Centrum voor Wiskunde en Informatica

**REPORT***RAPPORT*

MAS

Modelling, Analysis and Simulation

*Modelling, Analysis and Simulation*

# An adaptive-gridding solution method for the 2D unsteady Euler equations

J. Wackers

**NOTE MAS-N0301 JANUARY 31, 2003**

P.O. Box 94079, 1090 GB Amsterdam (NL)
Kruislaan 413, 1098 SJ Amsterdam (NL)
Telephone +31 20 592 9333
Telefax +31 20 592 4199

ISSN 1386-3703

# An Adaptive-Gridding Solution Method for the 2D Unsteady Euler Equations

J. Wackers
*CWI, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands*
*and*
*Delft University of Technology, Faculty of Aerospace Engineering*
*P.O. Box 5058, 2600 GB Delft, The Netherlands*

January 2003

## ABSTRACT

Adaptive grid refinement is a technique to speed up the numerical solution of partial differential equations by starting these calculations on a coarse basic grid and refining this grid only there where the solution requires this, e.g. in areas with large gradients. This technique has already been used often, for both steady and unsteady problems. Here, a simple and efficient adaptive grid technique is proposed for the solution of systems of 2D unsteady hyperbolic conservation laws. The technique is applied to the Euler equations of gasdynamics. Extension to other conservation laws or to 3D is expected to be straightforward.

A solution algorithm is presented that refines a rectangular basic grid by splitting coarse cells into four, as often as required, and merging these cells again afterwards. The small cells have a shorter time step too, so the grid is refined in space and time. The grid is adapted to the solution several times per coarse time step, therefore the total number of cells is kept low and a fast solution is ensured.

The grid is stored in a simple data structure. All grid data are stored in 1D arrays and the grid geometry is determined with, per cell, five pointers to other cells: one 'mother' pointer to the cell from which the cell was split and four 'neighbour' pointers. The latter are arranged so, that all cells around the considered cell can be quickly found.

To determine where the grid is refined, a refinement criterion is used. Three different refinement criteria are studied: one based on the first spatial derivative of the density, one on the second spatial derivative of the density and one on an estimate of the local truncation error, comparable to Richardson extrapolation. Especially the first-derivative $\rho$ criterion gives good results.

The algorithm is combined with a simple first-order accurate discretisation of the Euler equations, based on Osher's flux function, and tested.

A second-order accurate discretisation of the Euler equations is presented that combines a second-order limited discretisation of the fluxes with the time derivatives of the Richtmyer scheme. This scheme can be easily combined with the adaptive-gridding algorithm. Stability is proved for CFL numbers below 0.25. For cells with different sizes, several interpolation techniques are developed, like the use of virtual cells for flux calculation.

The scheme is tested with two standard test cases, the 1D Sod problem and the forward-facing step problem, known from the work of Woodward and Colella. The results show that the second-order scheme is more efficient than the first-order scheme. An accuracy, comparable with solutions on uniform grids is obtained, but with at least five times lower computational costs. Results from a last test problem, the shedding of vortices from a flat plate that is suddenly set into motion, confirm that the method can be used for different flow regimes and that it is very useful in practice for analysis of unsteady flow.

*2000 Mathematics Subject Classification:* 65M50, 65M60, 76N15
*Keywords and Phrases:* adaptive grid refinement, unsteady problems, hyperbolic systems, Euler equations, second-order accuracy, data structure, refinement criteria.
*Note:* This research was carried out under CWI-project MAS2.1 "Computational Fluid Dynamics".

# Acknowledgements

This work is a reprint of a Master's thesis at Delft University of Technology, Faculty of Aerospace Engineering, Chair of Aerodynamics. It was performed in part at the Centre for Mathematics and Computer Science (CWI) in Amsterdam.

This thesis does not stand on its own. It is a stand-alone project, but it is rooted in a tradition of CFD code development at CWI, that has led to a consistent way of developing solvers for fluid flow. I am proud to contribute to this tradition.

I want to thank the people at the Faculty of Aerospace Engineering. I haven't shown up there often this year... but without the four years of education they gave me, this work wouldn't have been possible.

Furthermore, I want to thank the people at CWI, especially my supervisor prof.dr.ir. B. Koren. He suggested a large number of interesting projects for me but still allowed me to do my own idea, he thought about the project together with me and looked at my work very critically indeed. And sometimes, when I got stuck in some difficult problem, he immediately came with a solution that was both effective and simple.

I am thankful to ir. M.R. Lewis. For me, he gave up the privilege of a room for himself... and he often listened to what I was doing and gave helpful advice. I am thankful as well to the others who helped me with advice, like prof.dr. P.W. Hemker and prof.dr. B. van Leer.

Finally, I am especially grateful to all the people at CWI, for creating a nice place to work, where inspiring research is done and where the world outside that research is not neglected. A place where I could compensate for the hours spent alone at my desk at home. Without them, I could never have done this.

Den Hoorn, January 2003,

Jeroen Wackers

# Table of Contents

# List of symbols

## Symbols

| | |
|---|---|
| $A$ | jacobian matrix, quasilinear Euler equations |
| $A^-$ | negative-eigenvalue part of $A$ |
| $C_{\partial\rho}$ | gradient $\rho$ refinement criterion |
| $C_{\partial\rho u}$ | gradient $\rho u$ refinement criterion |
| $C_{\partial^2\rho}$ | second derivative $\rho$ refinement criterion |
| $C_{EE}$ | error estimation refinement criterion |
| $C_p$ | pressure coefficient |
| $c$ | speed of sound |
| $d_m$ | lower limit for refinement criterion (‘merge’) |
| $d_s$ | upper limit for refinement criterion (‘split’) |
| $d_{s\Delta}$ | increase limit, $C_{EE}$ criterion |
| $E$ | total energy |
| $e$ | error in solution |
| $\boldsymbol{F}_O$ | approximate flux vector from Osher’s scheme |
| $\boldsymbol{f}$ | flux vector, horizontal |
| $\boldsymbol{G}_O$ | approximate flux vector from Osher’s scheme |
| $\boldsymbol{g}$ | flux vector, vertical |
| $H$ | total enthalpy |
| $h$ | $\Delta x$ in 1D cartesian grid |
| $L$ | Euler operator, discrete formulation |
| $\bar{L}$ | Euler operator, continuous formulation |
| $l$ | level of refinement |
| $M$ | maximum level of refinement |
| $M$ | Mach number |
| $N$ | total number of cells |
| $P$ | prolongation operator |
| $p$ | pressure |
| $\boldsymbol{q}$ | state vector |
| $\bar{\boldsymbol{q}}$ | state vector, continuous solution |
| $\boldsymbol{q}_0$ | initial state |
| $q$ | state vector component |

| | |
|---|---|
| $R$ | restriction operator |
| $r$ | measure for the curvature of the solution |
| $t$ | time |
| $u$ | velocity component in $x$-direction |
| $v$ | velocity component in $y$-direction |
| $x$ | spatial coordinate |
| $y$ | spatial coordinate |
| $z$ | unscaled entropy |

| | |
|---|---|
| $\gamma$ | ratio of specific heats |
| $\Delta t$ | time step |
| $\Delta u$ | velocity change over a wave |
| $\Delta x$ | cell size, $x$-direction |
| $\Delta y$ | cell size, $y$-direction |
| $\zeta$ | amplification factor, Von Neumann analysis |
| $\lambda$ | eigenvalue of the Euler equations |
| $\lvert\lambda\rvert_{max}$ | largest $\lambda$ (in absolute value) in a flow problem |
| $\rho$ | density |
| $\tau$ | truncation error |
| $\phi$ | limiter function |
| $\phi_{MM}$ | minmod limiter |
| $\phi_{MVA}$ | modified Van Albada limiter |
| $\psi$ | Riemann invariant, Osher scheme |
| $\Omega$ | spatial domain, cell or collection of cells |
| $\partial\Omega$ | boundary of $\Omega$ |
| $\Omega_*^0$ | basic coarse grid |
| $\omega$ | error frequency, Von Neumann analysis |

## Subscripts

| | |
|---|---|
| $\infty$ | undisturbed conditions |
| $0$ | left state, Osher scheme |
| $1$ | right state, Osher scheme |
| $1/3$ | left intermediate state, Osher scheme |
| $1/2$ | left / right intermediate state, Osher scheme |
| $2/3$ | right intermediate state, Osher scheme |
| $a$ | above (value of $n$) |

| | |
|---|---|
| $B$ | boundary |
| $b$ | below (value of $n$) |
| $d$ | diagonal ($m$, $dr$, $dd$ or $da$) |
| $d_i$ | diagonal cell $d$ of cell $i$ |
| $da$ | above left (value of $d$) or daughter cell above |
| $dd$ | above right (value of $d$) or daughter cell diagonal |
| $dr$ | below right (value of $d$) or daughter cell right |
| $i$ | cell number or first index of cartesian cell number |
| $i, d$ | virtual state $d$ of cell $i$ |
| $i, n$ | cell face $n$ of cell $i$ |
| $j$ | second index of cartesian cell number |
| $L$ | left cell face |
| $l$ | left (value of $n$) |
| $m$ | below left (value of $d$) or mother cell |
| $n$ | neighbour ($b$, $r$, $a$ or $l$) |
| $n_i$ | neighbour cell $n$ of cell $i$ |
| $n + 2$ | opposite side of $n$ |
| $p$ | component of state vector |
| $R$ | right cell face |
| $r$ | right (value of $n$) |
| $s$ | sonic point |

## SUPERSCRIPTS

| | |
|---|---|
| $k$ | time step number |
| $l$ | level of refinement |
| $V$ | virtual cell, virtual state |

# Chapter 1
# Introduction

In present-day aerodynamics, the solution of the Euler equations for realistic geometries has become possible. Due to the increase in calculation speed for computers and the introduction of efficient solvers, this solution process is now fast enough to enable the use of Euler solvers as a design tool for aircraft etc. But as the Euler solvers become faster, aerodynamic designers will certainly use this increased capacity to stretch the area of their application to ever more complex geometries and more accurate solutions, requiring still faster solvers. Therefore, the development of more efficient methods to solve the Euler equations is still of paramount importance, as, together with the increase in performance of computers, it is the only way to provide the designer of the future with fast enough design tools.

One way of increasing the efficiency of Euler calculations is to use very accurate discretisations of the equations. The first Euler solvers were all first-order accurate [5, 6], but later solvers used ever more complex higher-order discretisations. One disadvantage of these methods is the high complexity of the code, another is sometimes called the ‘conservation of difficulty’ problem [24]: accurate methods require many calculations per cell, so simpler methods on finer grids may give a comparable accuracy at the same computational costs.

A completely different approach is the use of adaptive grid refinement. This approach is based on the realization that, for uniform grids, the grid size needed to solve the entire problem with a given accuracy is determined by a few areas in the flow with e.g. large gradients. The other areas may still have the required accuracy when they are solved on coarser grids. So, by calculating the solution for these last areas on unnecessary fine grids, computation time is wasted. The idea behind adaptive gridding is to reduce this waste by solving only the areas that really need it on fine grids and the rest of the flow domain on coarser grids. This allows the grid to be chosen very fine locally, so that a good accuracy can be obtained with a relatively simple discretisation of the equations. And although the grid refinement adds complexity to the code, the result may be less complex than a code with a very advanced discretisation of the equations.

