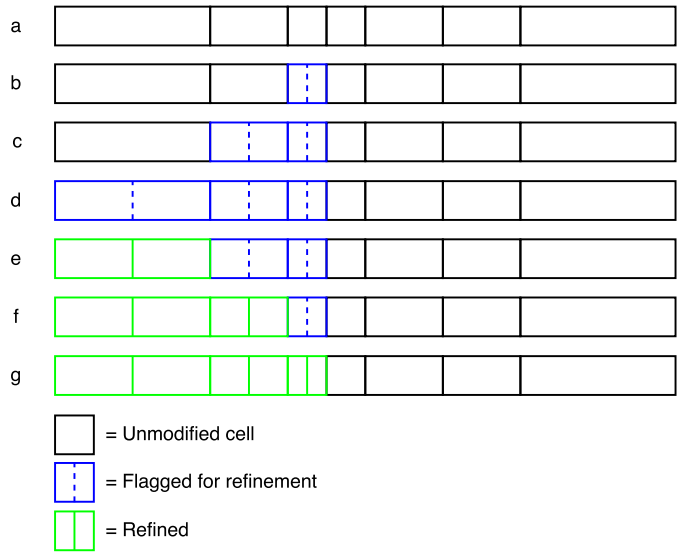


Figure 5.11: Visual representation of the steps taken during iterative splitting of a cell.

When splitting a cell, some nodes and/or volumes of the newly created cells are not aligned with the original cell node/volume locations. The solution values at these locations are found by interpolation. Which properties can be extracted from neighbours and which have to be interpolated depends on the neighbours' relative depth levels and the discrete time level. Due to the previous mesh validation step, the neighbours of a to be refined cell are either on the same depth level as this cell, or one level above it. All this means that upon refining, a cell and its neighbours are in one of 16 possible states. A refinement step for one of these states is shown in Figure 5.12.

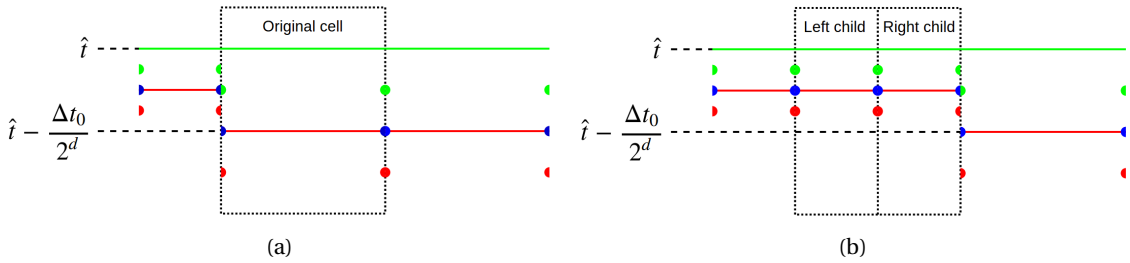


Figure 5.12: One out of a possible 16 states when refining a cell. (a) Before refinement. (b) After refinement.

For sake of clarity, for just this state we will shortly discuss the steps that are taken to transition from Figure 5.12a to Figure 5.12b. To refer to the variables in a clear and concise way, a superscript is added

to indicate the cell. The cells that will be referred to are the left neighbour $\left(^{L}\right)$, right neighbour $\left(^{R}\right)$, left child cell $\left(^{LC}\right)$, right child cell $\left(^{RC}\right)$ and the original cell $\left(^{O}\right)$. Right after the two child cells are created, the state is as shown in Figure 5.13:

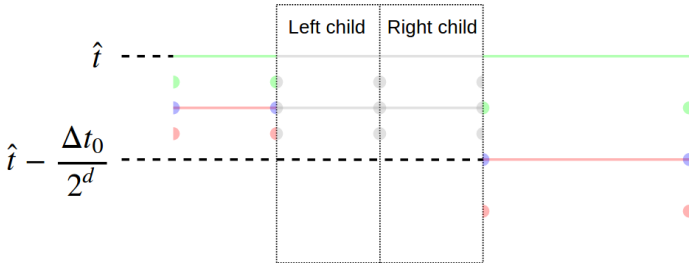


Figure 5.13: The initial cell state immediately after splitting a cell.

The three newly created node values on the left interface $\left(u^{LC}_{-1,L}, u^{LC}_{0,L}, u^{LC}_{+1,L}\right)$ can be retrieved from the neighbouring cell on the left. Similarly, $u^{RC}_{0,R}$ is set equal to $u^{R}_{+1,L}$:

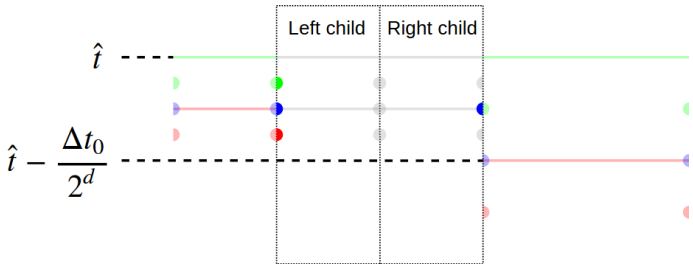


Figure 5.14: Step 1 of the reconstruction of variables within the child cells.

Now, $u^{RC}_{+1,R}$ is calculated by advancing the original cell by a quarter of its normal time step:

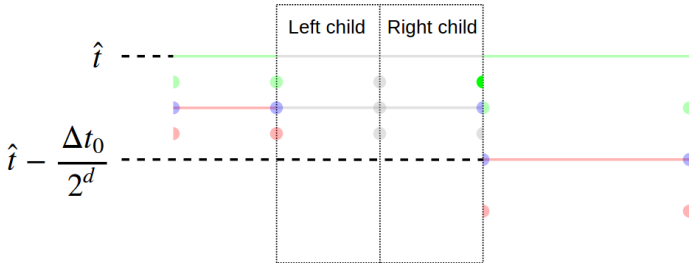


Figure 5.15: Step 2 of the reconstruction of variables within the child cells.

Now that both $u^{LC}_{+1,L}$ and $u^{RC}_{+1,R}$ are known, we can construct flux terms on these interfaces and use them in conjunction with $\overline{u}^{O}_{+1}$ to find the sum of $\overline{u}^{LC}_{-1}$ and $\overline{u}^{RC}_{-1}$:

$$\left(\overline{u}^{LC}_{-1} + \overline{u}^{RC}_{-1}\right) = \overline{u}^{O}_{+1} + \frac{A}{2}\left(f\left(u^{RC}_{+1,R}\right) - f\left(u^{LC}_{+1,L}\right)\right), \tag{5.6}$$

where A designates a cell's aspect ratio $(= \Delta t/\Delta x)$. Since the nodes enclosing the union of these volumes, $u^{LC}_{0,L}$ and $u^{RC}_{0,R}$, are known, it is now possible to make a quadratic reconstruction. Based on this reconstruction, the volume sum $\left(\overline{u}^{LC}_{-1} + \overline{u}^{RC}_{-1}\right)$ can be separated into individual components, by means of integration.

$$\overline{u}^{LC}_{-1} = \left(\overline{u}^{LC}_{-1} + \overline{u}^{RC}_{-1}\right) + \frac{u^{LC}_{0,L} - u^{RC}_{0,R}}{4}, \tag{5.7}$$

$$\overline{u}^{RC}_{-1} = \left(\overline{u}^{LC}_{-1} + \overline{u}^{RC}_{-1}\right) - \frac{u^{LC}_{0,L} - u^{RC}_{0,R}}{4}. \tag{5.8}$$

$u^{LC}_{0,R}\left(= u^{RC}_{0,L}\right)$ is determined by the reconstruction as well:

$$u^{LC}_{0,R} = \frac{-u^{LC}_{0,L} + 6\left(\overline{u}^{LC}_{-1} + \overline{u}^{RC}_{-1}\right) - u^{RC}_{0,R}}{4}. \tag{5.9}$$

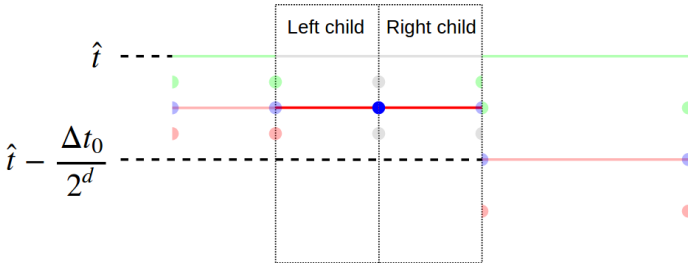


Figure 5.16: Step 3 of the reconstruction of variables within the child cells.

$u^{LC}_{+1,R}$ is found in a similar way, using the average of the surrounding volumes to again create a quadratic reconstruction:

$$u^{LC}_{+1,R} = \frac{-u^{LC}_{+1,L} + 6\frac{\left(\overline{u}^{LC}_{-1} + \overline{u}^{RC}_{-1}\right) + \overline{u}^{O}_{+1}}{2} - u^{RC}_{+1,R}}{4}. \tag{5.10}$$

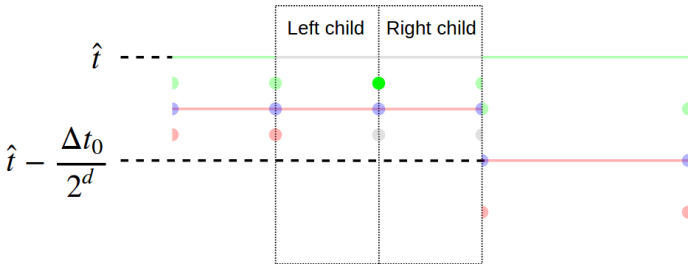


Figure 5.17: Step 4 of the reconstruction of variables within the child cells.

The flux term constructed using the previously calculated node value $u^{LC}_{+1,R}$ is used in conjunction with the previously calculated flux and volume terms to find the individual green volume values of the child cells as:

$\overline{u}^{LC}_{+1}$ and $\overline{u}^{RC}_{+1}$ can now be found using conservation:

$$\overline{u}^{LC}_{+1} = \overline{u}^{LC}_{-1} - A\left(f\left(u^{LC}_{+1,R}\right) - f\left(u^{LC}_{+1,L}\right)\right), \tag{5.11}$$

$$\overline{u}^{RC}_{+1} = \overline{u}^{RC}_{-1} - A\left(f\left(u^{RC}_{+1,R}\right) - f\left(u^{RC}_{+1,L}\right)\right). \tag{5.12}$$

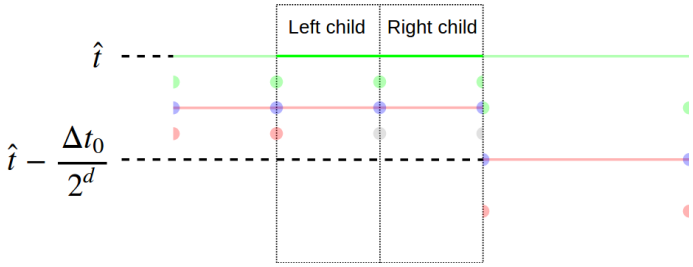


Figure 5.18: Step 5 of the reconstruction of variables within the child cells.

$u^{RC}_{-1,R}$ is determined by a flux equality constraint:

$$f\left(u^{RC}_{-1,R}\right) = 2f\left(u^{R}_{+1,L}\right) - f\left(u^{RC}_{+1,R}\right). \tag{5.13}$$

As the final step, $u^{LC}_{-1,R}$ can be interpolated using again a quadratic reconstruction:

$$u^{LC}_{-1,R} = \frac{-u^{LC}_{-1,L} + 6\frac{\left(\overline{u}^{LC}_{-1} + \overline{u}^{RC}_{-1}\right) + \overline{u}^{O}_{-1}}{2} - u^{RC}_{-1,R}}{4}. \tag{5.14}$$

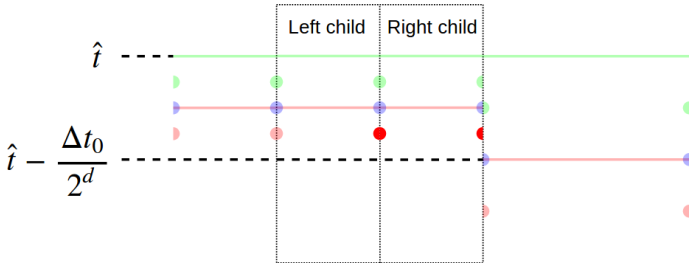


Figure 5.19: The final step of the reconstruction of variables within the child cells.

5.2.2. Coarsening

The coarsening procedure takes place after refinement, and is done to decrease the amount of cells in the mesh by determining if the local solution in two child cells is smooth enough to be represented by a single cell, thereby reducing its computational burden. The first step in doing this is to define if two child cells are actually allowed to merge. In the refinement step, a cell's neighbours can be forced to split if this is necessary to retain the validity of the mesh. Contrarily, the coarsening procedure may not iteratively force neighbours to merge, as these neighbours may concern cells whose increased resolution is required for an accurate solution representation. Thus, some constraints have to be checked for before a cell can even be considered to be merged.

Due to the use of the binary tree, the mesh always consists of 'left' and 'right' cells (excluding the root cell), see Figure 5.2. Because only cells with two child leaves are eligible for merging, only the parents of either 'left' or 'right' cells in the mesh have to be traversed in order to cover all coarsening possibilities. Then, it should be checked whether the parent cell has any grandchildren. If it does, apparently the cell depth is still needed and the cell cannot be merged. Finally, to sustain the validity of the mesh, the depths of neighbouring cells have to be checked for. To visualize these steps, each cell in the mesh traverses the flowchart as shown in Figure 5.20. Also, the cells are traversed in order of increasing depth level, so as to avoid multiple merges on a single tree path in one coarsening step.

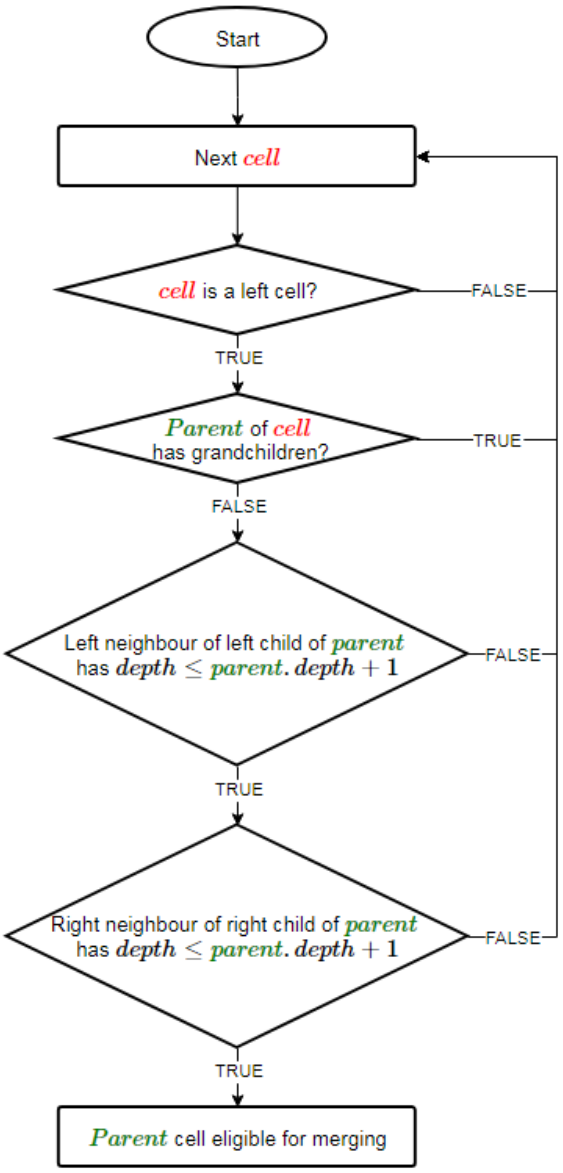


Figure 5.20: The procedure to determine if two child cells can be merged into their parent cell.

Because cell merges are only performed after a volume update, there are 8 unique cell states that have to be considered opposed to the 16 states found when refining. The coarsening procedure for one of these states, which in fact is the same state shown in Figure 5.12 but in the opposite direction, is explained here. Because in this case the total degrees of freedom within the domain of the target cell are reduced, this procedure involves less interpolation steps. The same sub- and superscripts are the same as used in the previous section, except that the 'original' cell has become the 'target' cell, and thus the superscript ^{O} is replaced by ^{T}.

In Figure 5.21 the original and target state of the coarsening step are shown.

After doing the structural cell changes, the situation is as shown in Figure 5.22.

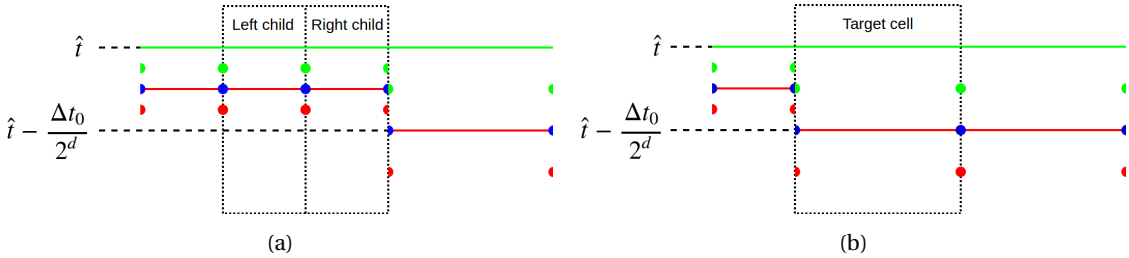


Figure 5.21: One out of a possible 8 states when coarsening a cell. (a) Before coarsening. (b) After coarsening.

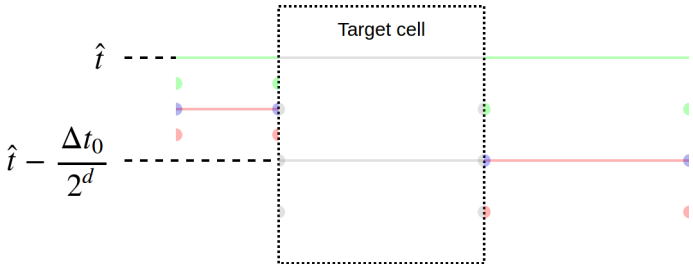


Figure 5.22: The initial cell state immediately after merging two cells into their parent.