Much work has already been done on the solution of the Euler equations with adaptive gridding, for both steady and unsteady problems and with different types of grid refinement. The most important of these are grid movement, for which the grid is distorted to fit the solution, and grid enrichment, for which small cells are added to a basic coarse grid. In the Netherlands, research at CWI has led to a code that combines a simple and robust discretisation of the steady Euler equations with adaptive gridding through enrichment [13]. For the unsteady Euler equations, pioneering work has been done by Berger e.a. [2, 3]. Many others have followed, including the author [21].

The present work concentrates on developing an adaptive-gridding solution technique for the 2D unsteady Euler equations, based on grid enrichment. The goal is to find an adaptive-gridding method that offers an accuracy comparable to second-order methods on uniform grids, but at substantially reduced computational costs. The method must be simple and robust, easily adaptable to different flow problems, and it must be easy to convert the method for the solution of other conservation laws. Therefore, a modular structure is adopted. The solution algorithm, the data structure used to store the grid geometry and the solution, the discretisation of the flow equations and the refinement criteria, used to determine where the grid should be refined, are developed separately to allow individual use and simple modification of each component.

In the first part of this report, an adaptive-gridding algorithm is developed and tested, using a simple first-order accurate discretisation. Chapter 2 introduces the Euler equations and the principle of finite-volume discretisation. The first-order discretisation to be used is discussed. In the next chapter, 3, the adaptive-gridding method is introduced. The algorithm, the data structure and the implementation of the discretisation are treated. Chapter 4 describes the function of refinement criteria and derives three such criteria. Finally, in chapter 5 the method is tested on two standard test problems and the refinement criteria are compared.

An important question is, whether it is worthwhile to use a second-order accurate discretisation together with the adaptive-gridding algorithm, as a better accuracy can also be obtained by keeping the first-order discretisation and refining the grid further. This is investigated in the second part of this report. A second-order accurate discretisation of the Euler equations, based on previous work at CWI [13, 16] on steady Euler problems, is given in chapter 6. This discretisation is adapted to the existing adaptive-gridding algorithm in chapter 7. Finally, results obtained with this second-order solver are given in chapter 8 and compared with the first-order results from chapter 5.

In a last test, the method is used to solve a problem that resembles the problems encountered in practical flow analysis: the shedding of vorticity from a flat plate that is suddenly set into motion. In chapter 9, the practical application of the solver is analysed with this test problem. The report ends with a conclusion and some suggestions for the further development of the adaptive-gridding solver.

# Chapter 2
# Discretisation of the Euler equations

In the following chapters, an adaptive-gridding algorithm is described. To solve the Euler equations with such an algorithm, a discretisation of these equations is needed: they must be written in an approximate form that depends on a limited number of unknowns, the average states in the grid cells. The basic part of this discretisation does not depend on the adaptive-gridding procedure and is described in this chapter. The first section gives the continuous Euler equations, section 2 describes their discretised form, section 3 briefly explains Osher's flux function and the last section describes the implementation of boundary conditions.

## 2.1. The Euler equations

The Euler equations of gas dynamics are derived from the more general Navier Stokes equations by neglecting friction and heat conduction. The resulting equations form a model for fluid flow that is reasonably accurate outside regions with strong velocity gradients (boundary layers, etc.). When discontinuities are allowed in the solution of the Euler equations, shocks can be modelled. They have an infinitesimally small thickness.

The unsteady Euler equations in two dimensions are a system of four conservation laws. These are expressions for the physical principles of conservation of mass, momentum in two directions and energy. Each equation states that the increase of the conserved quantity in a volume for a certain time step is equal to the net inflow of that quantity during the time step.

The equations are formulated as an initial / boundary value problem on a spatial area $\Omega \in \mathbb{R}^2$ with boundary $\partial\Omega$, and for a time interval $0 \leq t \leq T$. Their conservation form can be written as

$$\begin{aligned} \frac{\partial}{\partial t}\boldsymbol{q} + \frac{\partial}{\partial x}\boldsymbol{f}\left(\boldsymbol{q}\right) + \frac{\partial}{\partial y}\boldsymbol{g}\left(\boldsymbol{q}\right) &= 0 \qquad && x, y \in \Omega, \quad 0 \leq t \leq T, \\ B\left(\boldsymbol{q}\right) &= 0 && \text{on } \partial\Omega, \\ \boldsymbol{q}\left(x, y, 0\right) &= \boldsymbol{q}_0\left(x, y\right) && x, y \in \Omega. \end{aligned} \tag{2.1}$$

The boundary operator $B$ depends on the physical boundary conditions. Equation (2.1) consists of four equations, so the state vector $\boldsymbol{q}$ and the flux vectors $\boldsymbol{f}$ and $\boldsymbol{g}$ have four components. They are given by

$$\boldsymbol{q} = \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ \rho E \end{pmatrix} \qquad \boldsymbol{f} = \begin{pmatrix} \rho u \\ p + \rho u^2 \\ \rho u v \\ \rho u H \end{pmatrix} \qquad \boldsymbol{g} = \begin{pmatrix} \rho v \\ \rho u v \\ p + \rho v^2 \\ \rho v H \end{pmatrix}. \tag{2.2}$$

Herein $\rho$ is the fluid density, $p$ the pressure and $u$ and $v$ the speed in $x$- and $y$-direction. The total energy $E$ is given by

$$E = \frac{1}{\gamma - 1}\frac{p}{\rho} + \frac{1}{2}\left(u^2 + v^2\right) \tag{2.3}$$

and the total enthalpy $H$ by

$$H = E + \frac{p}{\rho}, \tag{2.4}$$

with $\gamma$ the ratio of specific heats.

## 2.2. Finite-volume discretisation

To calculate a numerical solution of the Euler equations, they must be discretised: the continuous equations (2.1) must be transformed to a system of equations with a finite number of unknowns, instead of the continuous state $\boldsymbol{q}$. In the finite-volume approach, this is done by dividing the domain $\Omega$ in $N$ small parts, the *cells* $\Omega_i$. In each cell $\Omega_i$, an average state $\boldsymbol{q}_i$ is defined. In this report, only rectangular cells are used.

When all cells are of the same size, each interior cell is surrounded by four neighbour cells: the cells below, right, above and left of it. These cells are indicated as $\Omega_{n_i}$, $n \in \{b(\text{elow}), r(\text{ight}), a(\text{bove}), l(\text{eft})\}$, see figure 2.1.

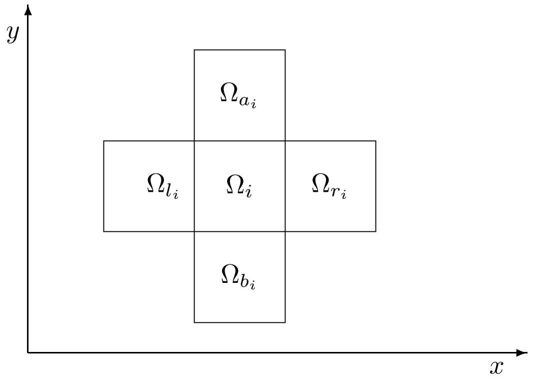


Figure 2.1: A cell $\Omega_i$ and its four neighbours.

Now take equation (2.1) and integrate it over a single cell $\Omega_i$. Using Gauss' theorem, this yields

$$\iint\limits_{\Omega_i} \frac{\partial \boldsymbol{q}}{\partial t}\, dx\, dy + \oint_{\partial\Omega_i} \boldsymbol{f}\, dy + \oint_{\partial\Omega_i} \boldsymbol{g}\, dx = 0. \tag{2.5}$$

For a cell with a size that is independent of time, the integration and time-differentiation may be exchanged. So, for a rectangular cell with size $\Delta x \times \Delta y$ and faces labelled $b, r, a, l$ as well, equation (2.5) is written as

$$\frac{\partial}{\partial t} \iint\limits_{\Omega_i} \boldsymbol{q}\, dx\, dy - \int_{\partial\Omega_{i,b}} \boldsymbol{g}\, dx + \int_{\partial\Omega_{i,r}} \boldsymbol{f}\, dy + \int_{\partial\Omega_{i,a}} \boldsymbol{g}\, dx - \int_{\partial\Omega_{i,l}} \boldsymbol{f}\, dy = 0. \tag{2.6}$$

Define an average cell state $\boldsymbol{q}_i$ and four average fluxes $\boldsymbol{f}_{i,n}$ and $\boldsymbol{g}_{i,n}$, then the equation becomes

$$\frac{\partial}{\partial t} \boldsymbol{q}_i\, \Delta x\, \Delta y + (\boldsymbol{f}_{i,r} - \boldsymbol{f}_{i,l})\ \Delta y + (\boldsymbol{g}_{i,a} - \boldsymbol{g}_{i,b})\ \Delta x = 0, \tag{2.7}$$

with

$$\boldsymbol{q}_i = \frac{1}{\Delta x\, \Delta y} \iint\limits_{\Omega_i} \boldsymbol{q}\, dx\, dy \qquad \boldsymbol{f}_{i,n} = \frac{1}{\Delta y} \int_{\partial\Omega_{i,n}} \boldsymbol{f}\, dy \qquad \boldsymbol{g}_{i,n} = \frac{1}{\Delta x} \int_{\partial\Omega_{i,n}} \boldsymbol{g}\, dx. \tag{2.8}$$

This equation, that is still exact, is the basis for a finite-volume discretisation (see figure 2.2).

Equation (2.7) is a conservation equation for cell $\Omega_i$. It says that the rate of change of the state $\boldsymbol{q}_i$ is equal to the net inflow over the cell faces. The fluxes over the cell faces have another important aspect: the flow out of a cell over one cell face must be equal to the inflow in the next cell over the same face, otherwise the state vector is not conserved. This is important later on, when we look at cells with different sizes.

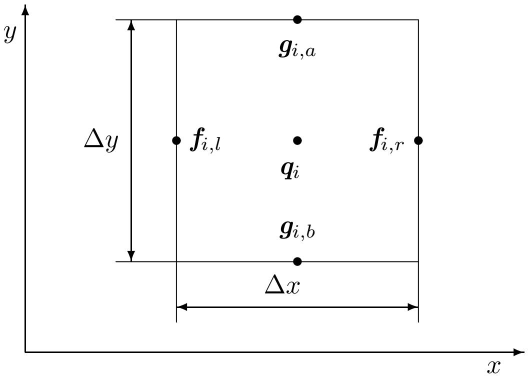


Figure 2.2: Average cell state $\boldsymbol{q}_i$ and average fluxes at the boundaries.

To discretise the finite-volume equations (2.7), we take the average states in the cells $\boldsymbol{q}_i$ as the only unknowns of the problem, no information is stored about the state anywhere else. Furthermore, the average states are only calculated at discrete time levels $k$ lying $\Delta t$ apart. Therefore, all equations for the problem must be approximated with equivalents that depend only on these average states $\boldsymbol{q}_i^k$. This set of approximate equations is called a discretisation of the Euler equations.