As was the case for the refinement step, several values can be retrieved from the neighbours. As the neighbour to the right is of the same depth level, the node values on this interface $\left(u^T_{-1,R}, u^T_{0,R}, u^T_{+1,R}\right)$ can be directly retrieved. On the left interface, $u^T_{+1,L}$ is set equal to $u^L_{0,R}$.

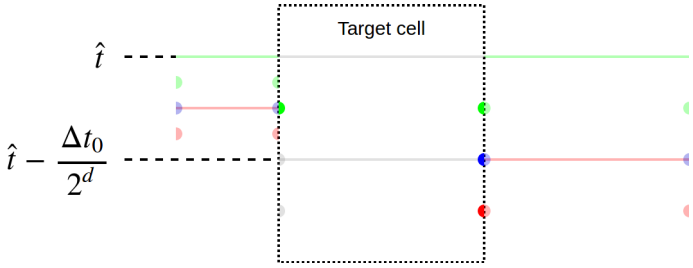


Figure 5.23: Step 1 of the reconstruction of variables within the target cell.

The average volume value $\overline{u}^T_{+1}$ is found from the child cells as follows:

$$\overline{u}^T_{+1} = \frac{\overline{u}^{LC}_{+1} + \overline{u}^{RC}_{+1}}{2}. \tag{5.15}$$

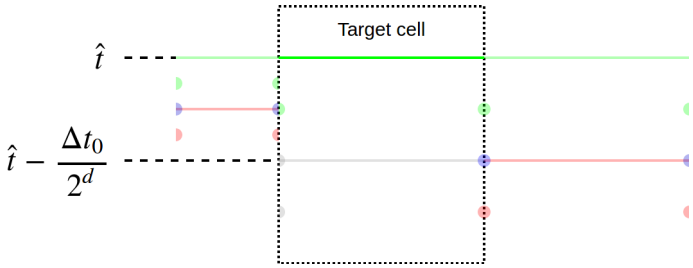


Figure 5.24: Step 2 of the reconstruction of variables within the target cell.

Then, the conservation requirement is used to calculate the previous average volume value $\overline{u}^T_{-1}$:

$$\overline{u}^T_{-1} = \overline{u}^T_{+1} + \frac{A}{2}\left(f\left(u^T_{+1,R}\right) - f\left(u^T_{+1,L}\right)\right). \tag{5.16}$$

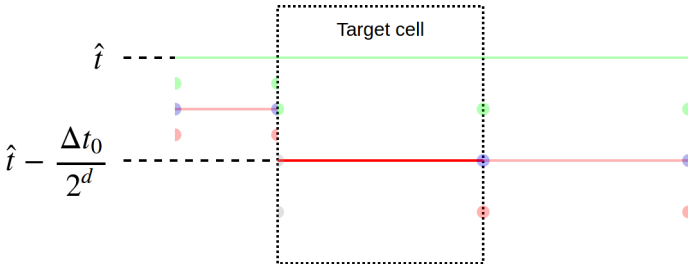


Figure 5.25: Step 3 of the reconstruction of variables within the target cell.

At this point only two node values at previous time levels remain. These are calculated by simple first order extrapolation:

$$u^T_{0,L} = 2u^L_{0,R} - u^T_{+1,L}. \tag{5.17}$$

$$u^T_{-1,L} = 4u^L_{0,R} - 3u^T_{+1,L}. \tag{5.18}$$

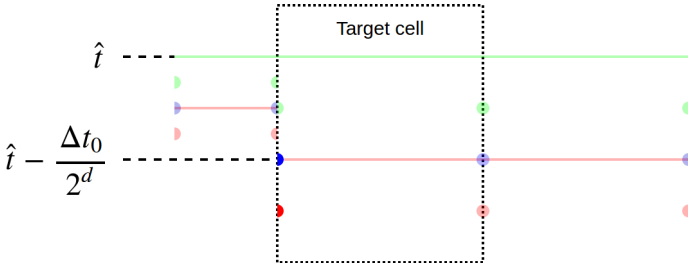


Figure 5.26: The final step of the reconstruction of variables within the target cell.

5.3. Adaption criteria

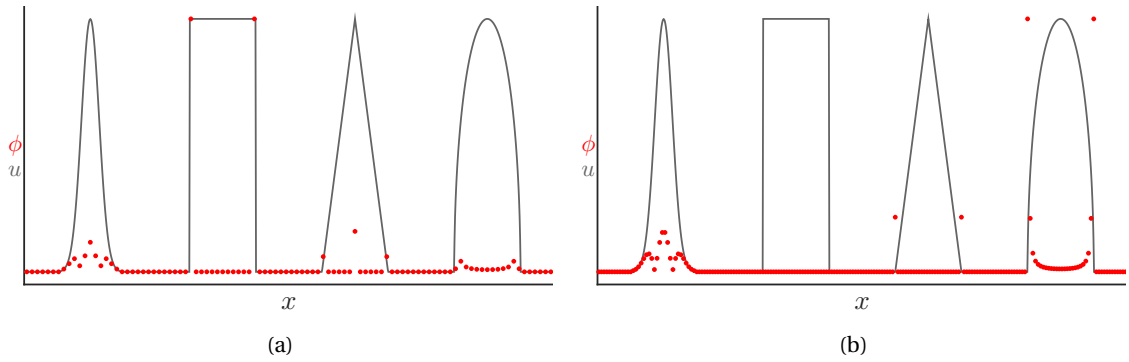
The most difficult aspect of AMR is the adaptive part, which requires implementing automated decision making on when and where the mesh should be refined or coarsened. This decision making is based on some indication of the local solution error or complexity. In Section 3.4 it was mentioned that Richardson extrapolation is a popular choice of error estimator in patch-based codes, but not feasible for codes where cells are treated individually due to boundary conditions. Since the latter is the case for the adaption method described in this chapter, it was decided to implement a feature detector as error estimator. In order to find a suitable adaption indicator for the linear advection equation, it is necessary to first identify the features that cause relatively rapid degradation of the solution accuracy. These features are problem dependent and the indicators therefore as well. For linear advection, the accuracy of representing a continuous signal on a discrete domain is related to signal smoothness and the sample interval in the discrete domain. The dissipation that is present in nearly all numerical schemes to guarantee stability, also has a smoothening effect that is particularly well observable near discontinuities, kinks (discontinuities in the derivative) and rapidly varying signals. In order to define an indicator that has a strong response in the vicinity of these features, Zalesak's waveform will be used as a test case. This waveform consists of a Gaussian distribution, a square wave, a triangle wave and an elliptic shape. These four shapes all have different characteristics and are therefore a good test case to validate the feature indicator. Three different indicators will be explored here.

5.3.1. Indicator #1

The first indicator that was tested is a simple indicator based on the local curvature. The idea is that large second derivatives occur near discontinuities as well as other rapid signal changes, which will thus be detected. For the quadratic reconstruction within a cell, the second order derivative equals:

$$\phi = \frac{\partial^2 u}{\partial x^2} = \frac{6u_L - 12\overline{u} + 6u_R}{\Delta x^2}, \tag{5.19}$$

where ϕ designates the indicator value. Applying this to Zalesak's waveform while normalizing the result is shown in Figure 5.27. These figures show that the indicator is quite sensitive to cell placement. When 100 cells are used, the discontinuity clearly shows a peak, while for 200 cells the integrated value at the discontinuity is approximately halfway it's surrounding node values, resulting in a very low curvature value. The sharp peak of the triangle wave is also no longer detected after doubling the cell count.



(a) (b)

Figure 5.27: Feature indicator based on curvature. (a) 100 cells. (b) 200 cells.

5.3.2. Indicator #2

The first indicator suggests that use of the values of one cell is not sufficient to detect a discontinuity in all cases. Therefore this next indicator makes use of information of neighbouring cells. For this indicator the idea is that near discontinuities the difference in linear slopes between a cell and its neighbours has a large value for at least one neighbour. This idea is depicted in Figure 5.28.

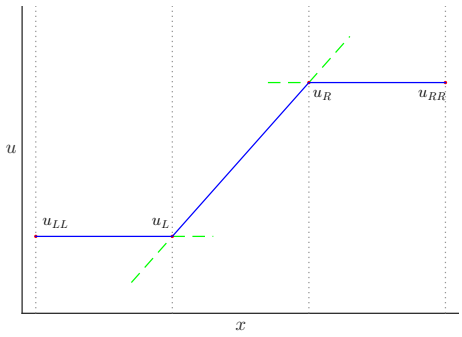
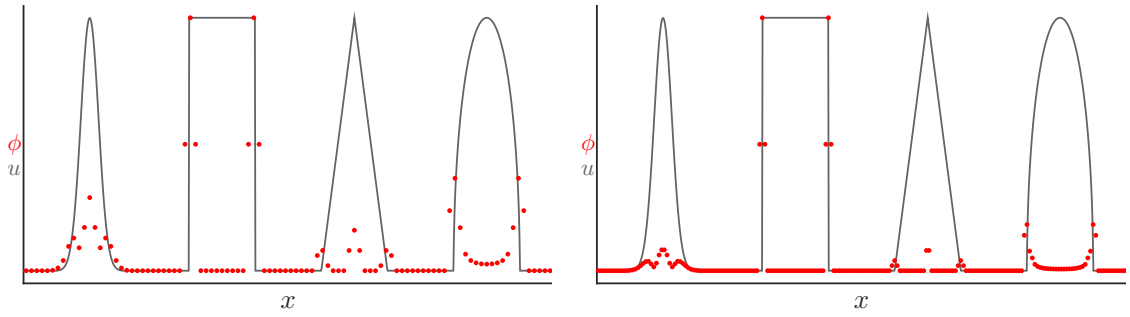


Figure 5.28: Visualization of the rationale behind the linear slope indicator. Blue solid lines indicate the signal, the red dots the node values, and the green dashed lines represent the slopes.

The value of the indicator is defined as:

$$\begin{aligned} \phi &= abs\left(\frac{\partial u}{\partial x} - \left.\frac{\partial u}{\partial x}\right|_L\right) + abs\left(\left.\frac{\partial u}{\partial x}\right|_R - \frac{\partial u}{\partial x}\right) \\ &= abs\left(\frac{u_R - u_L}{\Delta x} - \frac{u_L - u_{LL}}{\Delta x_L}\right) + abs\left(\frac{u_{RR} - u_R}{\Delta x_R} - \frac{u_R - u_L}{\Delta x}\right). \end{aligned} \tag{5.20}$$

When again applied to Zalesak's waveform, the indicator detects the discontinuity for both cases, see Figure 5.29. Although smaller, there is also a response for sharp corner values, which are apparent in the triangle and elliptic shape.



(a) (b)

Figure 5.29: Feature indicator based on the sum of the absolute differences in slopes (of the linear reconstruction) between neighbouring cells. (a) 100 cells. (b) 200 cells.

5.3.3. Indicator #3

For the last indicator that is tested, the same approach as indicator #2 is taken, except instead of the linear reconstruction, we now use the quadratic reconstruction to approximate the slopes at the cell interfaces, see Figure 5.30.

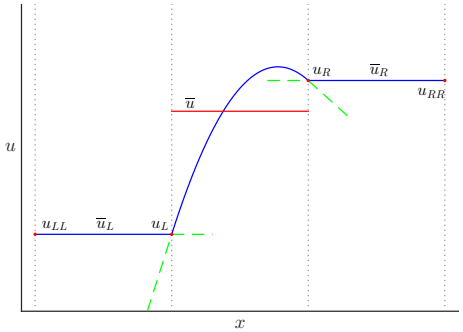


Figure 5.30: Visualization of the rationale behind the quadratic slope indicator. Blue solid lines indicate the signal, the red dots the node values, the red solid line the cell's integral value, and the green dashed lines represent the slopes.

For this indicator, the value becomes:

$$\begin{aligned}\phi &= abs\left(\frac{\partial u}{\partial x} - \frac{\partial u}{\partial x}\bigg|_L\right) + abs\left(\frac{\partial u}{\partial x}\bigg|_R - \frac{\partial u}{\partial x}\right) \\ &= abs\left(\frac{-4u_L + 6\overline{u} - 2u_R}{\Delta x} - \frac{2u_{LL} - 6\overline{u}_L + 4u_L}{\Delta x_L}\right) + abs\left(\frac{-4u_R + 6\overline{u}_R - 2u_{RR}}{\Delta x_R} - \frac{2u_L - 6\overline{u} + 4u_R}{\Delta x}\right).\end{aligned} \tag{5.21}$$

As seen in Figure 5.31, this indicator also responds well to the discontinuity. However, the response to the Gaussian function is much smaller, which is desirable as this shape is quite smooth. Also, the response for sharp corners seems to be more concentrated compared to indicator #2.

5.3.4. Depth mapping

After choosing a feature indicator, the last step is to map the indicator values to a corresponding refinement level. This involves a manually set parameter Φ that determines how aggressive the adaption is and therefore influences the average amount of cells in a simulation and thereby its accuracy. The interval $[0, \Phi]$ is linearly binned to accommodate all possible depth levels, see Figure 5.32. When the value of the indicator for a specific cell is above the upper limit of this cells' depth bin, the cell is refined.

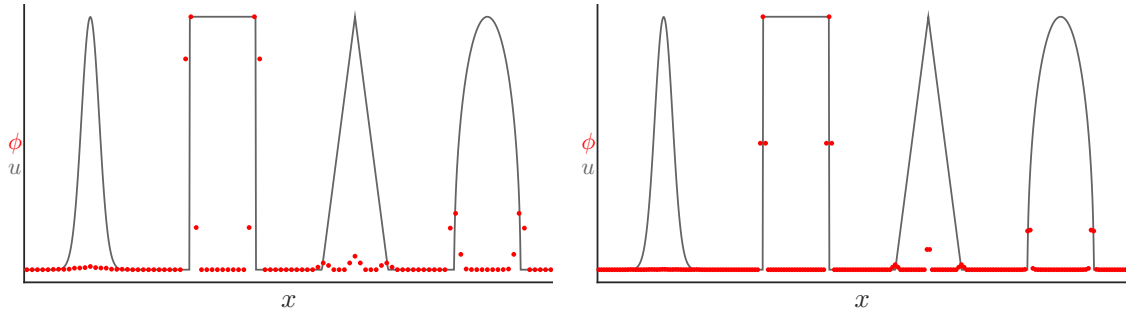


Figure 5.31: Feature indicator based on the sum of the absolute differences in slopes (of the quadratic reconstruction) between neighbouring cells. (a) 100 cells. (b) 200 cells.

Coarsening is done similarly, but for this situation a parent cell is merged only when the indicator values of both child cells drop below the lower limit. Merging is still only possible when the requirements as explained in section 5.2.2 are met.

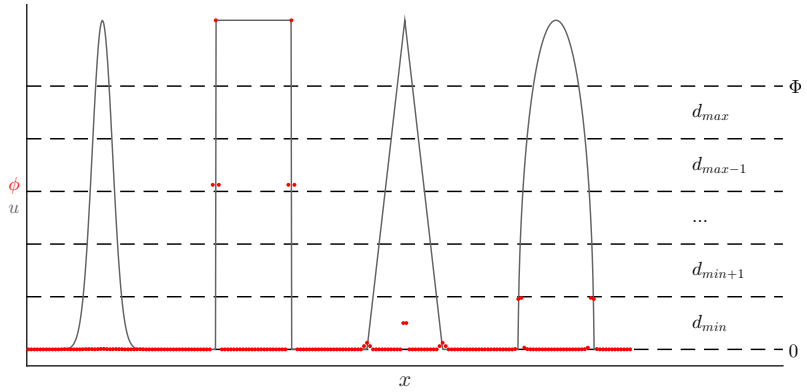


Figure 5.32: A plot showing how the indicator value ϕ is mapped to a required depth level through the introduction of the parameter Φ.

Of course, instead of a linear division, where each depth bin is equally spaced, more advanced non-uniform divisions can be defined, but this introduces more degrees of freedom and is deemed out of scope for this thesis.

5.4. Mesh initialization

An accurate simulation starts with an accurate reflection of the initial conditions on the discrete domain. A poor translation of the continuous initial conditions to a discrete mesh leads to a large discretization error. Mesh creation can be done in multiple ways, including visual inspection. However, this section focuses on automatic generation of the mesh.

Since mesh initialization is a step to be performed just once per simulation and will therefore likely not be dominant in efficiency issues, accuracy and simplicity of translating the initial conditions to a mesh are in general prioritized over efficiency. The only manual step in this process is to determine the maximum depth. Performing a Fourier transform on the input signal provides a good indication of the maximum level of refinement required to capture the highest frequencies. Of course, for kinks and discontinuities the required depth goes to infinity, and the optimum depth level will then need to be derived from the user's requirements. Once the maximum depth is determined, the following steps are taken:

1. Refine mesh uniformly to the maximum depth d_{max}.

2. Use initial conditions to set the node and volume values for each cell in the mesh.

3. Starting at the maximum depth, iteratively traverse the mesh and coarsen cells where possible, solely using the node and integral values.

This process can be executed in batches by segmentizing the mesh in advance. This allows for easy parallelization to decrease time consumption, or sequential processing to decrease memory overhead.

Once the mesh has been defined in terms of required refinement levels, a final step has to be performed before the simulation can start. The nodes and volumes are temporally staggered, but the initial conditions are often provided at a single moment in time. One method of converting the initial conditions to a useful input state for the MTSETE3 scheme is to derive the node values at time level $-\Delta t_d/2$ from the initial conditions by means of characteristic tracing, see Figure 5.33. Although there are other ways of obtaining the staggered initial conditions, this step has been deemed of minor importance here and was thus not investigated further.

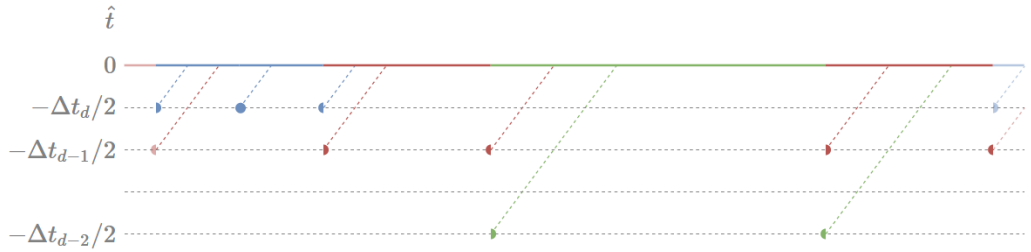


Figure 5.33: This figure shows how tracing of characteristics is used to derive the node values from the continuous initial conditions at time levels < 0, which is necessary due to temporal staggering.

5.5. Further considerations

Since the whole procedure of implementing mesh adaption for linear advection in one dimension has been described now, this chapter is concluded here by some preliminary prospects for extension to higher spatial dimensions and also for Burgers' equation and, very briefly, the Euler equations.

5.5.1. Higher order extension

One aspect that separates a one-dimensional advection problem from its higher dimensional analogue is the fact that the characteristic velocity becomes a vector in higher dimensions. Thus, in addition to a normal component, the velocity has a shear component. Since a shear velocity does not contribute to an interface flux, higher dimensional problems are commonly reduced to a set of one-dimensional problems. Figure 5.34 provides a visualization of how this approach could be used in case of the MTSETE3 scheme. In this figure, the green plane represents the average integral value, which is depicted as a line in the one-dimensional problem. The dots indicate the node values, where the blue and red colors provide a visual distinction between the nodes used for the two one-dimensional problems in x- and y-direction respectively. The locations of the nodes at elevated time level reveal that this situation corresponds to the case where both components of the (constant) characteristic velocity $\boldsymbol{a} = (a_x, a_y)$ are positive. Of course, in case of non-orthogonal interfaces, or for example triangular cells, the situation becomes more complicated.

Concerning refinement depth inequalities of neighbouring cells, the two-dimensional situation is also somewhat different. Assuming quadrilateral cell geometries and a refinement factor of two in both x- and y-direction, upon refinement four cells will be created from a parent cell. This means that, in order to use the fully threaded tree, each cell has four references to either neighbouring or descending cells, depending on whether the cell is a leaf or a branch. However, as Figure 5.35 suggests, a single cell may now have up to eight neighbours. One way to resolve this incompatibility is to still refer to one neighbour per interface and in addition set a convention on reference order, such that a reference to the disconnected neighbour (C_3) can be obtained through the connected neighbour (C_2), see Figure 5.35b.

In addition, in higher dimensions the refinement and coarsening procedures themselves are more complex and there are more unique cases that are to be considered. Another difference is that in one dimension, the nodes sharing the interface between two cells of unequal refinement levels were only staggered temporally, and could thus be updated by halving the time step of the larger cell. However, in two dimensions such nodes will still be staggered temporally, but also in the spatial direction parallel to

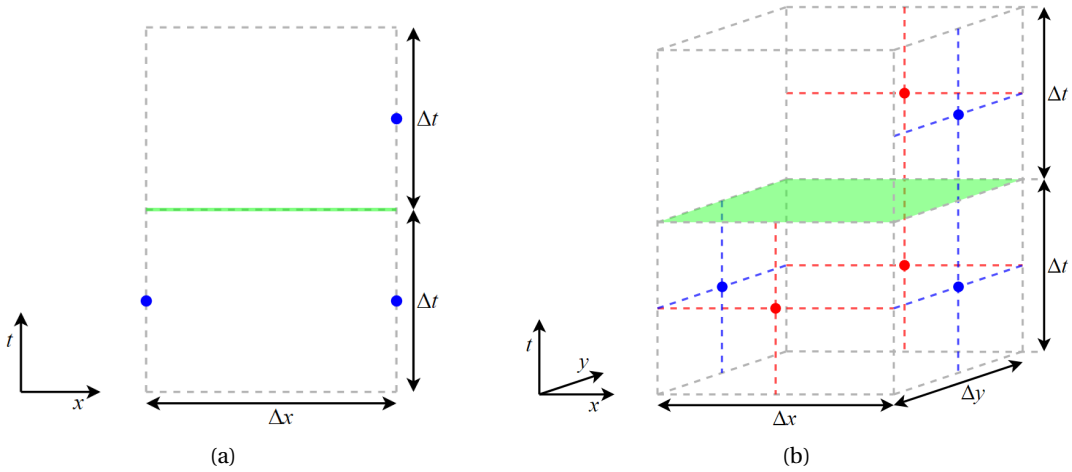


Figure 5.34: Visualization of how an orthogonal two-dimensional problem can be decomposed into two one-dimensional problems. (a) The one-dimensional problem. (b) The decomposed two-dimensional problem. The plane represents the average integral value, and the dots correspond to the locations of the interface nodes.

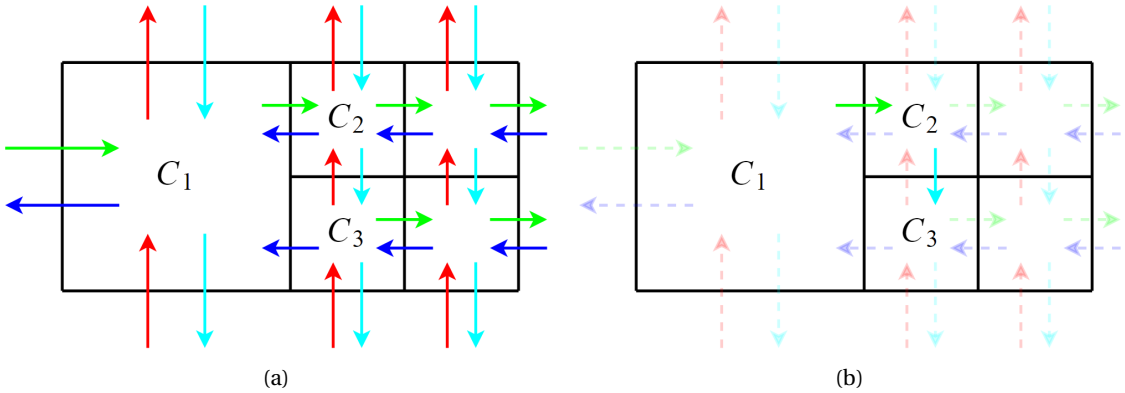


Figure 5.35: (a) A visualization of neighbouring cell references. Note that cell C_1 has no direct link to cell C_3. (b) The route through which the reference to cell C_3 can be obtained from cell C_1.

the interface. Thus, some attention is required on how to handle such cases. After handling these issues, transitioning from two to three dimensions is simpler than the transition from one to two dimensions as all concepts in three dimensions are direct extensions of the two-dimensional implementation.

5.5.2. Nonlinear extension

Besides exploring an increased number of spatial dimensions, it is interesting to touch upon the implications of applying the MTSETE3 method in combination with mesh adaption to nonlinear equations.

As mentioned before, Burgers' equation is often used to model the development of a discontinuity from smooth initial conditions. In Burgers' equation the characteristic velocity equals the solution value itself, meaning that information does not necessarily flow in a single direction. In terms of characteristic tracing, the characteristic origin may now have multiple solutions. This requires decision making on how to handle this situation. Also, for the linear case, the characteristic velocity is constant, such that a constant cell aspect ratio is sufficient to guarantee stability throughout the simulation. While Burgers' equation is theoretically unable to generate new extrema which surpass the extrema of the initial condition, non-monotone numerical schemes are likely to cause overshoots as discontinuities develop, which may destabilize the simulation if not acted upon. One way to solve this is to base the cells' aspect ratio on an overestimation of the maximum absolute local velocity. A more robust option would be to investigate the use of separate refinement factors in time and space. This would enable treating the problem locally, instead of penalizing the whole simulation. At the same time, this approach enables cells in which characteristic velocities are low to take larger time steps.

An important aspect in which the approach for Burgers' equation will differ from the linear case is the

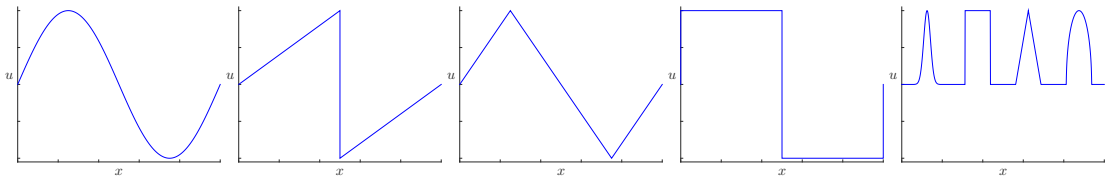
fact that regions which require high levels of refinement are known from the start for the linear case. This means that such regions will be contained in a fine mesh throughout the whole simulation. Contrarily, since Burgers' equation enables the development of discontinuities from smooth initial conditions, it is unclear where and when troublesome regions may arise. The main problem here is that many feature indicators aim to trigger as soon as the scheme attempts to resolve behavior for which the mesh is not fine enough. However, because this indicates that refinement level is not high enough, ideally one would want to go back in time and refine the smoother local solution, instead of inevitable interpolation of the erroneous solution. This requires the implementation of performing retroactive time steps, which increases the amount of memory used since the program will need some form of short-term memory.

Numerically approximating the Euler equations is much more complex. As seen in Chapter 2, the Euler equations describe a simplified version of the Navier-Stokes equations through three coupled non-linear conservation equations. Since the characteristic variables cannot be explicitly solved for, it no longer makes sense to solely rely on tracing along characteristics. The solution state is dependent on the collection of states inside a conical domain bounded by the relative acoustic velocities $u \pm a$. Because each of the three state variables may have individual refinement requirements, one may decide to use a separate mesh for each variable. However, shock waves result in discontinuous jumps in all state variables, such that these regions of high refinement overlap. A contact discontinuity is the only feature that causes a discontinuous density profile, without disrupting velocity and pressure profiles, which will thus be overly refined here. Further research is required to determine which approach is more appropriate.

6

Results

In this chapter the results of multiple simulations are discussed. The focus will be on evaluating the impact of adaptive mesh refinement in terms of accuracy improvement and the expectation on overall efficiency. To do this, simulations on the five waveforms as shown in Figure 6.1 have been performed. All simulations have been run for 100 global time steps, for varying levels of the maximum depth and refinement parameter Φ. The accuracy of the simulations will be quantified through analysis of the difference in L^2 error norm for an adaptive simulation and a simulation at constant maximum refinement depth. Because the results for the square wave were found to display trends nearly identical to those found for the sawtooth wave, these results are excluded here. However, for sake of completeness, these results can be found in Appendix C. To be able to investigate the effects of mesh adaption, for each waveform a reference solution is retrieved by turning off the adaption, thereby using a uniform mesh at the specified maximum refinement level.



(a) (b) (c) (d) (e)

Figure 6.1: The five waveforms used in the simulations: (a) Sine wave, (b) sawtooth wave, (c) triangle wave, (d) square wave and (e) Zalesak's waveform.

Because the scheme is stable up to a Courant number of 0.5, but also exact at this value, simulations are run at a slightly lower value of $\nu = 0.4$. Furthermore, the depth mapping parameter Φ is set to one of 0.1, 0.5, 1.0 and 2.0. Finally, it was observed that the structure of the mesh on initialization and a few steps after that were in some cases quite different. For this reason, it was chosen to not use mesh adaption in the initialization step, but refine to the maximum depth uniformly and from there on let the method decide which regions to coarsen. Because simulations are run for 100 time steps, this decision will be of negligible influence on further analysis.

This chapter contains five sections in which different aspects will be elaborated on. First, Section 6.1 focuses on the evolution of the mesh structure over time, varying the refinement aggressiveness parameter Φ. The impact of mesh adaption on the solution quality relative to a non-adaptive simulation is visualized in Section 6.2. Then, Section 6.3 aims at clarifying the effect of maximum refinement depth on the accuracy and efficiency in terms of computational savings. The results of a high resolution simulation on Zalesak's waveform, see Figure 6.1e, are presented in Section 6.4. Finally, a few concluding remarks on the results are made in Section 6.5.

6.1. Mesh deterioration

An important aspect of stable solution accuracy in prolonged simulations of linear advection with an adaptive mesh is sustaining the structure of the mesh. The value of Φ plays an important role in this. A too low value of this parameter will tend to unnecessarily increase the amount of cells in the domain. On the other hand, a too high value underestimates the need for refinement and the mesh will thus show a faster rate of resolution loss. The expectation is that due to numerical dissipation, all simulations will at some point be smeared out and converge to a one-cell mesh. These effects are observed best at a low initial resolution, so for low maximum depth levels. Therefore, the maximum refinement depth is set to 7, such that the mesh contains 128 cells at most. Tables 6.1, 6.2 and 6.3 contain figures of the mesh structure at different steps and Φ, for a sine, sawtooth and triangle wave respectively. All tables show that increasing Φ results in coarser meshes. Especially for the triangle wave the mesh coarsens rather quickly, due to the two kinks in the original waveform being increasingly smoothened by dissipation.

Table 6.1: Mesh visualizations at different time steps and varying Φ for a sine waveform. Horizontal axis corresponds to the spatial domain, and the vertical axis spans a single global time step. Colors indicate depth level. Legend at the bottom indicates refinement depth.

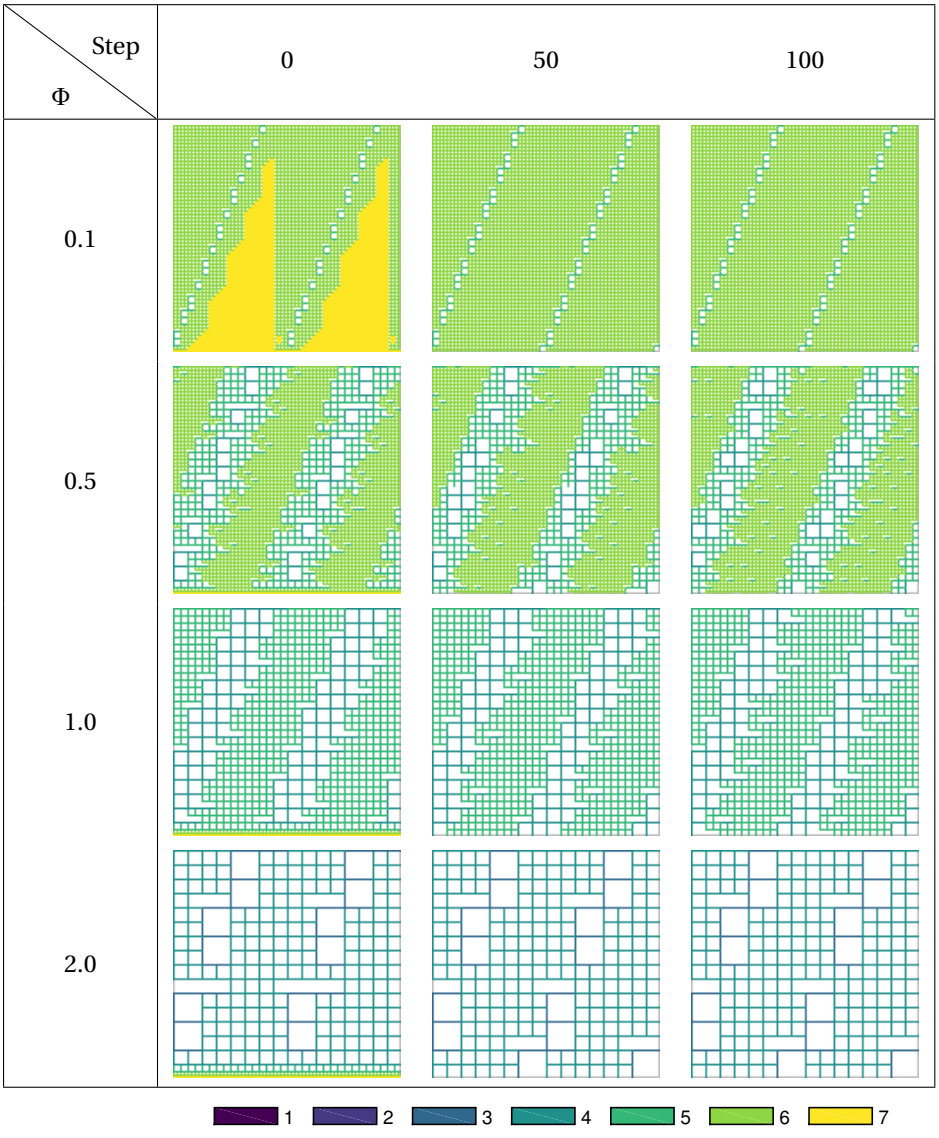


Table 6.2: Mesh visualizations at different time steps and varying Φ for a sawtooth waveform. Horizontal axis corresponds to the spatial domain, and the vertical axis spans a single global time step. Colors indicate depth level. Legend at the bottom indicates refinement depth.