For unsteady problems, both the term with the time derivative and the terms with the fluxes are discretised. The first-order accurate forward-Euler discretisation of the time derivative $\frac{\partial}{\partial t}\boldsymbol{q}_i$ is, for two time levels $k$ and $k+1$,

$$\frac{\partial \boldsymbol{q}_i^k}{\partial t} = \frac{\boldsymbol{q}_i^{k+1} - \boldsymbol{q}_i^k}{\Delta t} + \mathcal{O}\left(\Delta t\right). \tag{2.9}$$

For the discretisation of the fluxes, one may first replace an average flux $\boldsymbol{f}_{i,n}$ (or $\boldsymbol{g}_{i,n}$) by the flux calculated from the average *state* over that face: $\boldsymbol{f}(\boldsymbol{q}_{i,n})$. According to Van der Maarel ([13], ch. 2), this limits the accuracy to second-order.

$$\frac{1}{\Delta y}\int_{\partial\Omega_{i,n}} \boldsymbol{f}\left(\boldsymbol{q}\right)\, dy = \boldsymbol{f}\left(\frac{1}{\Delta y}\int_{\partial\Omega_{i,n}} \boldsymbol{q}\, dy\right) + \mathcal{O}\left(\Delta x^2, \Delta y^2\right). \tag{2.10}$$

Then the average state $\boldsymbol{q}_{i,n}$ is approximated using the state $\boldsymbol{q}_i$ and the states $\boldsymbol{q}_{n_i}$ in the neighbour cells $\Omega_{n_i}$. One way to do this is to use an approximate Riemann solver. The next section describes this solver.

## 2.3. Osher's flux function

The idea of using a Riemann solver as a flux function was first introduced, for 1D problems, by Godunov [6]. His idea was to use the solution of a discontinuity problem to determine the fluxes $\boldsymbol{f}\left(\boldsymbol{q}_{i,n}\right)$ at the cell boundaries. This problem is the 1D Riemann problem of gas dynamics: a shock tube with a fictitious membrane at $x$ = 0 is filled with a fluid, the membrane separates two different, given states $\boldsymbol{q}_0$ and $\boldsymbol{q}_1$. At $t$ = 0, the membrane is removed instantaneously and waves start running into the fluid, two shocks / expansions and a contact discontinuity (figure 2.3). When the Riemann problem is used to calculate a flux, the average states at the left and right side of the cell face, $\boldsymbol{q}_L$ and $\boldsymbol{q}_R$, are chosen as initial state (these are cell face states $\boldsymbol{q}_{i,n}$ and $\boldsymbol{q}_{n_i,n+2}$ in two neighbour cells $\Omega_i$ and $\Omega_{n_i}$, $n+2$ denotes the opposite side of $n$). The resulting approximate flux $\boldsymbol{F}\left(\boldsymbol{q}_L, \boldsymbol{q}_R\right)$ is the flux calculated from the resulting state at $x$ = 0,

$$\boldsymbol{F}(\boldsymbol{q}_L, \boldsymbol{q}_R) = \boldsymbol{f}(\boldsymbol{q}(x=0)). \tag{2.11}$$

It is possible to find the exact solution of a Riemann problem, see e.g. [6, 18, 21], but this requires a lot of computation. Therefore, approximate Riemann solvers like the Osher scheme have been proposed. Osher's

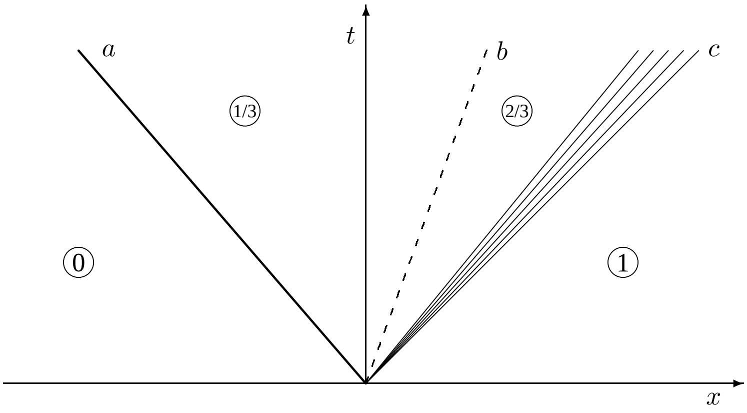


Figure 2.3: Wave pattern of a Riemann problem. Waves $a$ and $c$ are either a shock or an expansion. The intermediate wave $b$ is always a contact discontinuity.

solver is essentially an approximation to the Riemann problem that uses isentropic equations to calculate both shock / expansion waves (see [9, 15, 16]). This approximation is exact when both waves ($a$ and $c$ in figure 2.3) are expansions, since these are isentropic. In case of weak shocks, the states behind the shocks are still accurate, because the difference between a shock wave and an isentropic wave is of $\mathcal{O}\left(\Delta u^3\right)$, when $\Delta u$ is the speed change over the wave. A brief overview of the Osher scheme is given here, for a more thorough explanation see e.g. [16].

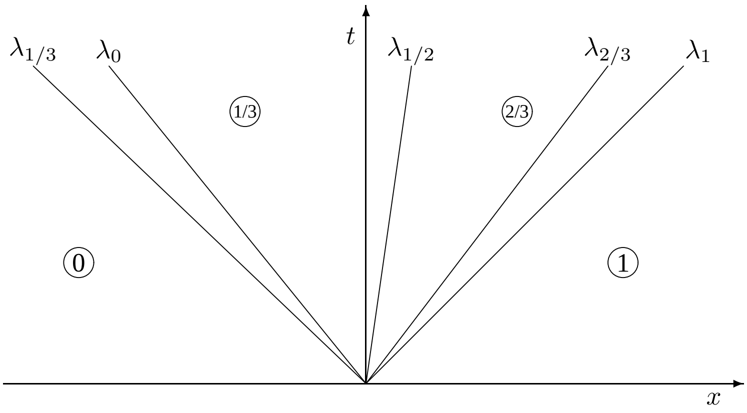


Figure 2.4: Osher's approximation to the Riemann problem of figure 2.3: always two isentropic waves. The four characteristics $\lambda_0$, $\lambda_{1/3}$, $\lambda_{2/3}$ and $\lambda_1$ are the limiting characteristics of the isentropic waves, $\lambda_{1/2}$ is the contact discontinuity. Note that the left wave, the equivalent of a shock wave, is 'overturned': $\lambda_0$ is steeper than $\lambda_{1/3}$.

The approximation to the Riemann problem in Osher's scheme is sketched in figure 2.4. The characteristics $\lambda$ are calculated as follows. For convenience, the state vector is taken as $(u, v, c, z)$, with $u$ the normal speed, $v$ the tangential speed, $c$ the speed of sound and $z$ the unscaled entropy, $z = \ln\left(p/\rho^\gamma\right)$. The Riemann problem is one-dimensional, but $v$ is calculated too, because the Osher solver is used here for 2D problems. $v$ has no influence on the other three components of the solution.

The scheme starts by calculating the post states (1/3) and (2/3). From the theory of characteristics, it is known that

$$\begin{aligned} u_0 + \frac{2}{\gamma-1}c_0 &= u_{1/3} + \frac{2}{\gamma-1}c_{1/3} \equiv \psi_0, \\ v_0 &= v_{1/3}, \\ z_0 &= z_{1/3}, \end{aligned} \tag{2.12a}$$

$$\begin{aligned} u_1 - \frac{2}{\gamma-1}c_1 &= u_{2/3} - \frac{2}{\gamma-1}c_{2/3} \equiv \psi_1, \\ v_1 &= v_{2/3}, \\ z_1 &= z_{2/3}, \end{aligned} \tag{2.12b}$$

and also,

$$\begin{aligned} u_{1/3} &= u_{2/3} \equiv u_{1/2}, \\ p_{1/3} &= p_{2/3}; \end{aligned} \tag{2.13}$$

the normal speed and the pressure do not change over a contact discontinuity. Solving equations (2.12) and (2.13) gives

$$\begin{aligned} c_{1/3} &= \frac{\gamma-1}{2}\left(\frac{\psi_0-\psi_1}{1+\alpha}\right), \\ c_{2/3} &= \alpha c_{1/3}, \\ u_{1/2} &= \frac{\psi_1+\alpha\psi_0}{1+\alpha}, \\ &\text{with} \\ \alpha &= \exp\left(\frac{z_1-z_0}{2\gamma}\right). \end{aligned} \tag{2.14}$$

Now the slopes of the 5 characteristics in figure 2.4 can be calculated (four limiting characteristics of isentropic waves and one contact discontinuity):

$$\begin{aligned} \lambda_0 &= u_0 - c_0, \\ \lambda_{1/3} &= u_{1/2} - c_{1/3}, \\ \lambda_{1/2} &= u_{1/2}, \\ \lambda_{2/3} &= u_{1/2} + c_{2/3}, \\ \lambda_1 &= u_1 + c_1. \end{aligned} \tag{2.15}$$

If the isentropic waves are expansions, then $\lambda_{1/3} > \lambda_0$ and $\lambda_1 > \lambda_{2/3}$, if they are compressions, then $\lambda_0 > \lambda_{1/3}$ and $\lambda_{2/3} > \lambda_1$. These waves are 'overturned', like the left wave in figure 2.4.

Once the shape of the problem, as in figure 2.4, is known, the flux at $x = 0$ can be calculated. In Osher's scheme, this calculation is based on the matrix $A = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{q}}$ from the quasi-linear formulation of the Euler equations. The matrix is split as

$$A = A^- + A^+,$$

where $A^-$ depends only on the negative eigenvalues of $A$ and $A^+$ only on the positive eigenvalues. Now $\boldsymbol{f}_{x=0}$ is estimated as

$$\boldsymbol{F}_O = \boldsymbol{f}\left(\boldsymbol{q}_0\right) + \int_{\boldsymbol{q}_0}^{\boldsymbol{q}_1} A^- \, d\boldsymbol{q}. \tag{2.16}$$

When the two compression / expansion waves in the Riemann problem are isentropic, then this integral does not depend on the integration path. Therefore, a smart choice is available for the integration path. In Osher's scheme, the integral is evaluated between the states in the Riemann problem.

$$\boldsymbol{F}_O = \boldsymbol{f}\left(\boldsymbol{q}_0\right) + \int_{\boldsymbol{q}_0}^{\boldsymbol{q}_{1/3}} A^- \, d\boldsymbol{q} + \int_{\boldsymbol{q}_{1/3}}^{\boldsymbol{q}_{2/3}} A^- \, d\boldsymbol{q} + \int_{\boldsymbol{q}_{2/3}}^{\boldsymbol{q}_1} A^- \, d\boldsymbol{q}. \tag{2.17}$$