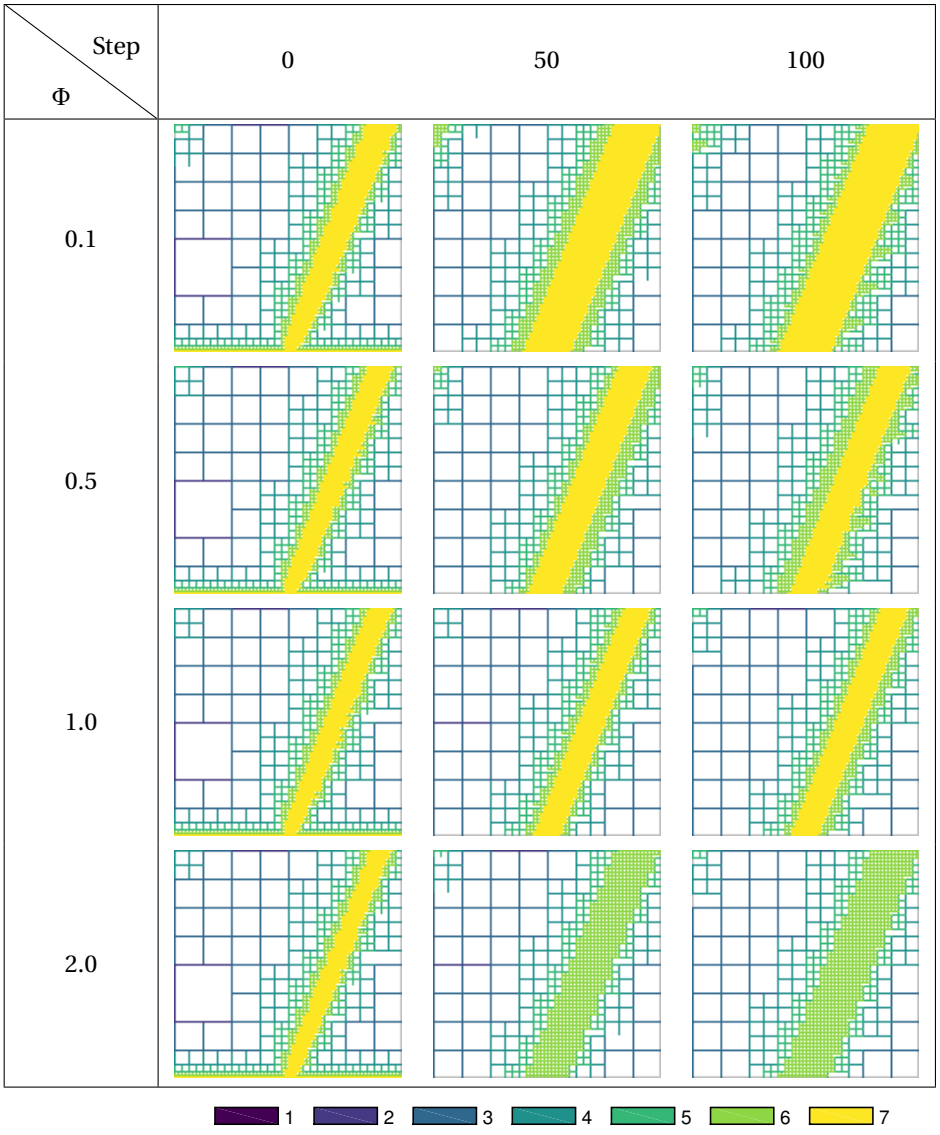
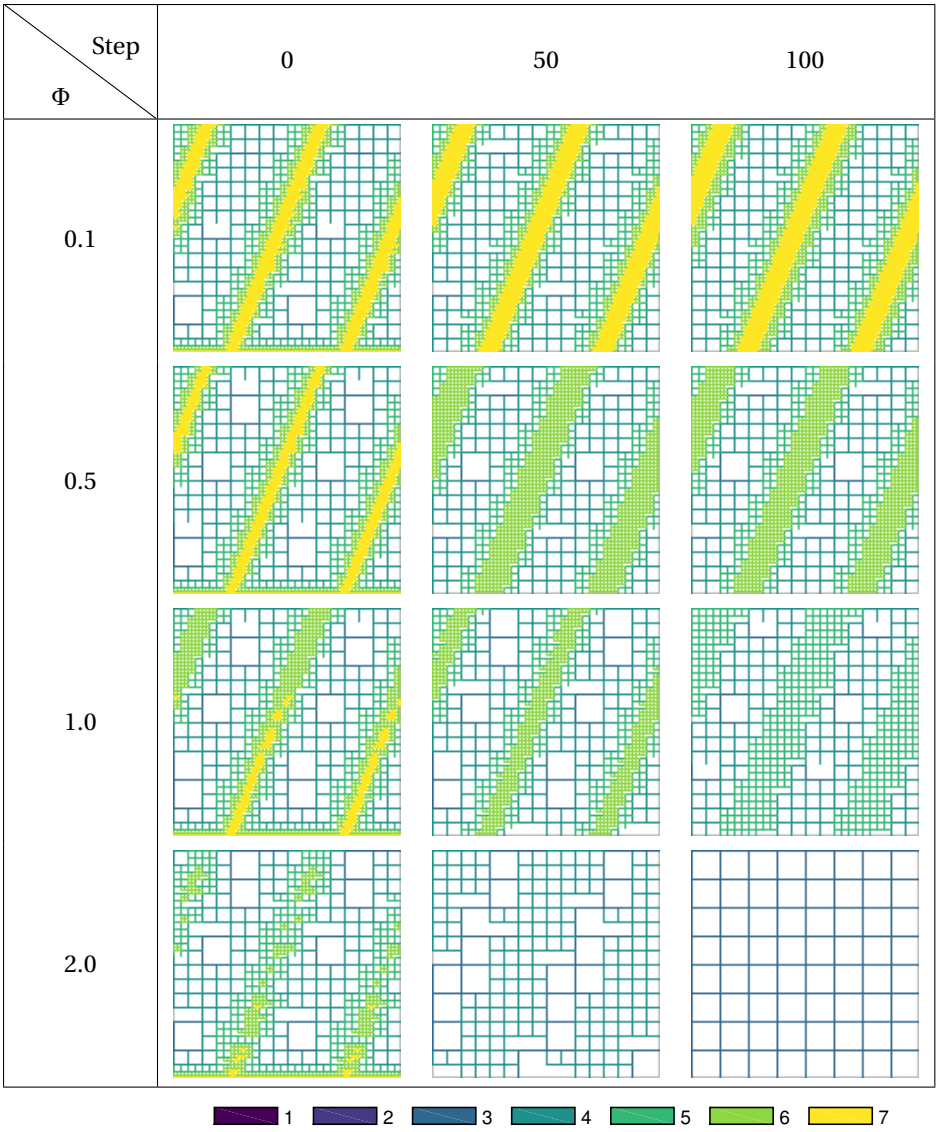


Table 6.3: Mesh visualizations at different time steps and varying Φ for a triangle waveform. Horizontal axis corresponds to the spatial domain, and the vertical axis spans a single global time step. Colors indicate depth level. Legend at the bottom indicates refinement depth.



The effects of Φ can also be observed by plotting the amount of cells for each depth level over time. Figures 6.2-6.4 display the same trend of decreasing refinement depth for increasing Φ. In Figures 6.4c and 6.4d emphasize the effect already observed in Table 6.3, where decay of the sharpness of the kinks causes rapid mesh coarsening.

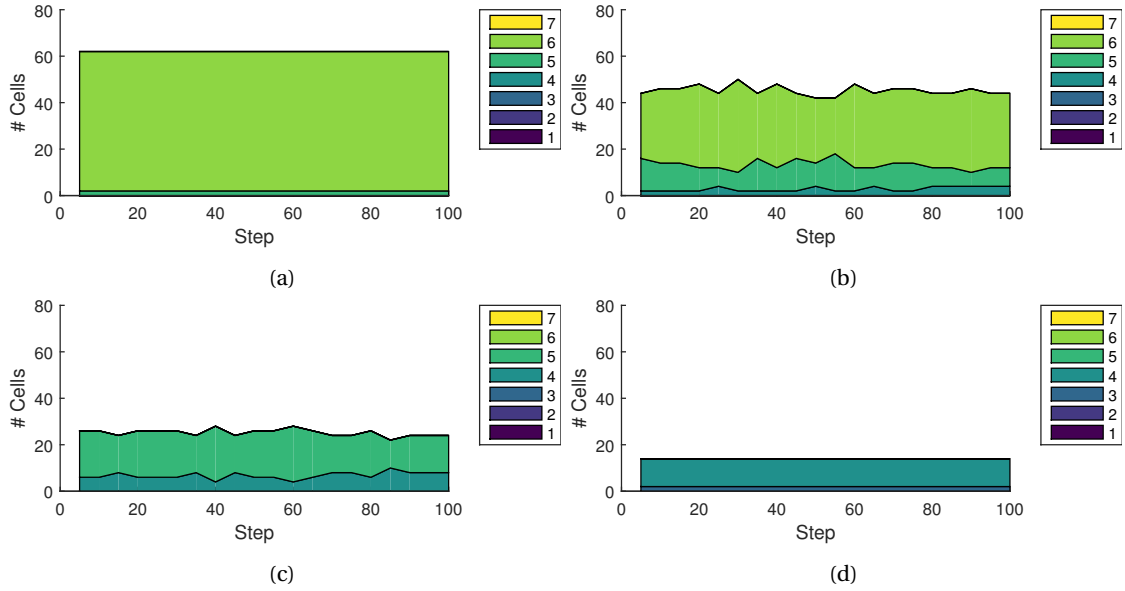


Figure 6.2: The cell count per depth level during the simulation of a sine wave for 100 global time steps. Legend entries indicate depth level. The first step is excluded as the simulation starts with a uniform mesh. (a) $\Phi = 0.1$, (b) $\Phi = 0.5$, (c) $\Phi = 1.0$, (d) $\Phi = 2.0$.

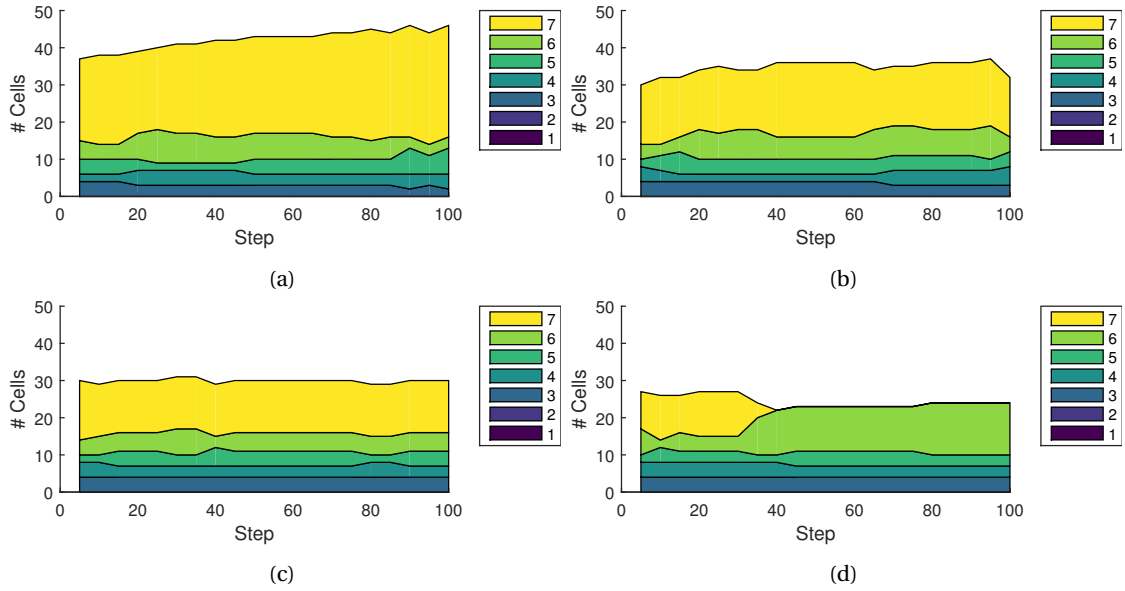


Figure 6.3: The cell count per depth level during the simulation of a sawtooth wave for 100 global time steps. Legend entries indicate depth level. The first step is excluded as the simulation starts with a uniform mesh. (a) $\Phi = 0.1$, (b) $\Phi = 0.5$, (c) $\Phi = 1.0$, (d) $\Phi = 2.0$.

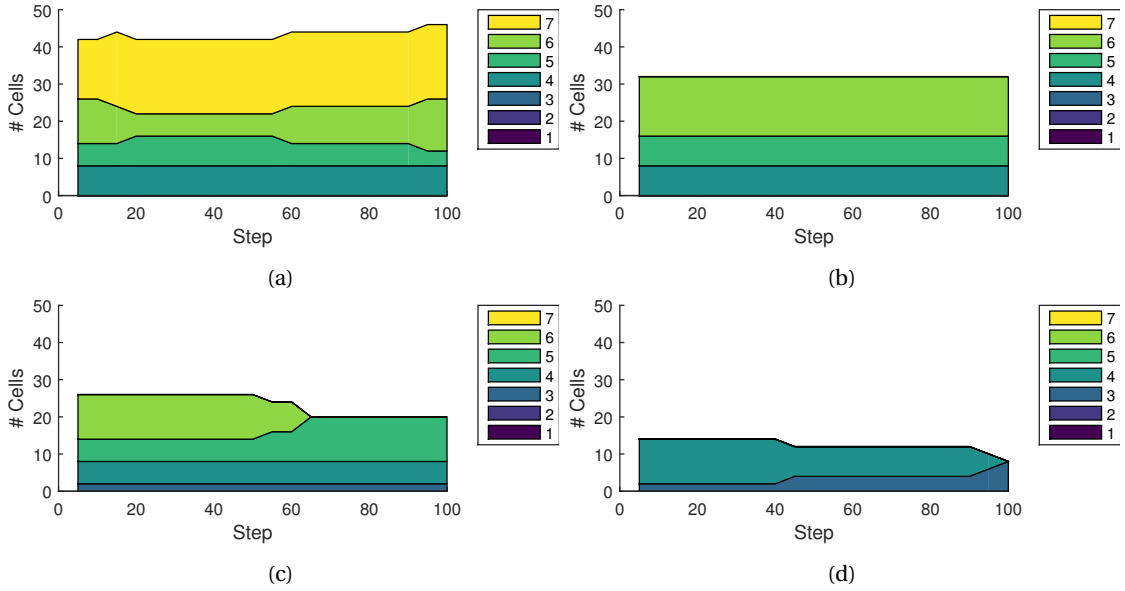


Figure 6.4: The cell count per depth level during the simulation of a triangle wave for 100 global time steps. Legend entries indicate depth level. The first step is excluded as the simulation starts with a uniform mesh. (a) $\Phi = 0.1$, (b) $\Phi = 0.5$, (c) $\Phi = 1.0$, (d) $\Phi = 2.0$.

6.2. Accuracy

In this section the accuracy of the adaptive simulations will be analyzed. This will be done by inspecting the L^2 error norm, which for one cell is defined as:

$$L^2_{cell} = \sqrt{\int_0^1 (u - u_{exact})^2 \, \mathrm{d}\xi}. \tag{6.1}$$

Summing over all cells in the mesh, scaling for cell size, the total error of the mesh becomes:

$$L^2 = \sum_{cells} L^2_{cell} \Delta x_{cell}. \tag{6.2}$$

In order to compare simulations with adaption enabled to their uniform analogue (in terms of computational effort), it is necessary to determine how many cells the uniform mesh should contain. To do this, first the simulations are run with adaption enabled and the total number of computational cells used during the whole simulation is kept track of. With this number, the amount of cells for the uniform mesh such that in total an approximately equal amount of cells is used in both simulations can be found through:

$$N = \left\lceil \sqrt{\frac{T}{S}} \right\rceil, \tag{6.3}$$

where T represents the total number of cells used in the adaptive simulation, S the number of steps which equals 100 here, and finally N the resulting number of cells in the uniform mesh. As reference, the results for uniform simulations at maximum refinement are included in the plots. This allows us not only to see whether the adaptive simulation actually displays smaller errors compared to a uniform solution of approximately equal computational intensity, but also to which extent the adaptive simulation approximates the smallest error possible with this mesh configuration and update method. Figures 6.5, 6.6 and 6.7 show for varying Φ how the L^2 error develops over time. The maximum depth level is still set to 7, and besides Φ, the legend entries also indicate the cell load, which we define as the percentage of total number of cells used in a simulation compared to the total number of cells used for the reference

solution. The lower the cell load, the higher the potential for mesh adaption to efficiently distribute computational resources over the domain. It should be kept in mind that a low cell load does not imply reasonable solution accuracy. Figure 6.5, which shows the error development for a sine wave, proves that mesh adaption is not useful for use on smooth waveforms. For all tested values of Φ, the adaptive simulation performs worse than its uniform analogue.

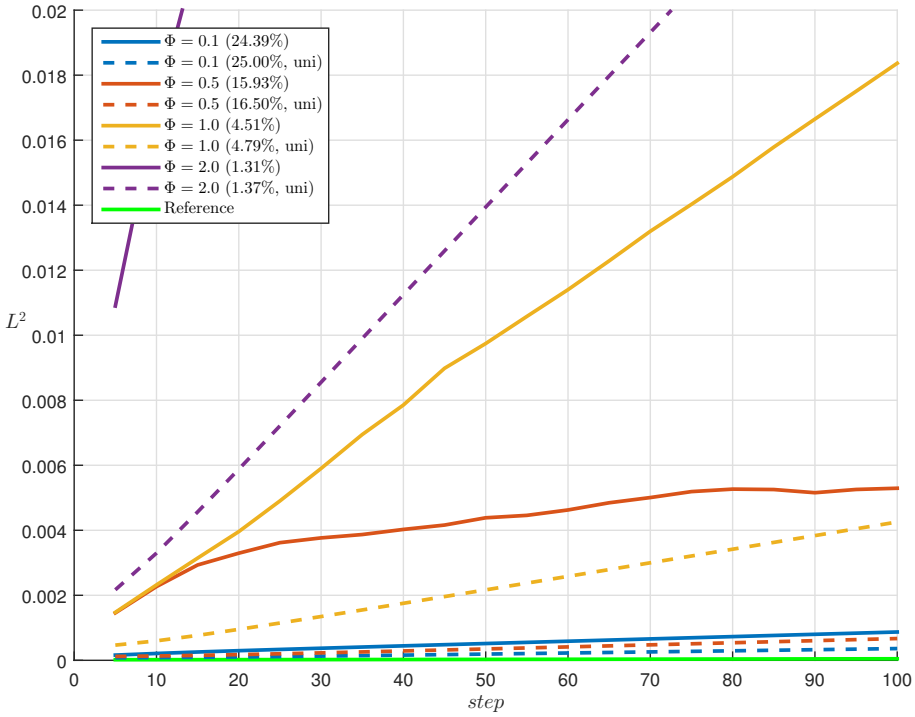


Figure 6.5: L^2 error development over time for a sine wave. Maximum refinement depth is set to 7. The green line indicates the uniform maximum refinement reference simulation. The dashed lines indicate the uniform mesh simulations with approximately equal cell load.

For a sawtooth wave, see Figure 6.6, the positive effect of mesh adaption becomes clear. From this figure it can be observed that for decreasing Φ, both the adaptive and uniform simulations approach the reference solution. This is expected as the number of cells increases. However, it is also shown that the adaptive simulations converge to the reference solution faster than the uniform solutions do. At $\Phi = 0.1$ and after 100 steps, the L^2 error of the adaptive solution is 2.1% higher than the error of the reference solution, while the error of the uniform analogue is 71.6% higher. For $\Phi = 2.0$, the time derivative of the error shows an instantaneous jump after around 30 time steps. This behavior corresponds to the results shown in Figure 6.3d, where between step 30 and 40 all cells of depth 7 are coarsened to depth 6, resulting in faster error growth.