It is proved in [16] that these integrals reduce to

$$\int_{\boldsymbol{q}_{(i-1)/3}}^{\boldsymbol{q}_{i/3}} A^- \, d\boldsymbol{q} = \begin{cases} \boldsymbol{f}\left(\boldsymbol{q}_{i/3}\right) - \boldsymbol{f}\left(\boldsymbol{q}_{(i-1)/3}\right) & \lambda_{(i-1)/3} < 0 \text{ and } \lambda_{i/3} < 0, \\ 0 & \lambda_{(i-1)/3} > 0 \text{ and } \lambda_{i/3} > 0, \\ \boldsymbol{f}\left(\boldsymbol{q}_s\right) - \boldsymbol{f}\left(\boldsymbol{q}_{(i-1)/3}\right) & \lambda_{(i-1)/3} < 0 \text{ and } \lambda_{i/3} > 0, \\ \boldsymbol{f}\left(\boldsymbol{q}_{i/3}\right) + \boldsymbol{f}\left(\boldsymbol{q}_{(i-1)/3}\right) - \boldsymbol{f}\left(\boldsymbol{q}_s\right) & \lambda_{(i-1)/3} > 0 \text{ and } \lambda_{i/3} < 0. \end{cases} \quad i = 1, 3. \tag{2.18a}$$

for the integrals across the two compression / expansion waves and to

$$\int_{\boldsymbol{q}_{1/3}}^{\boldsymbol{q}_{2/3}} A^- \, d\boldsymbol{q} = \begin{cases} \boldsymbol{f}\left(\boldsymbol{q}_{2/3}\right) - \boldsymbol{f}\left(\boldsymbol{q}_{1/3}\right) & \lambda_{1/2} < 0, \\ 0 & \lambda_{1/2} > 0, \end{cases} \tag{2.18b}$$

for the integral across the contact discontinuity. Here $\boldsymbol{q}_s$ is the state in the sonic point, which appears when $x = 0$ lies inside an isentropic wave. It is given by

$$\begin{aligned} u_s^0 &= c_s^0 = \frac{\gamma - 1}{\gamma + 1} \psi_0, \\ v_s^0 &= v_0, \\ z_s^0 &= z_0, \end{aligned} \tag{2.19a}$$

if the sonic point lies in the left wave, and by

$$\begin{aligned} u_s^1 &= -c_s^1 = \frac{\gamma - 1}{\gamma + 1} \psi_1, \\ v_s^1 &= v_1, \\ z_s^1 &= z_1, \end{aligned} \tag{2.19b}$$

if the sonic point lies in the right wave. Summing the integrals from equation (2.18) shows that evaluating (2.16) usually corresponds to taking the flux from just one state: the state at $x = 0$ in the approximate Riemann solution from figure 2.4. This is true when $x = 0$ does not lie in an isentropic wave, then $\boldsymbol{F}_O$ is set from $\boldsymbol{q}_0$, $\boldsymbol{q}_{1/3}$, $\boldsymbol{q}_{2/3}$ or $\boldsymbol{q}_1$. It even holds when $x = 0$ lies in an expansion wave, then the state used is the sonic state $\boldsymbol{q}_s^0$ or $\boldsymbol{q}_s^1$. Only when $x = 0$ lies in one or more compression waves, the output flux is summed from more than one state. An overview of all the possible combinations can be found in [16].

A recent study [4] shows that rather large errors of $\mathcal{O}(\Delta u)$ appear in the flux solution when two or more fluxes are summed. In all other cases, the error is of $\mathcal{O}(\Delta u^3)$ like the errors in the isentropic waves.

The advantage of the Osher scheme and other Riemann-based flux functions is that they base the resulting flux on the left and right states in a physically correct way. In case of subsonic flow, the flux is determined from both states, but for supersonic flow from one side, the flux is calculated with the upwind state only, just like in the real flow information only travels downwind.

Another advantage: the flux function can handle discontinuities in the solution well. It gives sharp shocks, especially when these shocks are grid-aligned.

To turn the Osher scheme in a flux function like equation (2.11), the states at the cell face $\boldsymbol{q}_L$ and $\boldsymbol{q}_R$ are used for $\boldsymbol{q}_0$ and $\boldsymbol{q}_1$. One question remains to be answered: how are $\boldsymbol{q}_L$ and $\boldsymbol{q}_R$ calculated from the average cell states

$\boldsymbol{q}_i$? Consider a cell $\Omega_i$ and its $n^{th}$ neighbour $\Omega_{n_i}$ $(n = b, r, a, l)$. These cells have one cell face in common, which is labelled $\partial\Omega_{i,n}$ on $\Omega_i$ and $\partial\Omega_{n_i,n+2}$ on $\Omega_{n_i}$. Then the average states $\boldsymbol{q}_{i,n}$ and $\boldsymbol{q}_{n_i,n+2}$ must be set, as input for the Osher function $\boldsymbol{F}_O\left(\boldsymbol{q}_{i,n}, \boldsymbol{q}_{n_i,n+2}\right)$.

The simplest discretisation is to put the cell face states equal to the average cell states:

$$\begin{aligned} \boldsymbol{q}_{i,n} &= \boldsymbol{q}_i, \\ \boldsymbol{q}_{n_i,n+2} &= \boldsymbol{q}_{n_i}. \end{aligned} \tag{2.20}$$

In calculating the flux, we make three errors.

1. Setting the average flux equal to the flux calculated from the average states along the cell face, equation (2.10).
2. Calculating this flux with the Osher solver.
3. Taking non-exact states at the cell boundaries, equation (2.20).

With these errors, the discretisation of equations (2.9) and (2.20) is $\mathcal{O}\left(h\right)$ accurate.

## 2.4. Boundary conditions

One advantage of the Osher scheme is that it can be extended in a straightforward way to deal with boundary conditions. Depending on the type of boundary, between zero and four conditions have to be prescribed (for 2D problems). These conditions are used to calculate the flux into or out of the cells lying adjacent to the boundary. This is done by calculating a state $\boldsymbol{q}_B$ at the boundary and setting the boundary flux from this state. For the input, only one state is required, $\boldsymbol{q}_L$ for a right boundary and $\boldsymbol{q}_R$ for a left boundary. Five different types of boundary conditions are used, they are described here. The description is taken from Spekreijse [16], a more thorough description can be found there.

*1. Supersonic inflow* On a boundary with supersonic inflow, four boundary conditions are needed: a full state vector $\boldsymbol{q}$ is specified and $\boldsymbol{q}_B$ is equal to this state.

*2. Supersonic outflow* For supersonic outflow, no boundary conditions are needed. $\boldsymbol{q}_B$ is equal to the state in the cell $\boldsymbol{q}_L$ or $\boldsymbol{q}_R$.

*3. Subsonic inflow* In case of subsonic inflow, three of the four characteristics point into the cell, so three boundary conditions are required, e.g. $u_B$, $v_B$ and $c_B$. In case of a right boundary, we then find with equations (2.12) and (2.13), but now with $\boldsymbol{q}_{2/3}$ partially specified,

$$\begin{aligned} c_{1/3} &= c_0 + \frac{\gamma-1}{2}\left(u_0 - u_B\right), \\ \rho_{1/3} &= \left(\frac{c_{1/3}^2}{\gamma}e^{-z_0}\right)^{\frac{1}{\gamma-1}}, \\ p_B = p_{1/3} &= \frac{\rho_{1/3}c_{1/3}^2}{\gamma}, \\ \rho_B &= \frac{\gamma p_B}{c_B^2}, \\ z_B &= \ln\left(\frac{p_B}{\rho_B^\gamma}\right). \end{aligned} \tag{2.21}$$

With $z_B$ calculated, the entire boundary state is known.

*4. Subsonic outflow* Now only one characteristic is running into the cell, so one boundary condition is needed. $p_B$ is used here. From the three equations (2.12a), it follows for a right boundary:

$$\begin{aligned} v_B &= v_0, \\ z_B &= z_0, \\ \rho_B &= \left(p_B e^{-z_B}\right)^{\frac{1}{\gamma}}, \\ c_B &= \sqrt{\frac{\gamma p_B}{\rho_B}}, \\ u_B &= u_0 + \frac{2}{\gamma - 1}\left(c_0 - c_B\right). \end{aligned} \tag{2.22}$$

*5. Wall* In case of a wall, one boundary condition is given: $u_B = 0$, the speed normal to a wall is zero. One wave is running into the cell, so, for a right boundary, again the equations (2.12a) are used:

$$\begin{aligned} u_B &= 0, \\ v_B &= v_0, \\ z_B &= z_0, \\ c_B &= c_0 + \frac{\gamma - 1}{2} u_0. \end{aligned} \tag{2.23}$$

When the boundary flux is calculated, only the momentum term with $p_B$ has a nonzero value. All the other fluxes are zero.

## 2.5. Definition of errors

In the previous sections, errors have been mentioned a few times. The definitions needed to analyze these errors are given here.

Take the continuous Euler equations (2.1). They are written in a symbolic way as

$$\bar{L}\bar{\boldsymbol{q}} = 0. \tag{2.24}$$

The operator $\bar{L}$ contains terms for both the time derivative and the fluxes, $\bar{\boldsymbol{q}}(x, y, t)$ is the exact solution. Now consider a grid $\Omega$ with cells $\Omega_i$. The discretised equations are written as

$$L\boldsymbol{q} = 0, \tag{2.25}$$

with $L$ the discretised equation, $L$ consists of the discretisations of the time derivative (2.9) and the flux equations with Osher's Riemann solver.

The *error* in the solution $\boldsymbol{e}$ is the difference between the exact solution $\bar{\boldsymbol{q}}$ and the discretised solution $\boldsymbol{q}$. Since each $\boldsymbol{q}_i$ is an average over cell $i$, the exact solution must also be averaged. Therefore, a restriction operator $R_i$ is defined as

$$R_i \bar{\boldsymbol{q}}(t) = \frac{1}{\Delta x_i \Delta y_i} \iint\limits_{\Omega_i} \bar{\boldsymbol{q}}(x, y, t)\, dx\, dy \tag{2.26}$$

and the error becomes

$$\boldsymbol{e} = R_i \bar{\boldsymbol{q}}(t) - \boldsymbol{q}_i(t). \tag{2.27}$$

The *truncation error* $\tau$ is the difference between the results of the continuous and the discretised operator. It is what comes out when the exact solution is put into the discretised operator $L$,

$$\tau_i(t) = L R_i \bar{\boldsymbol{q}}(t). \tag{2.28}$$

If the discretised solution would be exactly the same as the restricted solution of the continuous equation, then $\tau$ would be 0, because that means that $R_i \bar{\boldsymbol{q}}$ is the solution of $L\boldsymbol{q} = 0$. But usually the solutions differ. We say that a discretisation $L$ is first-order accurate if

$$\tau = \mathcal{O}\left(\Delta x, \Delta y, \Delta t\right). \tag{2.29}$$

# Chapter 3
# First-order adaptive-gridding algorithm

In the previous chapter, a finite-volume discretisation for the Euler equations is given. Basically, this is just a way to write the Euler equations in terms of the average states in cells, it does not say exactly how these states are calculated, when the fluxes are calculated etc. And most importantly, it does not say anything about cells with different sizes. For adaptive gridding, none of these matters is trivial. Therefore, an adaptive-gridding algorithm is described in this chapter, that takes care of the refined grid, refines and unrefines cells, determines where and when fluxes or states need to be calculated and then calculates them, with the equations from chapter 2. The first section describes the 'philosophy' behind the chosen adaptive-gridding algorithm and the second section the algorithm itself. In section 3, the data structure is given, the way the grid and the states are stored by a computer program. And finally, the discretisation of the previous chapter must be expanded and adapted to be used in combination with adaptive gridding. This is described in section 4.

## 3.1. The basics of adaptive gridding

As stated in chapter 1, a solution of the Euler equations usually has a few areas where the state changes rapidly (discontinuities, but also expansion fans, strongly curved boundaries, etc.), while the rest of the solution has a state that is constant or changes only slowly in time and space. When the problem is solved on a uniform grid, this grid must be very fine to resolve these areas with rapid changes accurately. But then the areas with small changes are resolved with far too much accuracy, since they could be resolved accurately enough on a much coarser grid.

The idea behind solution-adaptive grid refinement is to have this fine grid only there where it is really needed and to calculate the remainder of the solution on a coarser grid. This means that the entire solution is resolved accurately, but no unnecessarily accurate calculations are made for the areas with small changes.