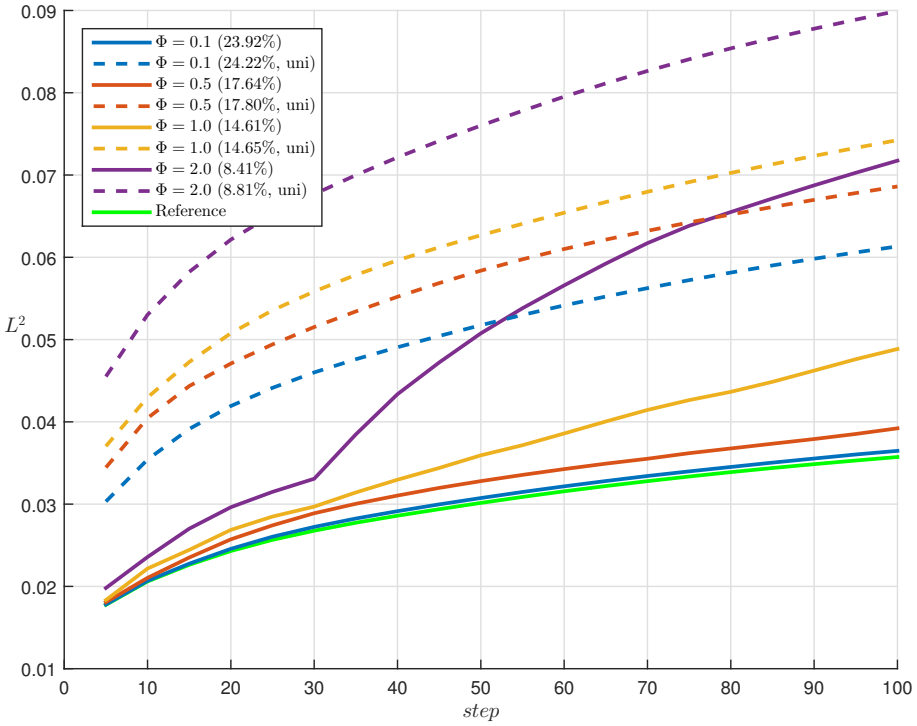


Figure 6.6: L^2 error development over time for a sawtooth wave. Maximum refinement depth is set to 7. The green line indicates the uniform maximum refinement reference simulation. The dashed lines indicate the uniform mesh simulations with approximately equal cell load.

Finally, Figure 6.7 contains the error development results for a triangle wave. Similar behavior as observed for the sawtooth wave is found. The improvements of the adaptive solution over their uniform analogue are still apparent for lower values of Φ, although smaller. For Φ = 2.0 and Φ = 1.0, eventually the error growth rates of the adaptive simulations surpass those of the uniform simulations, where for Φ = 0.5 and Φ = 0.1 the error development appears to be more stable, at least until the end of the simulation.

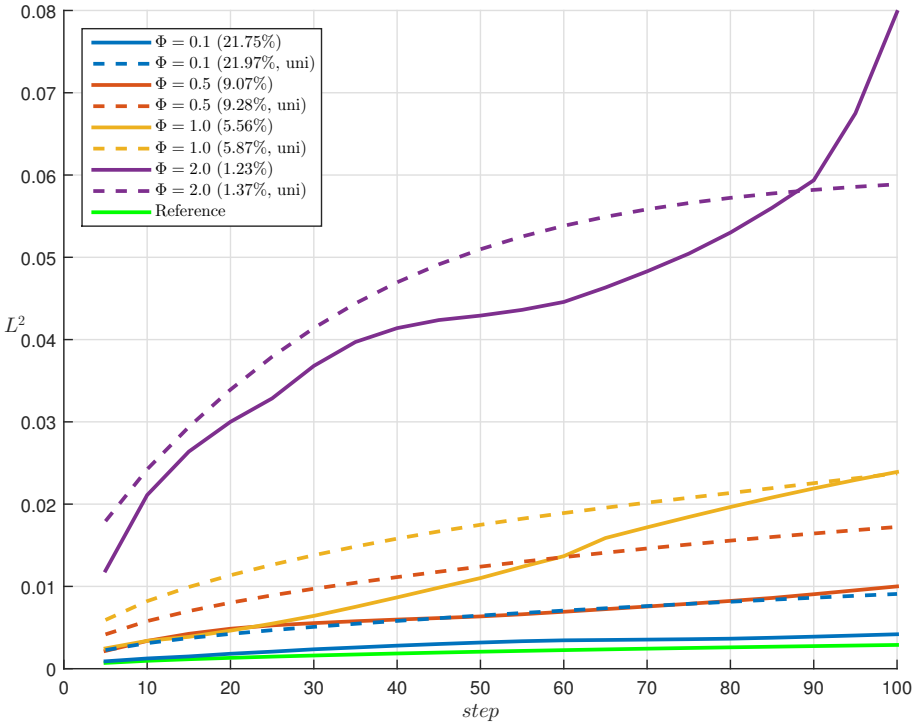


Figure 6.7: L^2 error development over time for a triangle wave. Maximum refinement depth is set to 7. The green line indicates the uniform maximum refinement reference simulation. The dashed lines indicate the uniform mesh simulations with approximately equal cell load.

6.3. Refinement depth

In the previous section we have seen that the accuracy of the simulation over time is largely affected by the value of Φ. Lowering the value of Φ decreases the L^2 error, but also increases the amount of cells in the mesh. The simulations in the previous sections have used a fixed maximum refinement depth of 7. In this section the effect of increasing refinement depth on performance and accuracy will be elaborated on.

The maximum depth will be increased from 7 to 15, in steps of 2. Table 6.4 shows the maximum amount of cells that the mesh can contain, along with the total amount of time steps performed for the reference solutions.

Depth	No. cells	No. time steps
7	128	12800
9	512	51200
11	2048	204800
13	8192	819200
15	32768	3276800

Table 6.4: The maximum number of cells in the mesh per depth level, along with the number of time steps taken by the reference solution (100 global steps).

First, we will consider the performance in terms of computational savings when using the adaptive method. Computational efficiency is difficult to compare as it is directly dependent on the way in which the code is written. It is only after this code has been optimized that conclusions can be drawn about its performance, although even then, other variables outside the scope of code structure that affect performance remain. Therefore it is assumed here that, in general, the cost of having the additional memory overhead and resource usage in the adaptive code will be outweighed by the resource savings resulting from using significantly less computational cells. Thus, these savings will be solely expressed in terms of the number of cells here.

As seen in the previous section, cell load is the parameter that expresses how many cells are used

relative to the amount of cells used in the reference simulation. Because it was shown in the previous section that the accuracy of the reference solution serves as an asymptote for the accuracy of the adaptive solution, the cell load does not represent the exact resource savings accountable to using less cells. It is not trivial to quantify these exact savings, as one would have to find the uniform mesh resolution such that it displays approximately equal error development. In Figures 6.8, 6.9 and 6.10, the cell load for varying Φ is shown for the three waveforms discussed, as function of the maximum refinement depth. All three figures show the same pattern of decreasing cell load with increasing depth. This effect is particularly apparent for the sawtooth and triangle waveforms. This indicates that the increased maximum depth allows the solution method to isolate the troubling regions in a smaller part of the domain, by relatively less cells.

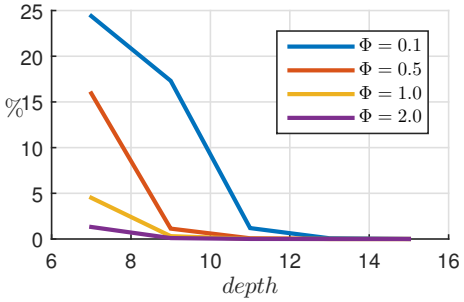


Figure 6.8: Cell load for a sine wave as a function of maximum refinement depth, for varying Φ.

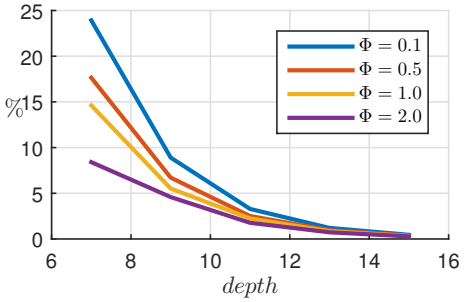


Figure 6.9: Cell load for a sawtooth wave as a function of maximum refinement depth, for varying Φ.

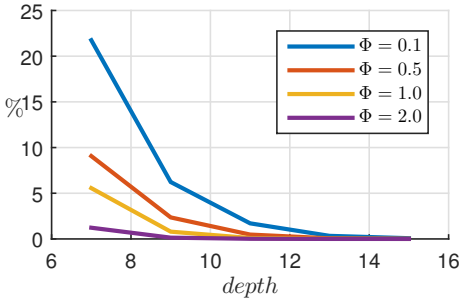


Figure 6.10: Cell load for a triangle wave as a function of maximum refinement depth, for varying Φ.

In order to compare the accuracy the L^2 error is again considered. Because we want to show the effect of varying depth on the error, instead of showing the error over time, only the error at the end of the simulation is shown, indicated by L^2_{100}. As for the previous results, Figures 6.11, 6.12 and 6.13 contain the plots for the sine, sawtooth and triangle wave. The error for the sawtooth waveform shows stable behavior when varying the maximum refinement depth. Especially for $\Phi = 0.5$ and $\Phi = 0.1$, the errors tend to stay parallel to the reference asymptote. This is much less the case for the triangle wave. For increasing depth the errors stray away from the reference solution much quicker, even showing

converging behavior, such that at some point, increasing the maximum refinement depth does not seem to really decrease the error anymore. This converging behavior is reflected by plotting the cell count over time for a maximum refinement depth of 15, see Figure 6.14. This plot shows that soon after the start of the simulation, all cells of depth 15 are coarsened such that for the majority of the simulation cells of depth 14 are used. This indicates that a value of Φ smaller than 0.1 is necessary at this depth level for the triangle wave in order to decrease error levels further.

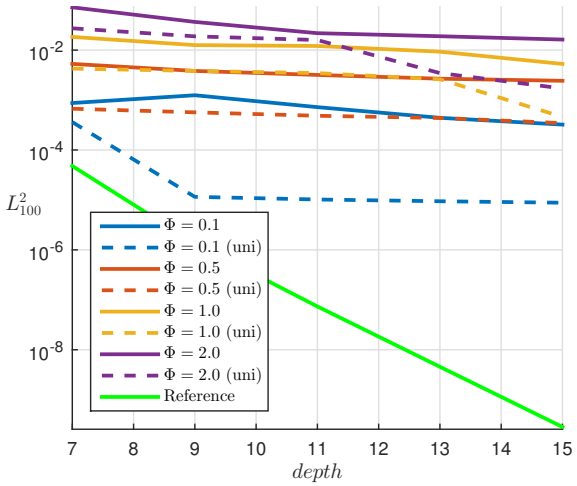


Figure 6.11: L^2 error after 100 global time steps for a sine wave as a function of maximum refinement depth, for varying Φ.

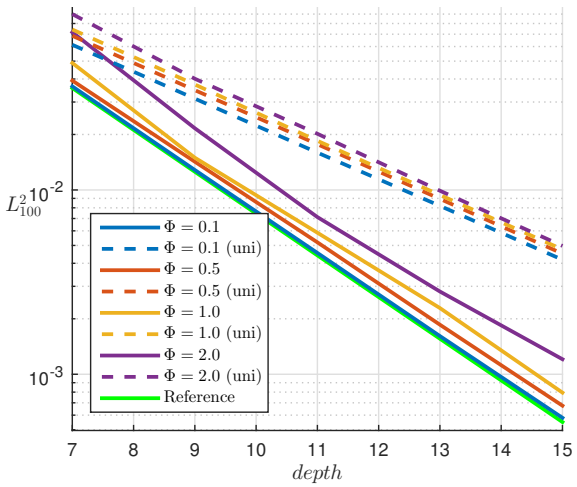


Figure 6.12: L^2 error after 100 global time steps for a sawtooth wave as a function of maximum refinement depth, for varying Φ.

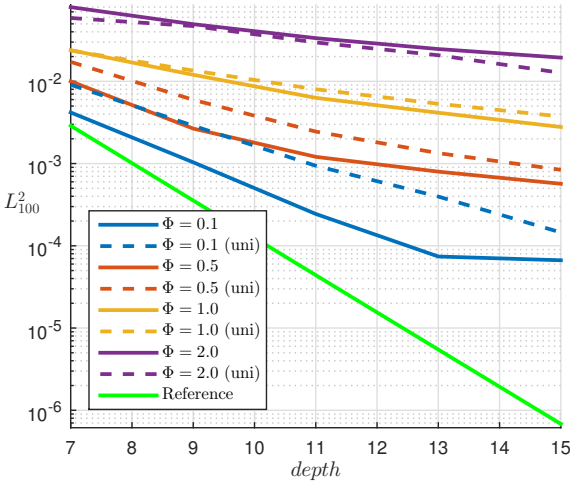


Figure 6.13: L^2 error after 100 global time steps for a triangle wave as a function of maximum refinement depth, for varying Φ.

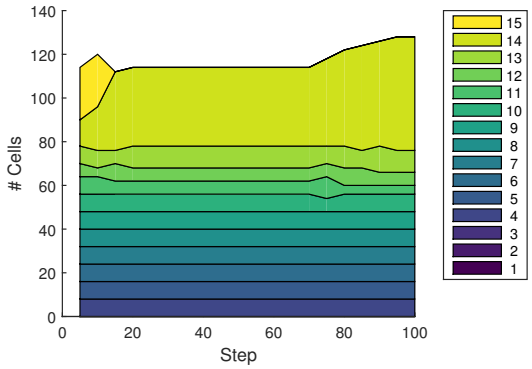


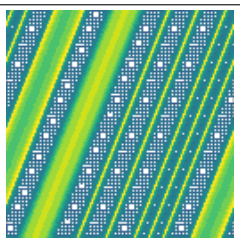
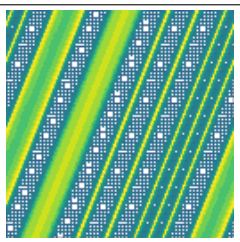
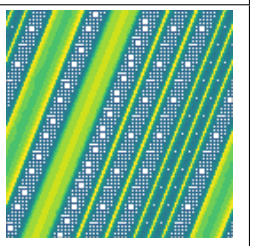
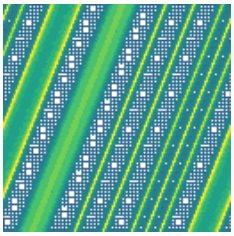
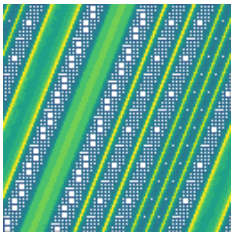
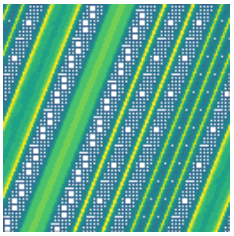
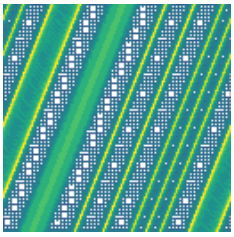
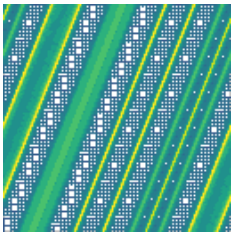
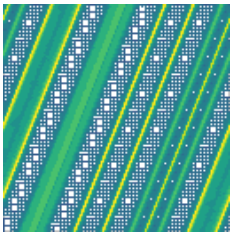
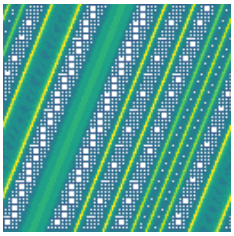
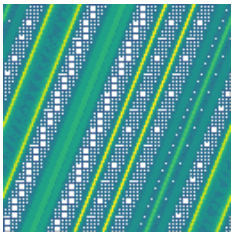
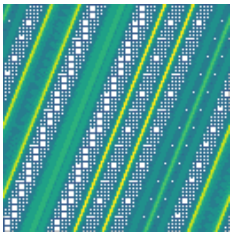
Figure 6.14: The cell count per depth level during the simulation of a triangle wave for 100 global time steps. Legend entries indicate depth level. The first step is excluded as the simulation starts with a uniform mesh.

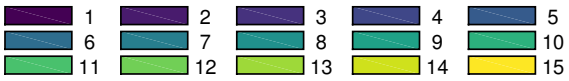
6.4. Zalesak's waveform

As a final summarizing test case, simulations have been run for Zalesak's waveform. As explained previously, this waveform is a collection of four individual waveforms that each have their own specific shape and accompanying challenges in terms of shape preservation in linear advection. At the end of this section the reconstructed solution is shown.

Because of the more compressed waveforms and the eagerness to do a high resolution simulation, the maximum refinement depth has been set to 15. Analogous to the previous sections, several plots on the mesh structure and solution accuracy will be presented here. First, Table 6.5 aims at revealing changes in mesh structure over time by showing the mesh for a global time step at different time steps. In the figures in this table the different waveforms can be easily distinguished. The two discontinuities of the square wave and three kinks of the triangle shape are recognized by the five thin consecutive regions of high refinement. Also, the refinement patterns of the Gaussian and elliptic wave appear to be each other's inverse, where the Gaussian reaches maximum refinement in the center of the wave while the elliptic shape displays maximum refinement at the sharp edges.

Table 6.5: Mesh visualizations at different time steps and varying Φ for Zalesak's waveform. Horizontal axis corresponds to the spatial domain, and the vertical axis spans a single global time step. Legend at the bottom indicates refinement depth.

Φ \ Step	0	50	100
0.1			
0.5			
1.0			
2.0			



The fact that there is visually barely any change in mesh structure during the simulation, especially for lower Φ, shows the stability of the mesh composition. This observation is also reflected by Figure 6.15, showing only gradual changes in total cell count.

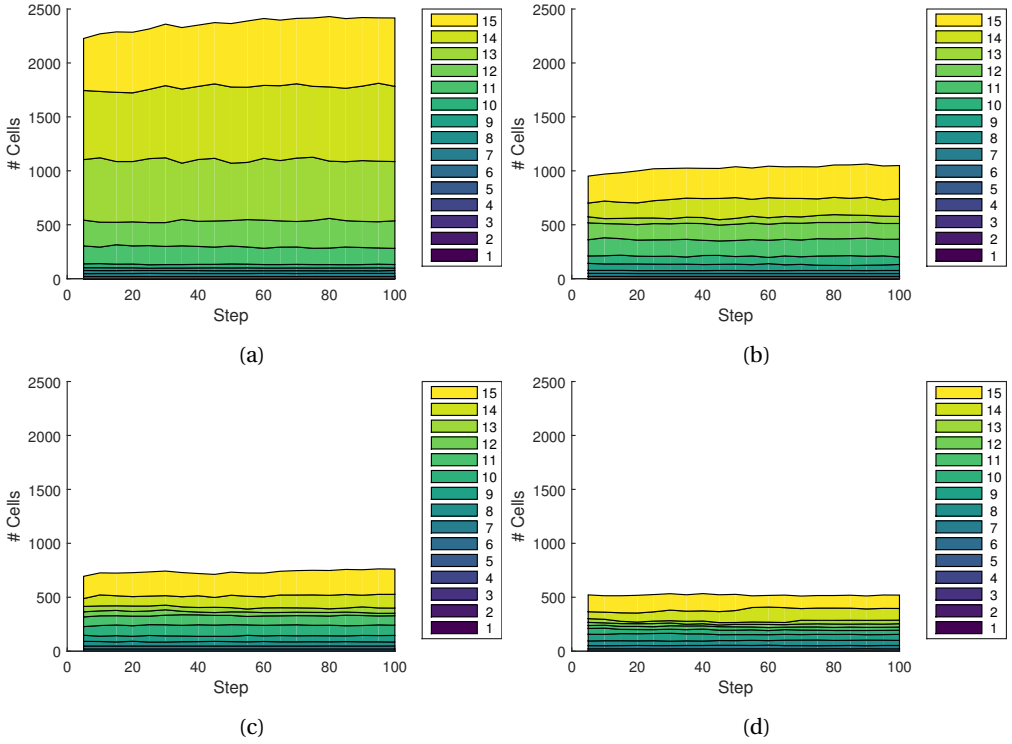


Figure 6.15: The cell count per depth level during the simulation of Zalesak's waveform for 100 global time steps. Legend entries indicate depth level. The first step is excluded as the simulation starts with a uniform mesh. (a) $\Phi = 0.1$, (b) $\Phi = 0.5$, (c) $\Phi = 1.0$, (d) $\Phi = 2.0$.

Figure 6.16 shows how the L^2 error term develops over time in relation to the reference solution. Similar trends to those found in section 6.2 are displayed, where the error of the adaptive simulation strays from the reference solution, eventually surpassing the error of the uniform solution at equal cell load.

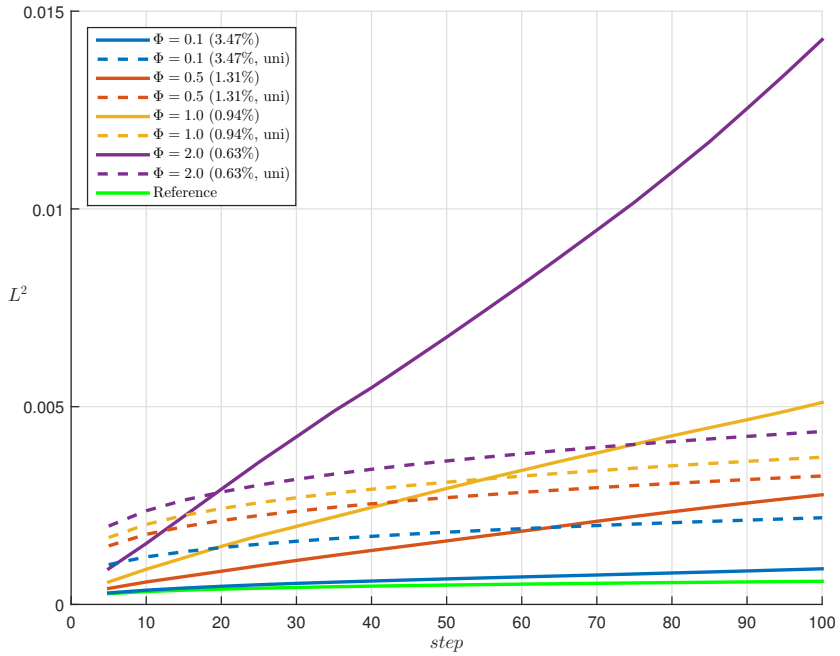


Figure 6.16: L^2 error development over time for Zalesak's waveform. Maximum refinement depth is set to 15. The green line indicates the uniform maximum refinement reference simulation. The dashed lines indicate the uniform mesh simulations with approximately equal cell load.

The presence of discontinuities and kinks within the domain raises the expectation that an increase in resolution through means of a higher refinement depth allows a more accurate and therefore more efficient distribution of cells, lowering the cell load. This expectation is once more verified based on the results contained in Figures 6.17 and 6.18a.

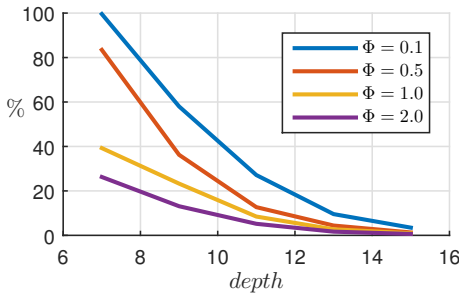
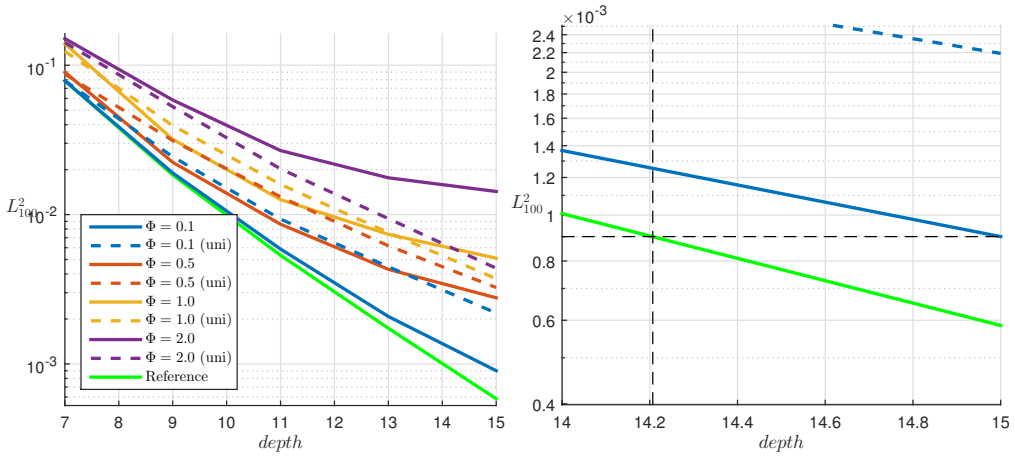


Figure 6.17: Cell load for Zalesak's waveform as a function of maximum refinement depth, for varying Φ.

An interesting result is how in Figure 6.17 the cell load for $\Phi = 0.1$ reaches 100% at a refinement depth of 7. This means that due to the waveforms being compressed on a relatively small domain, no pair of cells is deemed smooth enough to collapse into a cell of depth 6, such that this specific simulation is in fact equal to the uniform reference solution. This observation is reinforced by Figure 6.18a, where L^2_{100} of the adaptive and reference solution lie on top of each other. Figure 6.18b shows that the error level of a uniform solution with $2^{14.2068} \approx 18909$ cells is approximately equal to that of the adaptive solution with a maximum depth of 15. However, the uniform simulation would require ~9.6 times as many computational cells compared to the adaptive simulation.



(a) (b)

Figure 6.18: (a) L^2 error after 100 global time steps for Zalesak's waveform as a function of maximum refinement depth, for varying Φ. (b) Detailed view showing that the adaptive simulation reaches an error level equal to that of a uniform simulation with $2^{14.2068}$ cells.

In Figure 6.19 the complete solution reconstruction is shown for a simulation run at a maximum refinement depth of 15. This figure displays the exact solution, the reference (uniform + maximum refinement) solution, the solution retrieved from the adaptive simulation at cell load 0.0347 and finally the uniform solution at equal cell load.

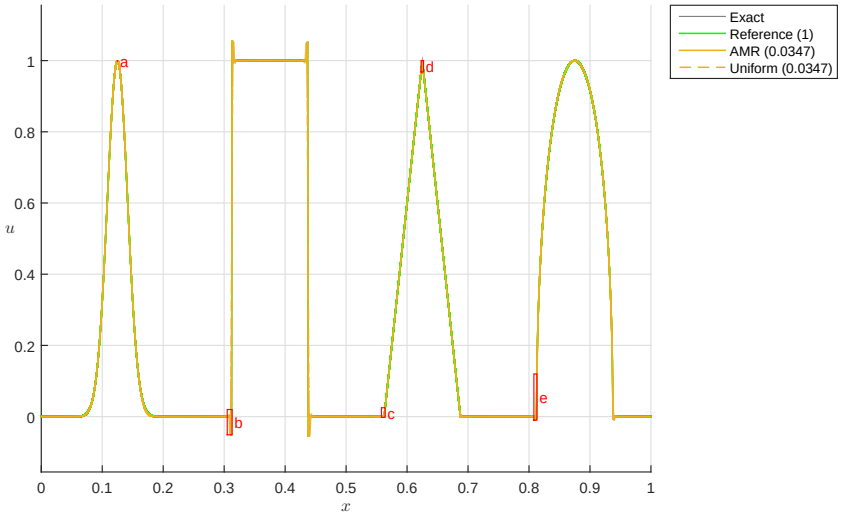


Figure 6.19: The final results after advecting Zalesak's waveform for 100 global time steps, with $\Phi = 0.1$.

Due to the high resolution, these four results overlap. Therefore, five regions (a, b, c, d, e) are focused upon in Figure 6.20 in order to show how the different simulations compare to the exact solution. Since a linear update method is used and each of these closeups features a discontinuous change in either signal or signal derivative, overshoots and other relatively large deviations from the exact solution shape are expected. However, these closeups indicate that the adaptive simulation approaches the reference solution much better than the uniform simulation does.

6.5. Concluding remarks

In the present chapter an attempt has been made at presenting the impact of mesh adaption in a structured way by treating several relevant aspects separately. It has been observed that linear advection of low bandwidth waveforms does not benefit from mesh adaption, as is expected. For waveforms of higher bandwidth, especially where high-frequency features comprise only a small part of the spatial domain, the advantages of mesh adaption become more apparent. At this point it is impossible to quantify the

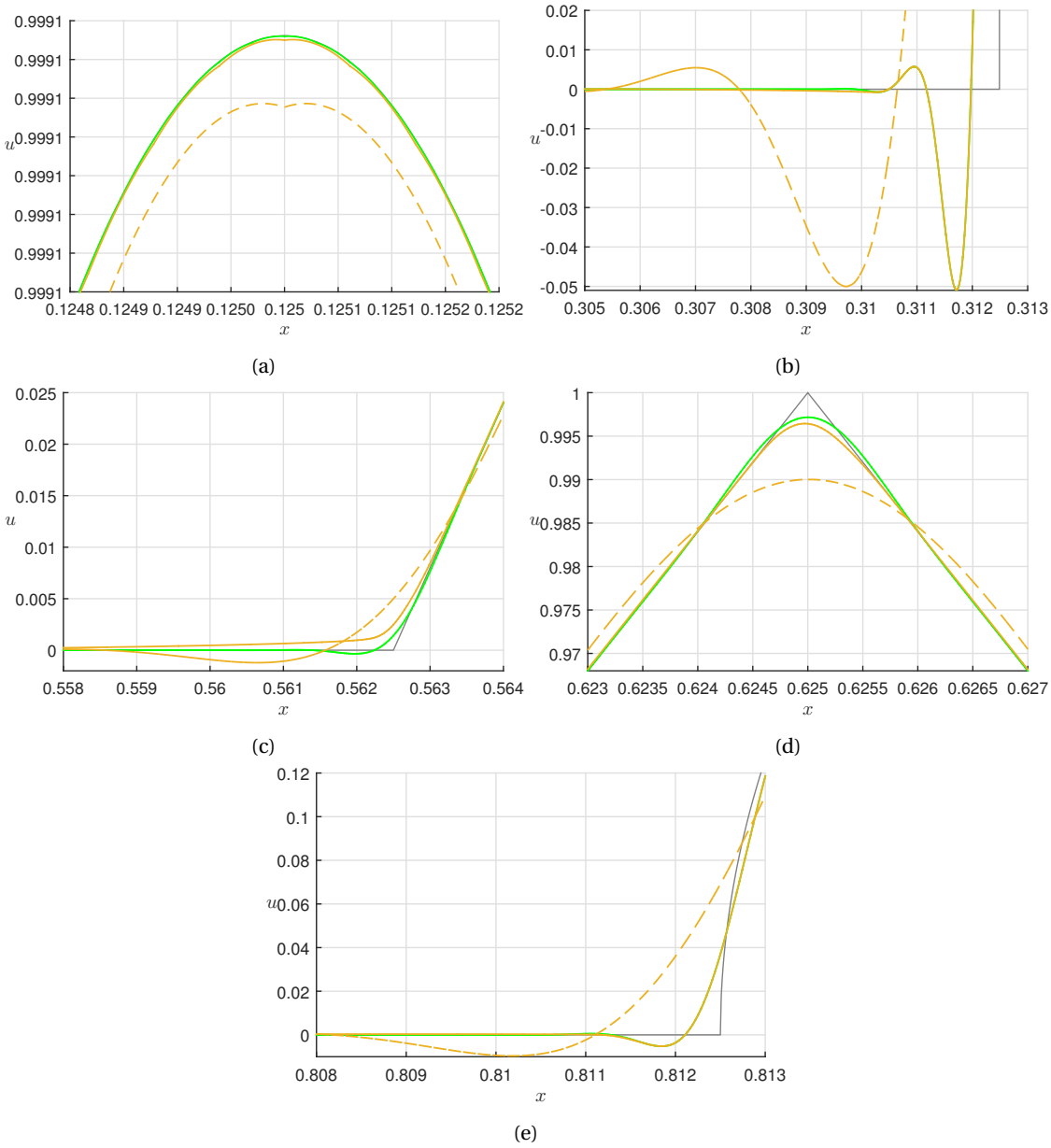


Figure 6.20: The five regions from Figure 6.19 on which is focused to compare the different solutions. Legends are omitted, but the line styles correspond to those displayed in Figure 6.19.

computational efficiency in an absolute way, as only after code optimization it becomes clear whether the additional adaption-induced efforts are truly outweighed by an overall decrease in number of cells. What can be stated, however, is that the high resolution simulation of Zalesak's waveform at a uniform refinement depth of 15 took 173 minutes to complete, while the adaptive simulation finished in 58 minutes, which is nearly 3 times less. However, since the total number of cells used during the reference simulation is approximately 28 times that of the adaptive simulation, it is estimated that the adaptive simulation increases computation time by a factor of 10 compared to a simulation of equal cell count. This shows that, with the code in its current state, it is quite costly to enable adaptivity. However, it should again be noted that these are very rough estimates and the actual price tag can only be determined after code optimization. Furthermore, only the one dimensional case is considered here. The relative cell savings increases for higher spatial dimensions, corresponding to exponentially lower cell loads.

7

Conclusions

Since technological advancements have pushed the extent of research in the field of fluid dynamics past the sound barrier, there is a demand for numerical methods that are able to correctly simulate the extremely rapid flow property changes that occur in shock waves. Over the past quarter century there has been a lot of discussion on the best way to approach problems with such large range of scales, which has resulted in an extensive and still expanding collection of numerical schemes. Each scheme has its strong and weak points and since many of the more successful methods incorporate nonlinearity in some way, it is generally not possible to prove that a method will yield correct results for all cases. Due to this, computational fluid dynamics remains an active research area.

Active Flux schemes provide a flexible framework where an extra degree of freedom is introduced on grid cell interfaces such that the conservation constraint can be isolated, enabling the use of a non-conservative interface update method to complete the scheme. A third order update method based on characteristic tracing was proposed by Eymann and Roe. In this thesis, update methods have been constructed by minimizing the truncation error of a Taylor series expansion. A fourth order method using the same stencil as the characteristic tracing update method was found to obviously have a lower dissipation rate, but also suffer from large oscillations near discontinuities, which is not surprising for an even-order scheme. A third order method with a more compact stencil was then derived, which showed similar behavior as the characteristic tracing method, although being somewhat less dissipative. Further investigation of the interface and integral values used in this method resulted in the discovery that they are staggered in time.

Contamination of a large part of the solution through the Gibbs phenomenon has led to the decision to implement mesh adaption in order to provide a mechanism to increase mesh resolution where needed and to slow down spreading of spurious oscillations. Due to its ability to adjust the mesh to local conditions in both the spatial and temporal domain without negatively affecting the remainder of the mesh, h-refinement is the most sensible choice of mesh adaption in the field of fluid dynamics and is thus also used here. Because of the additional interface value, hanging nodes occur between cells of unequal depth. The fact that the interface and integral values are staggered in time further complicated the cell refinement and coarsening procedures through an increase in the number of possible states. In total, 16 unique refinement states and 8 unique coarsening states are accounted for.

The most important aspect of mesh refinement is determining the criteria for refinement and coarsening. Since a cell-based method is used, a feature indicator was found to be most convenient to use as adaption criterion. Three feature indicators were tested among which the one based on the absolute value of the difference in reconstruction derivatives was selected due to its desirable response near discontinuities and kinks. A configurable parameter Φ was introduced which controls the strictness of refinement. The lower Φ, the lower the threshold for a cell to refine.

Simulations of various waveforms have been run with varying Φ and maximum refinement depth. As expected due to the small range of scales, applying mesh adaption for a globally smooth waveform such as a sine wave resulted in a negative impact in terms of both accuracy and computation time. For decreasing Φ, the simulation converges to a uniform mesh at maximum refinement depth. Simulations

of a sawtooth and triangle wave, characterized by the presence of a discontinuity and two kinks respectively, showed how mesh adaption is increasingly beneficial for higher maximum refinement depths. For the triangle wave it was found that the parameter Φ should vary inversely with maximum refinement depth in order to maintain the accuracy improvement of increased maximum refinement depth. Finally, as a verification case, a high-resolution simulation of Zalesak's waveform was analyzed. This waveform serves as an interesting test case as it is composed of multiple waveforms with each a challenging characteristic shape. As was also done for the other waveforms, snapshots of the mesh structure at different instances throughout the simulation showed the stability of the mesh composition. For a low value of Φ, the spurious oscillations produced by the Gibbs phenomenon tended to slowly widen the regions of maximum refinement throughout the simulation, while keeping the amount of cells of lower depth nearly constant. It was found that at $\Phi = 0.1$ and a maximum refinement depth of 15, a simulation on a uniform mesh would need ~9.6 times as many cells as the adaptive simulation in order to reach the same error level. On the other hand, higher values of Φ result in a higher overall dissipation rate due to a lower average cell depth. This increases the rate of smoothening, eventually causing indicator values to fall below thresholds at which point cells collapse into coarser cells.