For a time-dependent solution, the high-activity areas change in time. Shocks, expansions, etc. develop and move. This means that, ideally, the fine grid moves in time too. There are several ways to do this, one of them is known as grid enrichment: fine cells are added to a coarse grid. Basically, there are two ways to enrich a grid: one is to take a fine grid and move it across a coarser grid. This method was used by Berger and Oliger [3] and Berger and Colella [2]. Another approach is to start with a basic coarse grid and split the cells of this grid into finer cells when needed. When the fine cells are not needed anymore, they are merged to become a

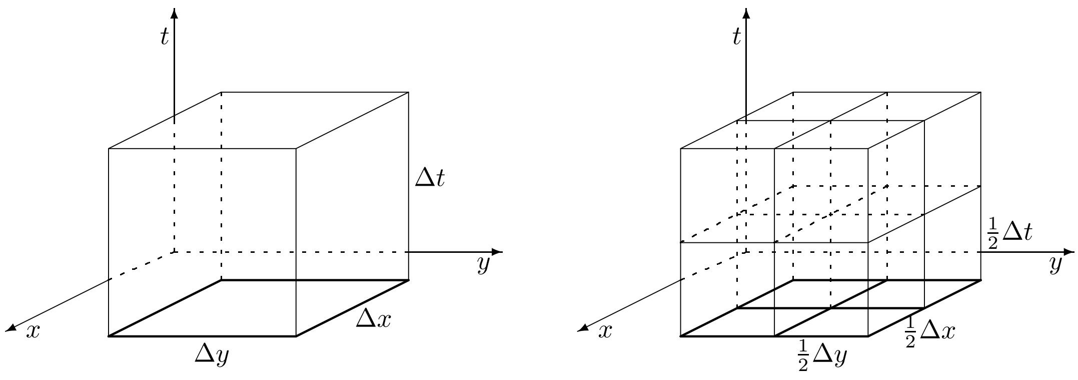


Figure 3.1: A cell with sizes $\Delta x$ and $\Delta y$ and time step $\Delta t$ is split into four smaller cells with a two times smaller time step.

coarse cell again. This approach was, among others, used by Trompert [19] and is also used here.

Consider a rectangular cell on a coarse grid, with size $\Delta x \times \Delta y$. In each single time step the state in this cell is advanced $\Delta t$ in time (figure 3.1). When this cell is split, it is divided into four cells that have the same geometrical proportions but are two times smaller in each spatial direction.

For explicit time integration, as is used here, the time step $\Delta t$ is limited by a stability requirement, the CFL requirement. For most first-order accurate methods, this requirement reads

$$|\lambda|_{max} \frac{\Delta t}{\Delta x} \leq 1, \tag{3.1}$$

with $|\lambda|_{max}$ the largest of the eigenvalues from equation (2.15). Previous studies [21] have shown that, for the Euler equations, the largest time step that is still stable gives the most accurate solution. Therefore, the time step is probably chosen with criterion (3.1), it is usually set at about 0.8 times the maximum value allowed. So when the coarse cell is split in cells that are half as big, the time step for these cells must also be halved to ensure stability (figure 3.1).

Therefore, in 2D, the same calculation requires eight times more work on the finer grid than on the coarse grid. If the cell is refined $n$ times, then the calculation will take $2^{3n}$ more times. This means that the total amount of work increases rapidly when cells are refined more than once. An example: a five times refined cell requires 32768 times the work of an unrefined cell, so the amount of computational work required to update the area of a single coarse cell a single coarse time step on a five times refined grid is about equivalent to that required to advance an entire, typical 2D problem (like those in chapter 5) one time step on a uniform grid!

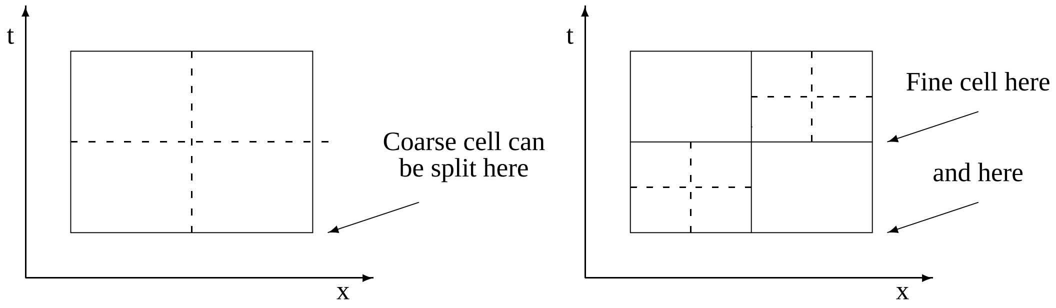


Figure 3.2: Small cells can be split more often than big cells (1D example).

So, to gain speed, it is very important that cells are not split unnecessarily. It is inefficient to determine the grid for every coarse time step and not to change it during that time step. Coarse cells can only be split after each coarse time step, but cells that are refined once can be split at two moments in the same time (figure 3.2). We can use this to get a more versatile and thus more efficient grid.

Again, an example: a shock in a 2D problem is usually about four cells wide (see the results in chapter 5 and 8). So, to capture a shock on an $n$ times refined grid, we need not split an entire coarse cell into $n \times n$ fine cells, but a strip of $4 \times n$ cells is enough. This strip must be moved often during one coarse time step.

Summarizing, the algorithm will start on a coarse grid, this grid is refined by splitting the cells in four as often as necessary. When the fine grid is not needed anymore, the cells are merged again. This splitting and merging is allowed as often as possible to minimize the number of cells needed.

## 3.2. An adaptive-gridding algorithm

In this section, an algorithm is presented, based on the general observations in the previous section.

*3.2.1 The grid*

Consider an Euler problem, equation (2.1) on a spatial domain $\Omega$ that consists of one or more rectangles. Let this domain be divided into $N$ rectangular cells $\Omega_i$, such that

$$\Omega = \Omega_1 \cup \Omega_2 \cup \ldots \cup \Omega_N.$$

We assume that these cells are obtained by splitting the cells of an original coarse grid between 0 and a maximum $M$ times. We say that a cell that has been split $l$ times is on *level* $l$ and denote all the cells that are on level $l$ by $\Omega^l$, ($l = 0, 1, \ldots, M$), see figure 3.3. $\Omega^0$ is the set of cells that are not split, so not the entire original coarse grid[1]. This entire original grid is denoted by $\Omega^0_*$. The actual grid is a function of time.

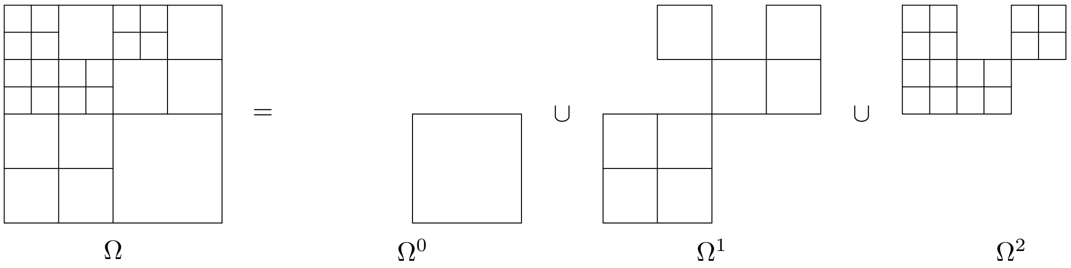


Figure 3.3: A grid $\Omega$ consists of several levels $\Omega^l$.

*3.2.2 Advancing the cell states*

If the states $\boldsymbol{q}_i^k$ are known after some coarse time step $k$, we want to find the state one time step later, $\boldsymbol{q}_i^{k+1}$. When all the cells have level 0, this is easy: the fluxes across all the cell faces are calculated with Osher's scheme and then all cell states are advanced simultaneously with a formula like (2.9). But when cells have different sizes, problems appear. For instance, smaller cells also have smaller time steps. And what to do with the fluxes when a cell has two smaller neighbours on one face?

At least we have one guideline: the principle of conservation. What flows out from one cell *must* flow into the next cell. So if two small cells lie next to one big cell, the flux into (or out of) the big cell must equal the sum of the fluxes out of (or into) the small cells. And if the small cells have two time steps instead of one for the big cell, then the flux into the big cell is equal to the sum over two small time steps of the fluxes out of the small cells.

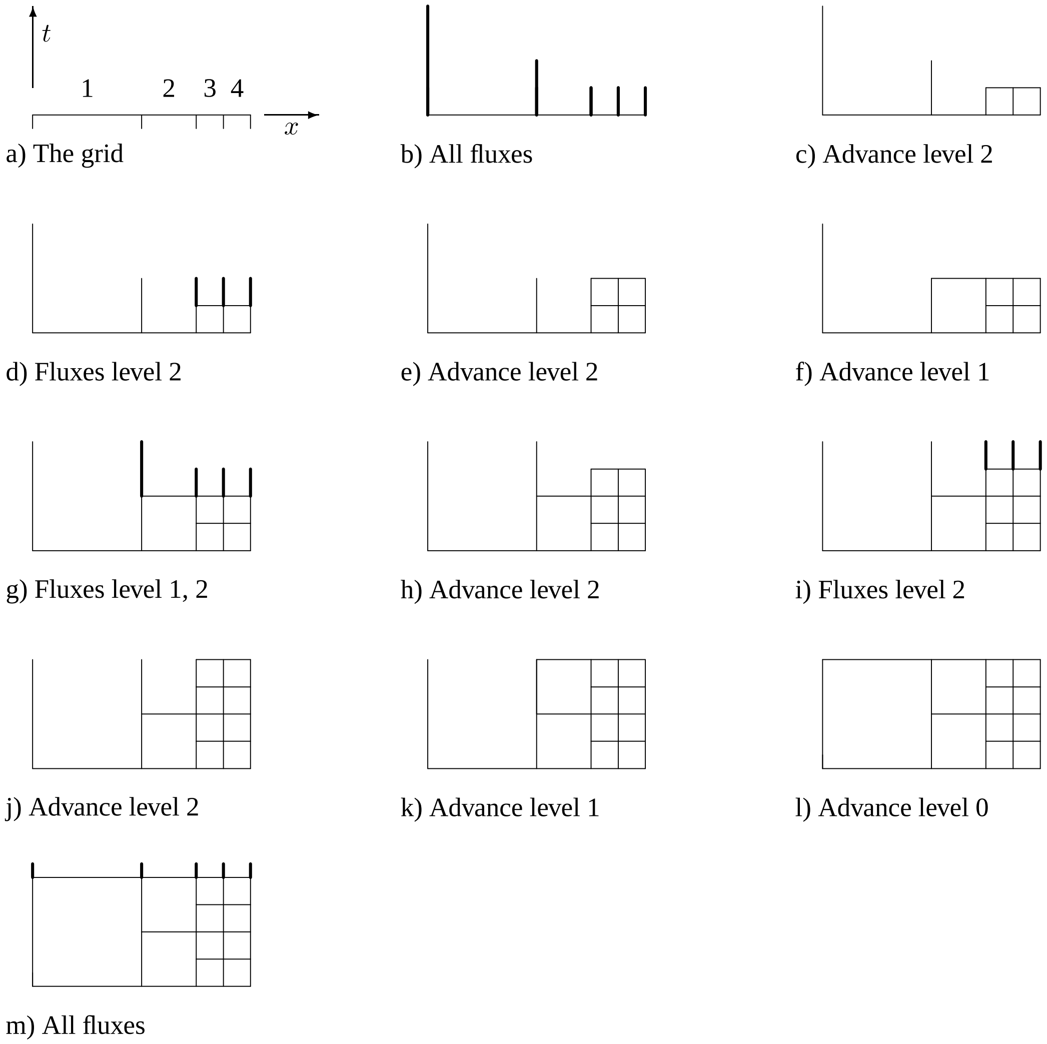
A way to advance the cell states, based on these principles, is the following algorithm. It will first be explained by an example: take the (1D) example grid from figure 3.4. The grid is shown in (a): one cell left at level 0, one at level 1 in the middle and two at level 2 to the right. We start by calculating the fluxes in all the cell faces (b). The fluxes into the smallest cells 3 and 4 at this moment are known, but we do not yet know the flow into cell 3 at a later time, so we can not yet sum these to get the flux into the bigger cell 2 from cell 3. Therefore, only the smallest cells are advanced a time step $\Delta t^l$,

$$\Delta t^l = 2^{-l}\Delta t \qquad l = 0, 1, \ldots, M. \tag{3.2}$$

In this case, the smallest level is 2, so these cells are advanced (c).

Now the fluxes into the cells at level 2 are calculated again, with the new states in these cells (d) and level 2 is advanced once more (e). By summing the fluxes from (b) and (d), we know also the flux into the level 1 cell 2, from cell 3, so level 1 can be advanced too (f). Calculate new fluxes (g), advance level 2 (h), fluxes level 2 (i), advance level 2 (j). Again we can sum fluxes and update level 1 (k), but now we can also sum the fluxes from cell 2 into the level 0 cell 1 and thus advance level 0 (l). After calculating all the fluxes (m) the time step is finished and we can proceed with (c) again: advance level 2.

[1] In methods for steady problems, especially multigrid methods, cells are kept after they are split. Here, this is not necessary.



a) The grid

b) All fluxes

c) Advance level 2

d) Fluxes level 2

e) Advance level 2

f) Advance level 1

g) Fluxes level 1, 2

h) Advance level 2

i) Fluxes level 2

j) Advance level 2

k) Advance level 1

l) Advance level 0

m) All fluxes

Figure 3.4: A 1D example of an advancing sequence.

### *3.2.3 Refining the grid*

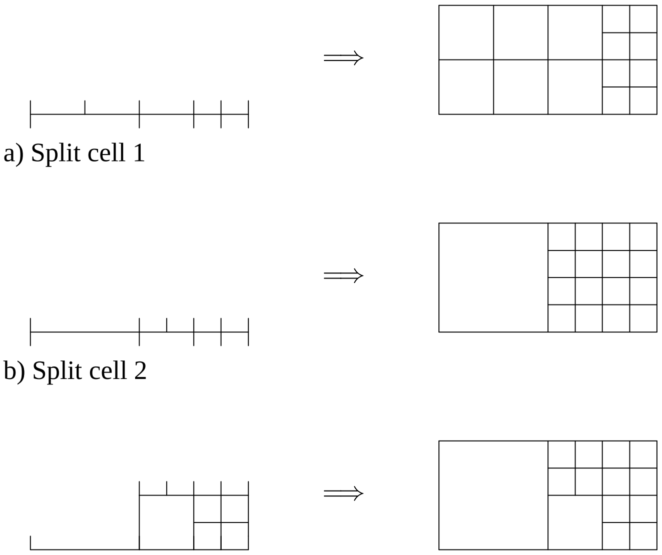
Until now, we have considered a grid that is fixed in time. But where and when can cells be refined (split in four)? Take the previous example from figure 3.4.

Before the first flux calculation (figure 3.4b) we could have split cell 1 into two cells at level 1 (two, not four, because the example is 1D). This grid and the result at the end of the time step are shown in figure 3.5a. At the same time, we could have split cell 2 into two level 2 cells (figure 3.5b). In the following algorithm, this will not be allowed because the grid changes two levels at once then (see section 4.4), but it can be done in principle. There even is another chance to split cell 2, after one advance step (that means after figure 3.4f). The result is shown in figure 3.5c. Cell 3 and 4 can never be split because 2 is the maximum level for this problem.

Summarizing: a cell can be refined after *every* advance step for that cell. The cells in $\Omega^M$ cannot be split.

### *3.2.4 Unrefining the grid*

The cells must also be unrefined at a certain moment, four cells (two in 1D) are merged to form a single cell again. Take the problem from figure 3.4: the cells 3 and 4 can be merged into one level 1 cell before step (b),

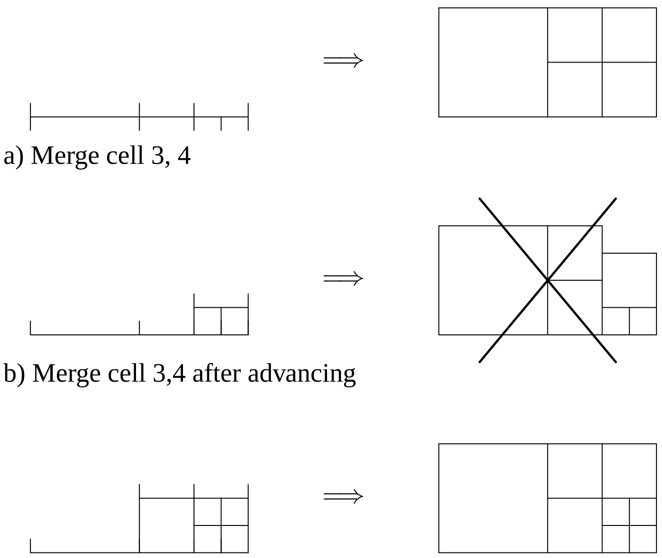


a) Split cell 1

b) Split cell 2

c) Split cell 2 after advancing

Figure 3.5: Refinement possibilities in the sequence of figure 3.4.



a) Merge cell 3, 4

b) Merge cell 3,4 after advancing

c) Merge cell 3, 4 after advancing twice

Figure 3.6: Unrefinement possibilities in the sequence of figure 3.4.

see figure 3.6a, but cell 2 cannot be merged into a level 0 cell. It has to be merged with another level 1 cell and the place of this cell is taken by cells 3 and 4. (Of course, we could merge cells 3 and 4 and then merge the resulting cell with cell 2 to form a level 0 cell.)

Now look at step (c) of figure 3.4. Cells 3 and 4 have just been advanced, but they cannot be merged into a level 1 cell. This cell would not fit in the time steps for the level 1 cells (figure 3.6b). Cell 3 and 4 *can* be merged after step (f) (figure 3.6c), but now cell 2 cannot be merged, because the resulting level 0 cell does not fit in.

So cells can also be merged after each time step, but not the cells on the *lowest* level that was just advanced (level 2 in figure 3.6b, level 1 in figure 3.6c). Also, all four cells to be merged must be on the same level.

### *3.2.5 The algorithm*

An algorithm, based on the previous considerations, is given here. It acts on the cells per level, so when something is done, it is always done for all the cells on $\Omega^l$, for some $l$.

The algorithm follows the advancing procedure followed in figure 3.4. Its basis is that all levels, except level 0, are advanced a multiple of two times. When a level is advanced twice, a total time step is taken that is equal to one time step on the next coarsest level (eq. (3.2)), so we can also advance this lower level. But if the level is advanced for the first time, no lower level can be advanced and we have to return to the highest level.

The algorithm, called Sagmmu[2], is given here in pseudo-code.

```
Program Sagmmu
do l = 0 to M times_advanced(l) = 1
do k = 1 to k_max
        call Timestep(M)
end do
end Sagmmu

Subroutine Timestep(l_now)
advance $\Omega^{\text{l\_now}}$
if times_advanced(l_now) = 1 then
        do i = l_now to M - 1 refine_check $\Omega^i$
        do i = l_now + 1 to M unrefine_check $\Omega^i$
        do i = l_now to M fluxes $\Omega^i$
        if l_now = 0 return, time step finished
        times_advanced(l_now) = 2
        call Timestep(M)
else
        times_advanced(l_now) = 1
        call Timestep(l_now - 1)
end if
end Timestep
```

For better understanding, the reader is encouraged to follow the steps taken in figure 3.4 with this algorithm.

In the algorithm a refinement / unrefinement check is indicated. Here it is decided which cells in $\Omega^l$ will be refined. The decision on where to refine or to unrefine is taken with a refinement criterion, treated in chapter 4. The exact treatment of fluxes and advancing is explained in section 3.4.

[2] Solution-Adaptive Gridding with Maximum Memory Usage…The test code was named SAGMMU because more variables are stored than strictly necessary. It was considered more important to have these variables at hand and to be able to study them, than to save memory.

## 3.3. The data structure

On a uniform 2D grid, it is quite easy to store the data from an Euler calculation. The states and fluxes are stored in a 2D array corresponding to the grid and the neighbours are found by just adding $\pm 1$ to one of the indices.

But for a locally refined grid, this does not work. A cell that has two neighbours on one face cannot be stored in an $i, j$-array. And the total number of cells is time-dependent. Therefore, a different data structure is needed.

Basically, the cells are stored at random in 1D arrays and all the relations with other cells are kept in pointer arrays. All arrays have the same length and the same position in all the arrays always corresponds to the same cell. Two different types of data are kept: geometrical and state / flux data.

### 3.3.1 Geometrical data

These are all data that have to do with the grid geometry: they are coordinates, cell levels, neighbour pointers and family ties.

*Coordinates* The $x$- and $y$-coordinate of the lower left corner are stored for each cell. As the cells are rectangular and the size of each cell is known:

$$\begin{aligned} \Delta x_i &= 2^{-l}\Delta x, \\ \Delta y_i &= 2^{-l}\Delta y, \end{aligned} \tag{3.3}$$

the other 3 corners can be calculated easily.

*Cell level* A single integer array is used to store the level $l$ per cell.

*Neighbours* Because the cells are stored at random, pointers are needed, arrays with the index numbers of the neighbour cells. Only one neighbour is stored per face, but these are chosen such that the others can be found quickly if there is more than one neighbour on the same face.

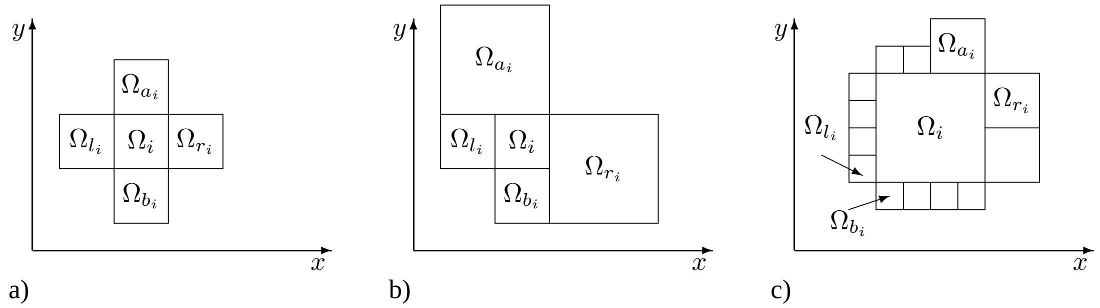


Figure 3.7: Stored neighbours. The neighbours have the same size as $\Omega_i$ (a), they are bigger (b) or smaller than $\Omega_i$ (c).

When the neighbours are of the same size as $\Omega_i$ (figure 3.7a) or bigger (figure 3.7b), it is obvious which cells to choose as neighbours. When the neighbour cells are smaller, the neighbours stored are:

- below: the leftmost cell,
- right: the highest cell,
- above: the rightmost cell,
- left: the lowest cell.