In conclusion, the results reflect that, given the appropriate settings, adaptive mesh refinement is beneficial for fluid flow problems which require large variations in mesh resolution. Especially for higher dimensional problems it becomes increasingly important to be able to efficiently distribute computational resources. One has to keep in mind that the presented implementation of mesh adaption is accompanied by several issues, e.g. the introduction of tunable parameters, mesh deterioration and an increased complexity of verification of the results. The possible performance increase, however, provides the motivation for further research and overcome these limitations.

8

Recommendations

Based on experience gained during this thesis, there are several aspects on which can be improved. The following list concisely provides some recommendations for future research.

- The results have shown that no matter the level refinement, a higher-order linear scheme near a discontinuity or otherwise instantaneous change induces oscillatory behavior. Some form of non-linearization of the method is required to suppress these oscillations. This is often done through the use of limiters. However, a common issue of limiters is that they tend to wrongly activate near smooth extrema, introducing an unnecessary amount of diffusion at these locations, resulting in flattened shapes. The adaptive mesh refinement framework as presented in this thesis provides a simple way of preventing incorrect limiter activation through means of checking the refinement depth in addition. For example, one could constrain a limiter to only be activated at maximum refinement depth, analogous to turbulence models where dissipation only occurs at the smallest length scales.
- Several of the 16 refinement and 8 coarsening procedures as presented in this thesis involve interpolation. This implies that information loss is taking place, and these procedures are not reversible. Thus, repeated refining and coarsening tends to add dissipation and degrade the solution. It may be worthwhile to investigate adjusting these procedures so as to minimize the dissipation introduced or even remove it completely.
- There is room for improvement in defining the refinement/coarsening criteria. Feature detection is a relevant research topic in disciplines other than computational fluid dynamics and covers a separate research area on its own. In comparison, the feature detector used in this thesis is very basic, and more advanced feature detection is likely to improve the results.
- Setting the depth mapping parameter Φ is currently manually done through trial and error. Investigating the influence of this parameter on the accuracy of a simulation may lead to an empirical relationship between Φ and accuracy, allowing Φ to be derived from a more tangible and universally used parameter. Furthermore, at the moment the depth binning happens in a linear fashion, meaning that the domain $[0, \Phi]$ is subdivided in equally spaced bins. More complex distributions can be investigated in order to create nonuniform adaption sensitivity.
- Of course, the exact solution for the linear advection equation is easily derived and is therefore not particularly interesting. It simply serves as a tool to investigate various important properties of a numerical scheme. A logical next step would be to implement the method for Burgers' equation. The ability to form discontinuities from smooth initial conditions poses additional challenges for the scheme in terms of correct development and propagation of the waveform.

A

Single expression derivation

The goal of this appendix is to rewrite the compound Active Flux scheme formulation as presented by Eymann and Roe to a single explicit update expression with the values defined on a uniform grid. In the original scheme, the computational domain consists of cells, with solution values at the cell interfaces and average integral values on the cell domain. Characteristics are used to advance the edge values and calculate the interface flux terms, with which the integrated values are updated.

For linear advection, interface values at the next time level are found through:

$$u_{i+\frac{1}{2}}^{n+1} = \nu(3\nu-2)u_{i-\frac{1}{2}}^{n} + 6\nu(1-\nu)\overline{u}_i^n + (1-\nu)(1-3\nu)u_{i+\frac{1}{2}}^{n}. \tag{A.1}$$

The solution at time level n is mapped to the time domain using characteristics, and integrated over a time step to find the flux terms:

$$\overline{f}_{i+\frac{1}{2}} = a\left(\nu(\nu-1)u_{i-\frac{1}{2}}^{n} + \nu(3-2\nu)\overline{u}_i^n + (1-\nu)^2 u_{i+\frac{1}{2}}^{n}\right). \tag{A.2}$$

Finally, the cell averages are updated conservatively:

$$\overline{u}_i^{n+1} = \overline{u}_i^n - \frac{\Delta t}{\Delta x}\left(\overline{f}_{i+\frac{1}{2}} - \overline{f}_{i-\frac{1}{2}}\right). \tag{A.3}$$

Substituting (A.2) into (A.3) results in:

$$\overline{u}_i^{n+1} = \overline{u}_i^n - \frac{a\Delta t}{\Delta x}\left(\nu(\nu-1)\left(u_{i-\frac{1}{2}}^{n} - u_{i-\frac{3}{2}}^{n}\right) + \nu(3-2\nu)\left(\overline{u}_i^n - \overline{u}_{i-1}^n\right) + (1-\nu)^2\left(u_{i+\frac{1}{2}}^{n} - u_{i-\frac{1}{2}}^{n}\right)\right). \tag{A.4}$$

Or:

$$\overline{u}_i^{n+1} - \overline{u}_i^n = -\nu^2(\nu-1)\left(u_{i-\frac{1}{2}}^{n} - u_{i-\frac{3}{2}}^{n}\right) - \nu^2(3-2\nu)\left(\overline{u}_i^n - \overline{u}_{i-1}^n\right) - \nu(1-\nu)^2\left(u_{i+\frac{1}{2}}^{n} - u_{i-\frac{1}{2}}^{n}\right). \tag{A.5}$$

The next step is to eliminate the integrated values from this expression. To do this, first subtract (A.1) from itself, shifted by a spatial interval:

$$u_{i+\frac{1}{2}}^{n+1} - u_{i-\frac{1}{2}}^{n+1} = \nu(3\nu-2)\left(u_{i-\frac{1}{2}}^{n} - u_{i-\frac{3}{2}}^{n}\right) + 6\nu(1-\nu)\left(\overline{u}_i^n - \overline{u}_{i-1}^n\right) + (1-\nu)(1-3\nu)\left(u_{i+\frac{1}{2}}^{n} - u_{i-\frac{1}{2}}^{n}\right). \tag{A.6}$$

Separating integrated values from edge values yields:

$$6\nu(1-\nu)\left(\overline{u}_i^n-\overline{u}_{i-1}^n\right)=u_{i+\frac{1}{2}}^{n+1}-u_{i-\frac{1}{2}}^{n+1}-\nu(3\nu-2)\left(u_{i-\frac{1}{2}}^n-u_{i-\frac{3}{2}}^n\right)-(1-\nu)(1-3\nu)\left(u_{i+\frac{1}{2}}^n-u_{i-\frac{1}{2}}^n\right). \tag{A.7}$$

Now again subtract (A.1) from itself, but shifted by a single time step:

$$u_{i+\frac{1}{2}}^{n+2}-u_{i+\frac{1}{2}}^{n+1}=\nu(3\nu-2)\left(u_{i-\frac{1}{2}}^{n+1}-u_{i-\frac{1}{2}}^n\right)+6\nu(1-\nu)\left(\overline{u}_i^{n+1}-\overline{u}_i^n\right)+(1-\nu)(1-3\nu)\left(u_{i+\frac{1}{2}}^{n+1}-u_{i+\frac{1}{2}}^n\right). \tag{A.8}$$

Again separating integrated values from edge values:

$$6\nu(1-\nu)\left(\overline{u}_i^{n+1}-\overline{u}_i^n\right)=u_{i+\frac{1}{2}}^{n+2}-u_{i+\frac{1}{2}}^{n+1}-\nu(3\nu-2)\left(u_{i-\frac{1}{2}}^{n+1}-u_{i-\frac{1}{2}}^n\right)-(1-\nu)(1-3\nu)\left(u_{i+\frac{1}{2}}^{n+1}-u_{i+\frac{1}{2}}^n\right). \tag{A.9}$$

To prepare the final substitution, multiply (A.5) by $6\nu(1-\nu)$:

$$\begin{aligned}6\nu(1-\nu)\left(\overline{u}_i^{n+1}-\overline{u}_i^n\right)=&6\nu^3(1-\nu)^2\left(u_{i-\frac{1}{2}}^n-u_{i-\frac{3}{2}}^n\right)-6\nu^2(1-\nu)^3\left(u_{i+\frac{1}{2}}^n-u_{i-\frac{1}{2}}^n\right)\\&-\nu^2(3-2\nu)6\nu(1-\nu)\left(\overline{u}_i^n-\overline{u}_{i-1}^n\right).\end{aligned} \tag{A.10}$$

Substitute (A.7) and (A.9) into (A.10):

$$\begin{aligned}u_{i+\frac{1}{2}}^{n+2}-u_{i+\frac{1}{2}}^{n+1}-&\nu(3\nu-2)\left(u_{i-\frac{1}{2}}^{n+1}-u_{i-\frac{1}{2}}^n\right)-(1-\nu)(1-3\nu)\left(u_{i+\frac{1}{2}}^{n+1}-u_{i+\frac{1}{2}}^n\right)=\\&6\nu^3(1-\nu)^2\left(u_{i-\frac{1}{2}}^n-u_{i-\frac{3}{2}}^n\right)-6\nu^2(1-\nu)^3\left(u_{i+\frac{1}{2}}^n-u_{i-\frac{1}{2}}^n\right)\\&-\nu^2(3-2\nu)\left(u_{i+\frac{1}{2}}^{n+1}-u_{i-\frac{1}{2}}^{n+1}-\nu(3\nu-2)\left(u_{i-\frac{1}{2}}^n-u_{i-\frac{3}{2}}^n\right)-(1-\nu)(1-3\nu)\left(u_{i+\frac{1}{2}}^n-u_{i-\frac{1}{2}}^n\right)\right).\end{aligned} \tag{A.11}$$

Rearranging and factorizing:

$$\begin{aligned}u_{i+\frac{1}{2}}^{n+2}=&2(1-\nu)\left(1-\nu-\nu^2\right)u_{i+\frac{1}{2}}^{n+1}\\&+2\nu\left(-1+3\nu-\nu^2\right)u_{i-\frac{1}{2}}^{n+1}\\&+2\nu(1-\nu)(1+\nu-\nu^2)u_{i-\frac{1}{2}}^n\\&-(1-\nu)^4u_{i+\frac{1}{2}}^n\\&-\nu^4u_{i-\frac{3}{2}}^n.\end{aligned} \tag{A.12}$$

Finally a shift in reference is applied such that the new edge value equals u_i^{n+1}:

$$\begin{aligned}u_i^{n+1}=&2(1-\nu)\left(1-\nu-\nu^2\right)u_i^n\\&+2\nu\left(-1+3\nu-\nu^2\right)u_{i-1}^n\\&+2\nu(1-\nu)(1+\nu-\nu^2)u_{i-1}^{n-1}\\&-(1-\nu)^4u_i^{n-1}\\&-\nu^4u_{i-2}^{n-1}.\end{aligned} \tag{A.13}$$

B

Active Flux form derivation

In node form, the update method equals:

$$u_i^{n+1} = -2\nu\frac{1-2\nu}{\nu+1}u_{i-1}^n + 2(1-2\nu)u_i^n + 2\nu u_{i-1}^{n-1} - \frac{(2\nu-1)(\nu-1)}{\nu+1}u_i^{n-1}. \tag{B.1}$$

The goal is to rewrite this to an Active Flux form, where nodes and integral values are updated separately. In the Active Flux form, we want the node update to be formulated as:

$$u_i^{n+1} = u_i^n - \nu\Delta x \left.\frac{\partial u}{\partial x}\right|_i^n. \tag{B.2}$$

To mold (B.1) into this form, we first introduce the additional time level $n-1$ by shifting (B.2) in time and finding the difference between the two:

$$u_i^{n+1} - u_i^n = u_i^n - u_i^{n-1} - \nu\left(\Delta x \left.\frac{\partial u}{\partial x}\right|_i^n - \Delta x \left.\frac{\partial u}{\partial x}\right|_i^{n-1}\right). \tag{B.3}$$

Relocating several terms in (B.1) yields:

$$u_i^{n+1} - u_i^n = u_i^n - u_i^{n-1} - \nu\left(-2\frac{2\nu-1}{\nu+1}u_{i-1}^n + 4u_i^n - 2u_{i-1}^{n-1} + 2\frac{\nu-2}{\nu+1}u_i^{n-1}\right). \tag{B.4}$$

The next step is to express the term in parenthesis in terms of time differences of node values, or:

$$u_i^{n+1} - u_i^n = u_i^n - u_i^{n-1} - \nu\left(c_1\left(u_{i-1}^n - u_{i-1}^{n-1}\right) + c_2\left(u_i^n - u_i^{n-1}\right) + c_3\left(u_i^n - u_{i-1}^n\right) + c_4\left(u_i^{n-1} - u_{i-1}^{n-1}\right)\right). \tag{B.5}$$

The coefficients c_{1-4} are found through solving the following system:

$$\begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 1 & 0 \\ -1 & 0 & 0 & -1 \\ 0 & -1 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix} = \begin{bmatrix} -2\frac{2\nu-1}{\nu+1} \\ 4 \\ -2 \\ 2\frac{\nu-2}{\nu+1} \end{bmatrix}, \tag{B.6}$$

resulting in:

$$\begin{aligned} c_1 &= 2, \\ c_2 &= -2\frac{\nu-2}{\nu+1}, \\ c_3 &= \frac{6\nu}{\nu+1}, \\ c_4 &= 0. \end{aligned}$$

We end up with:

$$u_i^{n+1} - u_i^n = u_i^n - u_i^{n-1} - \nu\left(2\left(u_{i-1}^n - u_{i-1}^{n-1}\right) - 2\frac{\nu-2}{\nu+1}\left(u_i^n - u_i^{n-1}\right) + \frac{6\nu}{\nu+1}\left(u_i^n - u_{i-1}^n\right)\right), \tag{B.7}$$

where only the last term needs to be converted from a spatial difference to a temporal difference. This is resolved by introducing the average integral value, $\overline{u}$. The average integral value advances in time by the summation of flux contributions on its enclosing boundaries. In one dimension this equals:

$$\overline{u}_{i-\frac{1}{2}}^n = \overline{u}_{i-\frac{1}{2}}^{n-1} - \frac{1}{\Delta x}\int_{n-1}^{n} f_i - f_{i-1}\mathrm{d}t. \tag{B.8}$$

For linear advection, the flux equals $f(t) = au(t)$. If the flux integral is estimated by $\int_{n-1}^{n} f_i \mathrm{d}t \approx au_i^n \Delta t$, (B.8) becomes:

$$\begin{aligned}\overline{u}_{i-\frac{1}{2}}^n =& \overline{u}_{i-\frac{1}{2}}^{n-1} - a\frac{\Delta t}{\Delta x}\left(u_i^n - u_{i-1}^n\right)\\ =& \overline{u}_{i-\frac{1}{2}}^{n-1} - \nu\left(u_i^n - u_{i-1}^n\right).\end{aligned} \tag{B.9}$$

This equation thus can be used to map a spatial node undivided difference to a temporal integral value undivided difference, which is exactly what is required to remove the spatial difference from (B.7). The result is then:

$$u_i^{n+1} - u_i^n = u_i^n - u_i^{n-1} - \nu\left(2\left(u_{i-1}^n - u_{i-1}^{n-1}\right) - 2\frac{\nu-2}{\nu+1}\left(u_i^n - u_i^{n-1}\right) - \frac{6}{\nu+1}\left(\overline{u}_{i-\frac{1}{2}}^n - \overline{u}_{i-\frac{1}{2}}^{n-1}\right)\right). \tag{B.10}$$

This expression reveals that in Active Flux form, the nodes will be updated by:

$$\begin{aligned}u_i^{n+1} &= u_i^n - \nu\left(2u_{i-1}^n - 2\frac{\nu-2}{\nu+1}u_i^n - \frac{6}{\nu+1}\overline{u}_{i-\frac{1}{2}}^n\right)\\ &= u_i^n - 2\nu\left(\frac{u_{i-1}^n - 3\overline{u}_{i-\frac{1}{2}}^n + 2u_i^n}{\nu+1} + \nu\frac{u_{i-1}^n - u_i^n}{\nu+1}\right).\end{aligned} \tag{B.11}$$

Finally, a spatial shift puts the update methods in the same framework as used in the original Active Flux formulation:

$$u_{i+\frac{1}{2}}^{n+1} = u_{i+\frac{1}{2}}^n - 2\nu\left(\frac{u_{i-\frac{1}{2}}^n - 3\overline{u}_i^n + 2u_{i+\frac{1}{2}}^n}{\nu+1} + \nu\frac{u_{i-\frac{1}{2}}^n - u_{i+\frac{1}{2}}^n}{\nu+1}\right) \qquad \text{(Node update).} \tag{B.12}$$

$$\overline{u}_i^{n+1} = \overline{u}_i^n - \nu\left(u_{i+\frac{1}{2}}^{n+1} - u_{i-\frac{1}{2}}^{n+1}\right) \qquad \text{(Integral update).} \tag{B.13}$$

C

Square wave results

As explained in Chapter 6, the results for the square wave were found to be very similar as those found for the sawtooth wave. For this reason, the square wave results are solely shown here. From a numerical point of view, the extra discontinuity in a square wave is the only aspect that distinguishes it from a sawtooth wave. Because of the constant distance between the discontinuities, there is only very little interaction in terms of mesh structure. This decreases further with increasing maximum refinement level. Therefore, in terms of mesh structure, there should be no radical difference between an adaptive sawtooth wave simulation of maximum refinement depth D and an adaptive square wave simulation of maximum refinement depth $D+1$. For consistency, most simulations for the square wave are still run at a maximum depth of 7. Mesh deterioration for increasing Φ is clearly shown in Table C.1.

Table C.1: Mesh visualizations at different time steps and varying Φ for a square waveform, at a maximum refinement depth of 7. Legend at the bottom indicates refinement depth.