This allows quick searching: when, say, all cells on the right cell face need to be known, we start with the right neighbour $\Omega_{r_i}$. Then we take the neighbour below of $\Omega_{r_i}$. This is the leftmost cell below $\Omega_{r_i}$, so it lies next to $\Omega_i$ as well! (figure 3.8). We can continue like this until all cells on the right cell face of $\Omega_i$ are known. This works for *all* other cell faces too, which is left for the reader to verify.

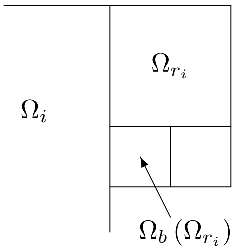


Figure 3.8: The lower neighbours of $\Omega_{r_i}$ lie next to $\Omega_i$.

This way of storing neighbours makes it possible to find any cell around $\Omega_i$ without having to search more than a few cells. The number of search steps depends on the level difference between the neighbour cells, but it does not depend on the total number of cells.

*Family ties* These describe the way the grid $\Omega(t)$ is formed from the basic coarse grid $\Omega^0_*$. Only one family tie is needed: the integer array of mother pointers[3].

When a cell $\Omega_i$ is split, four new cells are formed (figure 3.9). The cell in the lower left corner is called the mother, it gets the index $\Omega_i$ (and the memory locations for $\Omega_i$). So in the data structure, the big cell does not disappear, it becomes one of the smaller cells. Three new memory locations are claimed for the daughters right ($dr$), above ($da$) and on the diagonal ($dd$). Their mother pointers are set to $\Omega_i$, but $\Omega_i$ itself keeps the mother pointer of the old big cell. Nothing is stored to indicate that $\Omega_i$ has daughters or that cells are sisters: it is not needed.

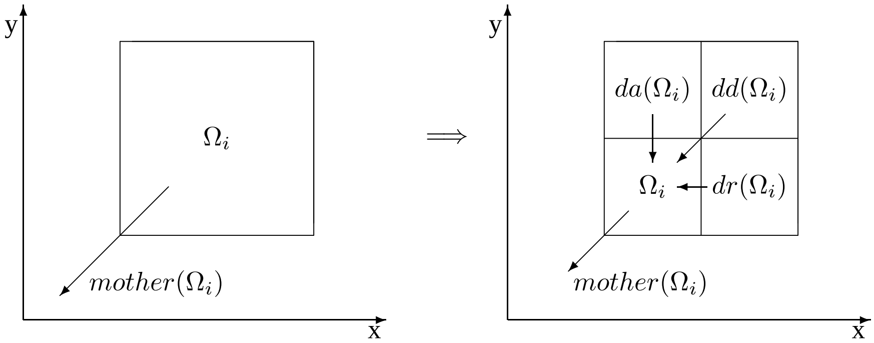


Figure 3.9: Mother pointers of a big cell (a) and four refined cells(b).

The mother pointer is only needed to know which cells can be merged. The only cells that may be merged are those cells that were once a single big cell, so we need a way to find these cells and to make sure that none of them is split (like cell 3 and 4 in figure 3.5, cell 2 cannot be merged there). This can be done with two checks:

[3] As in [21], the terms used to describe the cells are chosen female. Apart from positive discrimination and the fact that the Dutch word 'cel' is indeed female, the best argument for this is that a *father* cell is unlikely to give birth to other cells...

1. Find a cell $\Omega_i$ for which $\Omega_{l_i} = \text{mother}\,(\Omega_i)$. Then we have found a cell that was $dr$ when the big cell was split and we are sure that its mother cell is not split again.
2. From these $\Omega_i$, find those for which $\exists l \mid \Omega_i, \Omega_{l_i}, \Omega_{a_i}, \Omega_l\,(\Omega_{a_i}) \in \Omega^l$. Then these four cells can be merged.

When these four cells are merged, the new big cell $\Omega_i$ gets the mother pointer from the small mother cell $\Omega_i$ and the situation is again like it was before the cell was split. Thus, only the mother pointers and the neighbour pointers are needed to store the grid structure.

### *3.3.2 State and flux data*

The flow behaviour is stored with these data.

*State* Per cell, four state values are kept: $\rho$, $\rho u$, $\rho v$ and $\rho E$ (equation (2.2)), so the states are stored in a 4 × many 2D real array.

*Fluxes* To enable a study of the fluxes, they are stored per cell face, so each cell has four faces times four fluxes is 16 fluxes. They are stored in a 4 × 4 × many 3D real array. Note that the fluxes stored in vertical faces are $\boldsymbol{f}$ and those in horizontal faces are $\boldsymbol{g}$ (equation (2.2)). Furthermore, when two facing cells have the same level, the same flux is stored in both cells.

### *3.3.3 Other lists*

Apart from the data per cell, other lists are needed:

A list of cells per level. As most operations are done per level, it is useful to store all cells on one level in a list. This prevents unnecessary searching.

All the cell arrays must be chosen so big that the grid fits in always, so usually they are partially empty. A list is kept with all the free places in the arrays and when new cells are created, their places in the arrays are taken from this list. When cells are removed, the free memory spaces are added to the list again.

## 3.4. State and flux evaluation

In section 3.2, it was seen that states are advanced and fluxes calculated per level $\Omega^l$. The previous section 3.3 showed that one state vector is stored per cell and four flux vectors, one per cell face. This section shows how exactly these states and fluxes are calculated.

*State* By taking the basic finite-volume Euler equation (2.7), substituting equation (2.9) for the time derivative and dividing by $\frac{\Delta x \Delta y}{\Delta t}$, we find

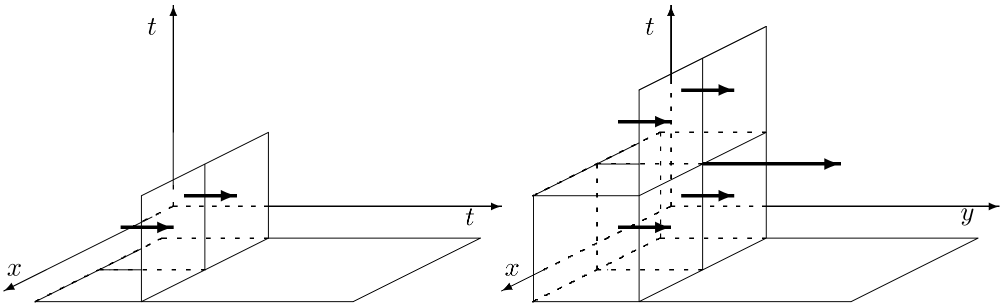
$$\boldsymbol{q}_i^{k+1} = \boldsymbol{q}_i^k - \left(\boldsymbol{F}_{O\,i,r} - \boldsymbol{F}_{O\,i,l}\right)\frac{\Delta t}{\Delta x} - \left(\boldsymbol{G}_{O\,i,a} - \boldsymbol{G}_{O\,i,b}\right)\frac{\Delta t}{\Delta y}. \tag{3.4}$$

This equation is used for all the advancing. One advantage is that both $\frac{\Delta t}{\Delta x}$ and $\frac{\Delta t}{\Delta y}$ are the same for every cell level, so the equation can be used for all the cells without any changes.

*Fluxes* The fluxes in equation (3.4) are, from equation (2.2),

$$\boldsymbol{f} = \begin{pmatrix} \rho u \\ p + \rho u^2 \\ \rho u v \\ \rho u H \end{pmatrix} \qquad \boldsymbol{g} = \begin{pmatrix} \rho v \\ \rho u v \\ p + \rho v^2 \\ \rho v H \end{pmatrix}.$$

When the fluxes for all cells on a level are calculated, most neighbour cells have the same level, so the flux between them is calculated with Osher's scheme and stored in both cells. But what if the neighbour cell is bigger?



a) First time step

b) Second time step, fluxes are summed

Figure 3.10: Flux evaluation in neighbour cells with different sizes.

Let two cells $\Omega_i$ and $\Omega_{i_2}$ on level $l$ lie next to a cell $\Omega_{n_i}$ on level $l-1$ (figure 3.10). Then the flux into $\Omega_i$ and $\Omega_{i_2}$ is calculated, these cells are advanced (figure 3.10a) and the fluxes into the small cells are calculated again, with the new cell states, but the old state for $\Omega_{n_i}$, because this cell is not yet updated. Then the flux into $\Omega_{n_i}$ is set (figure 3.10b). According to section 3.2, the fluxes are summed, but it is the total inflow over the cell face during the time step $\boldsymbol{f}\Delta y_i \Delta t_i$ which is meant there. So, from conservation of fluxes,

$$\boldsymbol{f}^k_{n_i,n+2} 2^{-(l-1)}\Delta y\, 2^{-(l-1)}\Delta t = \left(\boldsymbol{f}^k_{i,n} + \boldsymbol{f}^{k+1}_{i,n} + \boldsymbol{f}^k_{i_2,n} + \boldsymbol{f}^{k+1}_{i_2,n}\right) 2^{-l}\Delta y\, 2^{-l}\Delta t$$

and thus

$$\boldsymbol{f}^k_{n_i,n+2} = \frac{1}{4}\left(\boldsymbol{f}^k_{i,n} + \boldsymbol{f}^{k+1}_{i,n} + \boldsymbol{f}^k_{i_2,n} + \boldsymbol{f}^{k+1}_{i_2,n}\right). \tag{3.5}$$

The fluxes as they appear in (3.4) must be *averaged*.

So when a cell has a bigger neighbour, the flux is calculated as always, stored for the small cell and *added* to the flux for the big neighbour cell, but divided by four. In the end, when the bigger cell is advanced, the sum of these contributions is equal to the average flux from equation (3.5).

In the last case, when the neighbour cells are smaller, no fluxes are calculated. As seen above, the flux in the bigger cell is set when the fluxes for the smaller cells are calculated.

*State initialisation* When cells are split or merged, the state in the new cell(s) must be set before the calculation can continue. This must be done with at least the same order of accuracy as the discretisation of the fluxes and the time derivative. For cells that were just refined, we take the state in the new small cells equal to the state in the old big cell (a *prolongation* $P_l^{l+1}$ from level $l$ to level $l+1$):

$$\boldsymbol{q}_i^{l+1} = \boldsymbol{q}_{dr}^{l+1} = \boldsymbol{q}_{dd}^{l+1} = \boldsymbol{q}_{da}^{l+1} = P_l^{l+1}\boldsymbol{q}_i^l \equiv \boldsymbol{q}_i^l. \tag{3.6}$$

By developing the exact solution $\bar{\boldsymbol{q}}$ in a Taylor series around the centre of the old cell and averaging it over both the old big cell and the four new cells, we see that the approximation (3.6) is $\mathcal{O}\left(\Delta x, \Delta y\right)$ accurate.

To set the state in a new big cell after an unrefinement, we use a *restriction* $R^l_{l+1}$, the four cells are averaged (compare this with equation (2.26)):

$$\boldsymbol{q}_i^l = R^l_{l+1}\boldsymbol{q}^{l+1} \equiv \frac{1}{4}\left(\boldsymbol{q}_i^{l+1} + \boldsymbol{q}_{dr}^{l+1} + \boldsymbol{q}_{dd}^{l+1} + \boldsymbol{q}_{da}^{l+1}\right). \tag{3.7}$$

Taking again a Taylor expansion of the exact solution and averaging it over the big and the four small cells shows that this approximation is $\mathcal{O}\left(\Delta x^2, \Delta y^2\right)$ accurate.