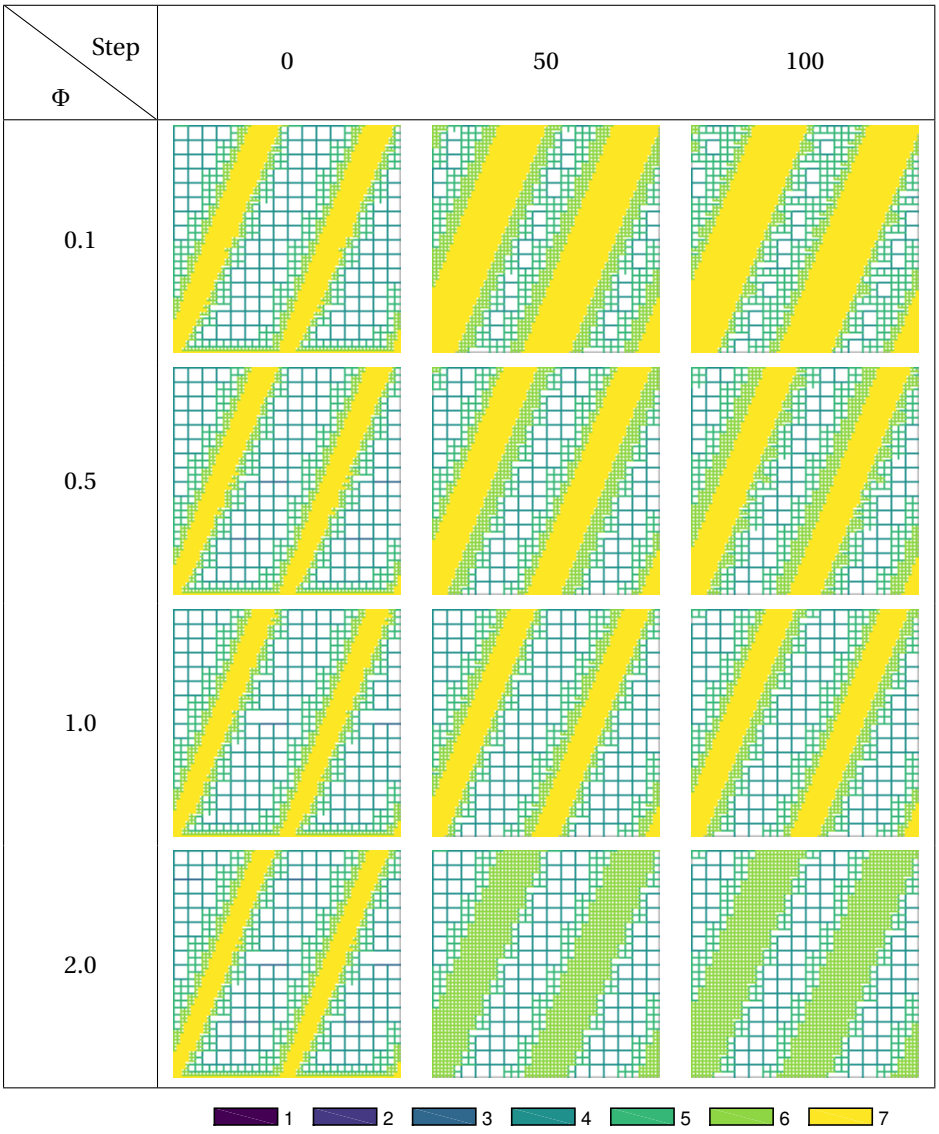


Figure C.1 visualizes how for Φ = 0.1 and Φ = 2.0 the number of cells shows diverging trends, where the cell count appears more stable for Φ = 0.5 and Φ = 1.0.

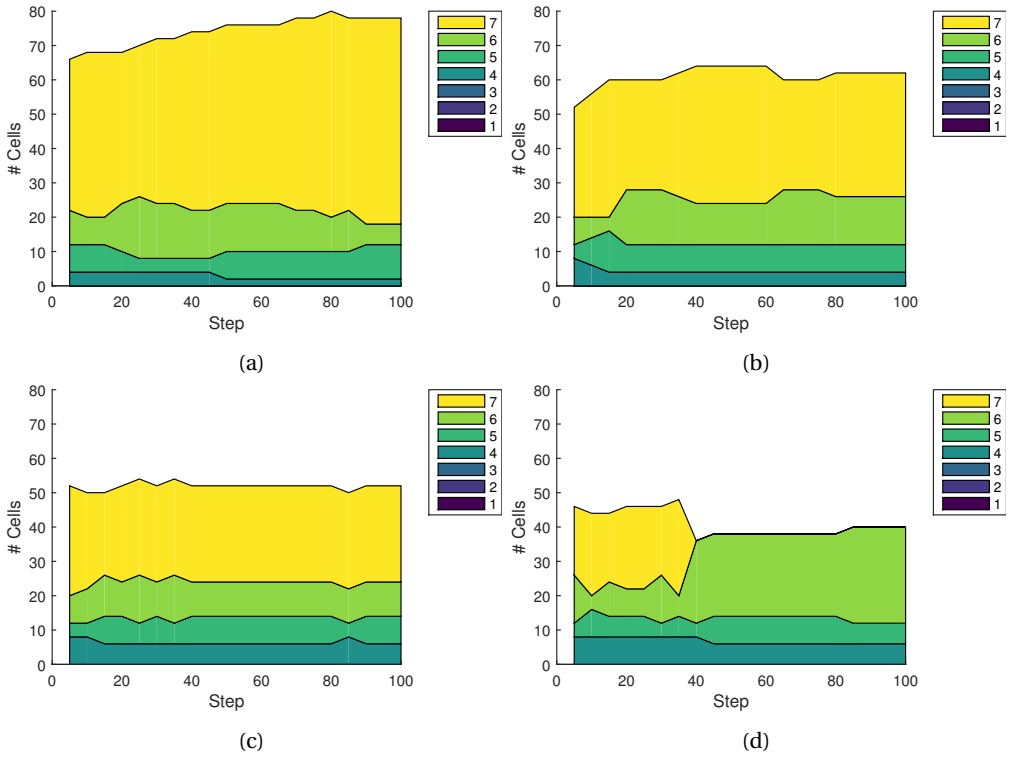


Figure C.1: The cell count per depth level during the simulation of a square wave for 100 global time steps. Legend entries indicate depth level. The first step is excluded as the simulation starts with a uniform mesh. (a) $\Phi = 0.1$, (b) $\Phi = 0.5$, (c) $\Phi = 1.0$, (d) $\Phi = 2.0$.

Next, Figure C.2 shows how the L^2 error develops over time. The same trends as found for the sawtooth wave are displayed, differing by an expected factor of two.

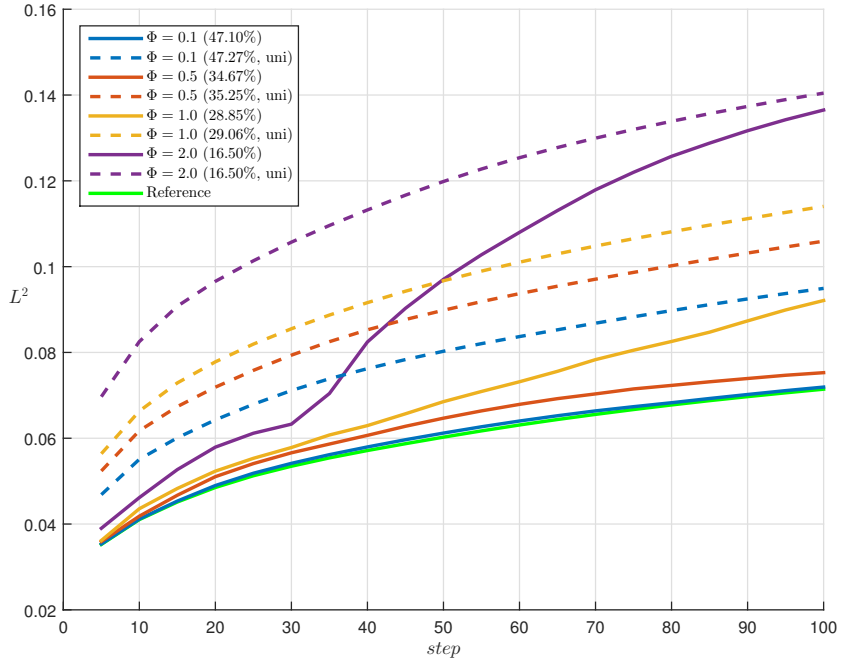


Figure C.2: L^2 error development over time for a square wave. Maximum refinement depth is set to 7. The green line indicates the uniform maximum refinement reference simulation. The dashed lines indicate the uniform mesh simulations with approximately equal cell load.

Nearly equal trends are also observed for the cell load and L^2 error as function of the maximum refinement depth, see respectively Figures C.3 and C.4.

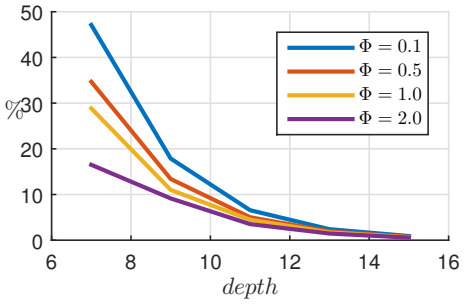


Figure C.3: Cell load for a square wave as a function of maximum refinement depth, for varying Φ.

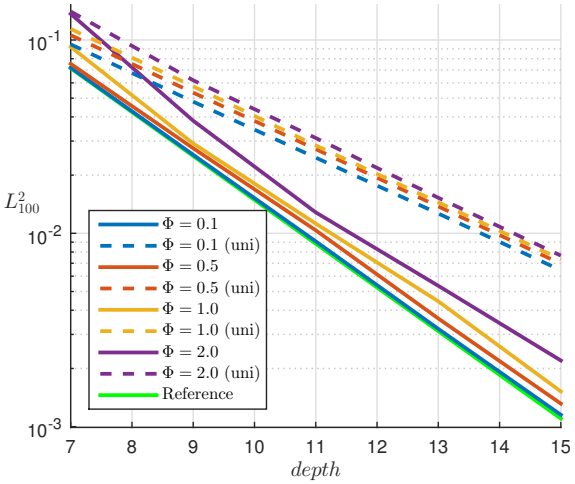


Figure C.4: L^2 error after 100 global time steps for a square wave as a function of maximum refinement depth, for varying Φ.

Bibliography

[1] Timothy J. Baker. Mesh adaptation strategies for problems in fluid dynamics. *Finite Elements in Analysis and Design*, 25(3):243 – 273, 1997. ISSN 0168-874X. doi: https://doi.org/10.1016/S0168-874X(96)00032-7. URL http://www.sciencedirect.com/science/article/pii/S0168874X96000327. Adaptive Meshing, Part 2.

[2] D. Balsara. Adaptive Mesh Refinement in Computational Astrophysics – Methods and Applications. *Journal of Korean Astronomical Society*, 34:181–190, December 2001. doi: 10.5303/JKAS.2001.34.4.181.

[3] Marsha J Berger and Joseph Oliger. Adaptive mesh refinement for hyperbolic partial differential equations. *Journal of Computational Physics*, 53(3):484 – 512, 1984. ISSN 0021-9991. doi: https://doi.org/10.1016/0021-9991(84)90073-1. URL http://www.sciencedirect.com/science/article/pii/0021999184900731.

[4] M.J. Berger and P. Colella. Local adaptive mesh refinement for shock hydrodynamics. *Journal of Computational Physics*, 82(1):64 – 84, 1989. ISSN 0021-9991. doi: https://doi.org/10.1016/0021-9991(89)90035-1. URL http://www.sciencedirect.com/science/article/pii/0021999189900351.

[5] M.W. Bern, J.E. Flaherty, and M. Luskin. *Grid Generation and Adaptive Algorithms*. The IMA Volumes in Mathematics and its Applications. Springer New York, 2012. ISBN 9781461215561. URL https://books.google.nl/books?id=V_TUBwAAQBAJ.

[6] A. Harten, B. Engquist, S. Osher, S. Chakravarthy. Uniformly high order accurate essentially non-oscillatory schemes, iii. *Journal of Computational Physics*, Vol. 131, 1987.

[7] S. Chang. The method of space-time conservation element and solution element - a new approach for solving the navier-stokes and euler equations. *Journal of Computational Physics*, Vol. 119, 1995.

[8] Ami Harten. High resolution schemes for hyperbolic conservation laws. *Journal of Computational Physics*, 49(3):357 – 393, 1983. ISSN 0021-9991. doi: https://doi.org/10.1016/0021-9991(83)90136-5. URL http://www.sciencedirect.com/science/article/pii/0021999183901365.

[9] J.Patrick Jessee, Woodrow A. Fiveland, Louis H. Howell, Phillip Colella, and Richard B. Pember. An adaptive mesh refinement algorithm for the radiative transport equation. *Journal of Computational Physics*, 139(2):380 – 398, 1998. ISSN 0021-9991. doi: https://doi.org/10.1006/jcph.1997.5870. URL http://www.sciencedirect.com/science/article/pii/S0021999197958708.

[10] A. M. Khokhlov. Fully threaded tree algorithms for adaptive refinement fluid dynamics simulations. *Journal of Computational Physics*, 143(2):519 – 543, 1998. ISSN 0021-9991. doi: https://doi.org/10.1006/jcph.1998.9998. URL http://www.sciencedirect.com/science/article/pii/S0021999198999983.

[11] H. Nishikawa, K. Kitamura. Very simple, carbuncle-free, boundary-layer-resolving, rotated-hybrid riemann solvers. *Journal of Computational Physics*, Vol. 227, 2008.

[12] Richard I. Klein. Star formation with 3-d adaptive mesh refinement: the collapse and fragmentation of molecular clouds. *Journal of Computational and Applied Mathematics*, 109(1):123 – 152, 1999. ISSN 0377-0427. doi: https://doi.org/10.1016/S0377-0427(99)00156-9. URL http://www.sciencedirect.com/science/article/pii/S0377042799001569.

[13] A.J. Kriel. Error analysis of flux limiter schemes at extrema. *Journal of Computational Physics*, 328:371 – 386, 2017. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2016.10.024. URL http://www.sciencedirect.com/science/article/pii/S0021999116305228.

[14] C.B. Laney. *Computational Gasdynamics*. Birkhäuser, 1998. ISBN 9783764324643.

[15] R.J. LeVeque. *Numerical methods for conservation laws*. Cambridge University Press, 1992. ISBN 9780521625586.

[16] Hong-Chia Lin. Dissipation additions to flux-difference splitting. *Journal of Computational Physics*, 117(1):20 – 27, 1995. ISSN 0021-9991. doi: https://doi.org/10.1006/jcph.1995.1040.

[17] D.Scott McRae. r-refinement grid adaptation algorithms and issues. *Computer Methods in Applied Mechanics and Engineering*, 189(4):1161 – 1182, 2000. ISSN 0045-7825. doi: https://doi.org/10.1016/S0045-7825(99)00372-2. URL http://www.sciencedirect.com/science/article/pii/S0045782599003722. Adaptive Methods for Compressible CFD.

[18] Tomoya Ogawa, Kazuyuki Yamashita, Takuma Ohta, Ryoji Matsumoto, and Mitsue Den. Hydrodynamical simulations using a fully threaded tree. *Advances in Space Research*, 30(3):561 – 563, 2002. ISSN 0273-1177. doi: https://doi.org/10.1016/S0273-1177(02)00343-5.

[19] S. Chakravarthy, S. Osher. High resolution applications of the osher upwind scheme for the euler equations. *AIAA-83-1943*, 1983.

[20] T. Plewa and E. Müller. Amra: An adaptive mesh refinement hydrodynamic code for astrophysics. *Computer Physics Communications*, 138(2):101 – 127, 2001. ISSN 0010-4655. doi: https://doi.org/10.1016/S0010-4655(01)00199-0.

[21] Kévin Pons and Mehmet Ersoy. Adaptive mesh refinement method. part 1: Automatic thresholding based on a distribution function. 2016.

[22] J.J. Quirk. A contribution to the great riemann solver debate. *International Journal for Numerical Methods in Fluids*, Vol. 18(no. 6), 1994. ISSN 1097-0363. doi: 10.1002/fld.1650180603. URL http://dx.doi.org/10.1002/fld.1650180603.

[23] P.L. Roe and Royal Aircraft Establishment. *Numerical Algorithms for the Linear Wave Equation*. Technical report: Royal Aircraft Establishment. Royal Aircraft Establishment, 1981.

[24] T.A. Eymann, P. Roe. Active flux schemes for systems. 2011.

[25] J. A. Schmidt, C. R. Johnson, J. C. Eason, and R. S. Macleod. *Applications of Automatic Mesh Generation and Adaptive Methods in Computational Medicine*, pages 367–393. Springer New York, New York, NY, 1995. ISBN 978-1-4612-4248-2. doi: 10.1007/978-1-4612-4248-2_18. URL https://doi.org/10.1007/978-1-4612-4248-2_18.

[26] Sung soo Kim, Chongam Kim, Oh-Hyun Rho, and Seung Kyu Hong. Cures for the shock instability: Development of a shock-stable roe scheme. *Journal of Computational Physics*, 185(2):342 – 374, 2003. ISSN 0021-9991. doi: https://doi.org/10.1016/S0021-9991(02)00037-2.

[27] E.F. Toro. *Riemann Solvers and Numerical Methods for Fluid Dynamics: A Practical Introduction*. Springer Berlin Heidelberg, 2009. ISBN 9783540498346.

[28] B. van der Holst and R. Keppens. Hybrid block-amr in cartesian and curvilinear coordinates: Mhd applications. *Journal of Computational Physics*, 226(1):925 – 946, 2007. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2007.05.007. URL http://www.sciencedirect.com/science/article/pii/S002199910700215X.

[29] B. van Leer. Towards the ultimate conservative difference scheme, II: Monotonicity and conservation combined in a second order scheme. *Journal of Computational Physics*, Vol. 14, 1974.

[30] R. Verfürth. *A review of a posteriori error estimation and adaptive mesh-refinement techniques*. Advances in numerical mathematics. Wiley-Teubner, 1996. ISBN 9780471967958. URL https://books.google.nl/books?id=DtRUAAAAYAAJ.

[31] T. Chan X. Liu, S. Osher. Weighted essentially non-oscillatory schemes. *Journal of Computational Physics*, Vol. 115, 1994.

[32] Kun Xu and Zuowu Li. Dissipative mechanism in godunov-type schemes. *International Journal for Numerical Methods in Fluids*, 37:1 – 22, 09 2001.

[33] Z. Shen, W. Yan, G. Yuan. A robust HLLC-type riemann solver for strong shock. *Journal of Computational Physics*, Vol. 309, 2016.