# Chapter 4
# Refinement criteria

The previous chapters describe *how* to refine a basic grid, but another question is still unanswered: *where* should the grid be refined? And, the most fundamental question, *why*?

Answering this question seems easy: we want to refine where things happen in the solution. But this answer is wrong: the truth is that we want the best possible solution at the lowest computational cost, so we must refine to find this solution. This does not necessarily mean refining where the solution is non-smooth.

But what is the best solution? In principle, this depends on what the users want to see. They may be interested in forces and moments on surfaces, then the global error of the solution is important. But for accurate pressure distributions, a good local error must be obtained. And even more exotic demands are possible.

In the next sections, refinement criteria are defined. These are parameters, calculated per cell, that indicate whether a cell must be split, kept the same size, or merged. *The choice for a refinement criterion must be made by the user*, because she or he decides which aspect of the solution is important. Therefore, no ‘best’ refinement criterion can be chosen. Instead, criteria are given with their behaviour, for the user to choose.

But when testing the criteria in chapter 5, we need some way to compare their performance. Therefore, a good all-round definition of the requirements for a refinement criterion is used:

1. Small enough global errors. With discontinuities, it is very difficult to get the errors in their neighbourhood small, but when integrating the errors over the entire domain, these large errors on a small domain are not that important. Low global errors assure accurate computation of forces and moments on the boundaries, important for e.g. wing profile design.

2. Sufficient resolution. The solution must enable us to see what is going on, all flow features must be clearly visible. This can help a researcher or designer with understanding the flow phenomena and thus explain force calculations and even predict the nature of the calculation errors.

Three different refinement criteria are given in the next sections. The last section deals with buffers, cells that are split to ensure a smooth distribution of refined cells.

## 4.1. Gradient $\rho$ criterion

The most obvious quantification of the ‘where things happen’ idea is to look for gradients in the solution. Where one of the components of the state vector has a steep gradient, the grid is refined. A good choice is to use $\rho$, the first component of the state vector, because this component changes in shocks, expansions and contact discontinuities. Furthermore, it is the only component that is the same for a stationary wave and a running wave with the same strength.

Ideally, we want the grid size to be inversely proportional to the gradient of $\rho$. Define a threshold parameter $d$, chosen by the user. Then it is desired that, for all cells:

$$\Delta x_i = \frac{d}{\left|\frac{\partial \rho_i}{\partial x}\right|}. \tag{4.1}$$

A similar formula can be formulated for $\Delta y$. Discretised on a uniform grid:

$$2^{-l_i}\Delta x = \frac{d}{\frac{|\rho_{n_i} - \rho_i|}{2^{-l_i}\Delta x}}. \tag{4.2}$$

And thus, in all directions and for both $\Delta x$ and $\Delta y$,

$$|\rho_{n_i} - \rho_i| = d \qquad n = b, r, a, l. \tag{4.3}$$

So, ideally, when the cell sizes are inversely proportional to the first derivative of $\rho$, the undivided difference of $\rho$ between all neighbour cells must be the same.

If two neighbour cells are not the same size, the derivative of $\rho$ must be calculated otherwise (see figure 4.1),

$$\frac{\partial \rho}{\partial x} = \frac{\rho_{n_i} - \rho_i}{\frac{1}{2}\Delta x_i + \frac{1}{2}\Delta x_{n_i}} + \mathcal{O}(\Delta x) = \frac{\rho_{n_i} - \rho_i}{2^{-l_i}\Delta x \left(\frac{1}{2} + 2^{l_i - l_{n_i} - 1}\right)} + \mathcal{O}(\Delta x). \tag{4.4}$$

The revised version of (4.3) is

$$\frac{|\rho_{n_i} - \rho_i|}{\frac{1}{2} + 2^{l_i - l_{n_i} - 1}} = d. \tag{4.5}$$

As we cannot choose $\Delta x_i$ and $\Delta y_i$ independently and we have to stick to the sizes that can be obtained by splitting the basic grid, this equation cannot be satisfied always. And when the solution contains discontinuities, it is clearly impossible to satisfy equation 4.5. Instead, a refinement criterion $C_{\partial\rho}$ is defined as the maximum of (4.5),

$$C_{\partial\rho} = \max_{n=b,r,a,l} \left( \frac{|\rho_{n_i} - \rho_i|}{\frac{1}{2} + 2^{l_i - l_{n_i} - 1}} \right), \tag{4.6}$$

and an upper value $d_s$ (split) and lower value $d_m$ (merge) are chosen for $d$. When $C_{\partial\rho}$ exceeds $d_s$ in a cell, it is split. When $C_{\partial\rho}$ becomes lower than $d_m$ in each of four neighbour cells, these four cells are merged.

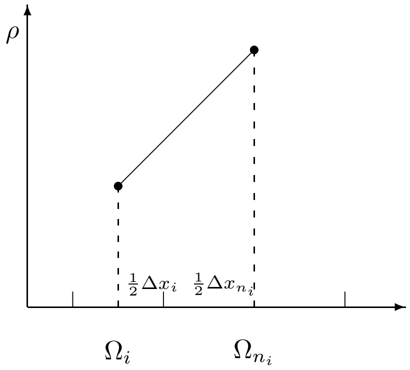


Figure 4.1: Approximation of $\rho_x$ on non-uniform grid.

The relation between $d_s$ and $d_m$ is important. Theoretically, when a cell is refined but the gradient does not change, $C_{\partial\rho}$ becomes two times smaller. When cells are unrefined, $C_{\partial\rho}$ becomes two times bigger. But we do not want cells that were just split to merge again after one step or vice versa, so $d_m$ must be more than two times smaller than $d_s$.

The gradient $\rho$ criterion is not the most fancy refinement criterion possible, as numerical results in section 5.2 show. To get good results, it is usually needed to choose $d_s$ so low that all waves are filled with small cells. But the criterion is very simple, robust and stable and can handle problems with shocks well.

## 4.2. SECOND DERIVATIVE $\rho$ CRITERION

The gradient $\rho$ criterion is not based on errors in the solution. Other criteria are possible, based on the truncation error $\tau$, the difference between the exact equation (2.7) and its discretised form (3.4). For first-order accurate discretisations of linear equations, it can be shown that

$$\tau \sim q_{xx}. \tag{4.7}$$

Therefore, it makes sense to base a refinement criterion on the second derivative of a component of $\boldsymbol{q}$. For the same reasons as in the previous section, $\rho$ is chosen. So again, a parameter $d$ is defined and we want

$$\Delta x_i = \frac{d}{|\rho_{xx,i}|} \tag{4.8}$$

to hold, as well as a similar relation for $\Delta y_i$.

The second derivative of $\rho$ is calculated as in the previous section, but now the derivative of the first derivative is taken (figure 4.2). Using the neighbour cells on two sides of $\Omega_i$, $\Omega_{n_i}$ and $\Omega_{(n+2)_i}$, the second derivative becomes

$$\begin{aligned} \rho_{xx} &= \frac{\frac{\rho_{n_i}-\rho_i}{2^{-l_i}\Delta x\left(\frac{1}{2}+2^{l_i-l_{n_i}-1}\right)} - \frac{\rho_i-\rho_{(n+2)_i}}{2^{-l_i}\Delta x\left(\frac{1}{2}+2^{l_i-l_{(n+2)_i}-1}\right)}}{\frac{1}{2}2^{-l_i}\Delta x\left(\frac{1}{2}+2^{l_i-l_{n_i}-1}\right)+\frac{1}{2}2^{-l_i}\Delta x\left(\frac{1}{2}+2^{l_i-l_{(n+2)_i}-1}\right)} + \mathcal{O}\left(h\right), \\ &= \frac{\frac{\rho_{n_i}-\rho_i}{\frac{1}{2}+2^{l_i-l_{n_i}-1}} + \frac{\rho_i-\rho_{(n+2)_i}}{\frac{1}{2}+2^{l_i-l_{(n+2)_i}-1}}}{2^{-2l_i}\Delta x^2\left(1+2^{l_i-l_{n_i}-1}+2^{l_i-l_{(n+2)_i}-1}\right)} + \mathcal{O}\left(h\right). \end{aligned} \tag{4.9}$$

This equation is only $\mathcal{O}(h^2)$ accurate if the cells have the same size. Taking the maximum from two directions, the refinement criterion becomes

$$C_{\partial^2\rho} = \max_{n=b,r}\left(\frac{\frac{\rho_{n_i}-\rho_i}{\frac{1}{2}+2^{l_i-l_{n_i}-1}} + \frac{\rho_i-\rho_{(n+2)_i}}{\frac{1}{2}+2^{l_i-l_{(n+2)_i}-1}}}{2^{-l_i}\Delta x\left(1+2^{l_i-l_{n_i}-1}+2^{l_i-l_{(n+2)_i}-1}\right)}\right). \tag{4.10}$$

Limits $d_s$ and $d_r$ are defined just like those for the gradient $\rho$ criterion, $d_m$ is chosen more than two times smaller than $d_s$.

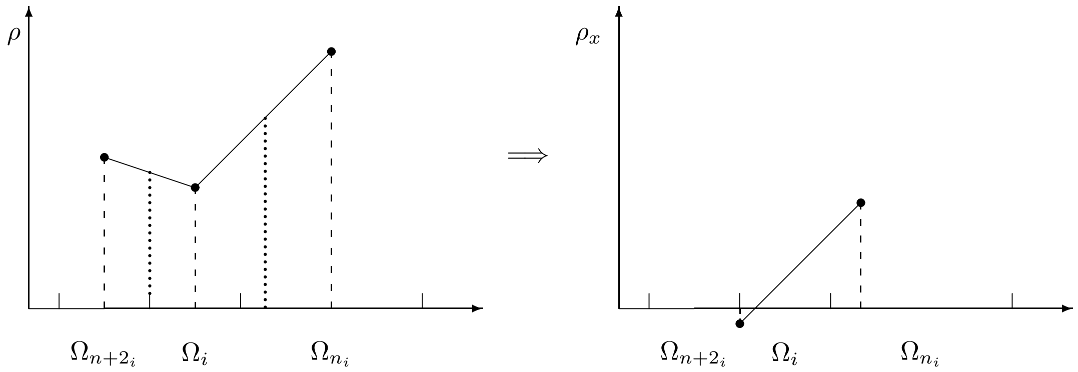


Figure 4.2: Approximation of $\rho_{xx}$ on non-uniform grid. Two approximations of $\rho_x$ first.

Unfortunately this criterion does not work for the first-order adaptive-gridding algorithm. Results (section 5.3) show that the splitting and merging of cells causes small wiggles which have no influence on the solution itself, but cause peaks in the second-derivative estimation (4.10) that are an order bigger than the second derivative of the global solution. So the criterion tends to split cells where other cells have already been split or merged and not where the solution requires it. The criterion suffers from ‘self-induction’.

## 4.3. Error estimate refinement criterion

As seen in the previous section, estimating the truncation error $\tau$ with a second-derivative criterion does not work well for the first-order Euler discretisation. But there is another way to estimate $\tau$, which is both mathematically more correct and less sensitive to wiggles.

The idea was suggested by P.W. Hemker [8] and is also used by Berger e.a. [2, 3]: when a discretisation is first-order accurate, the truncation error is halved when the solution is calculated on a two times finer grid. Therefore, this error can be estimated from the difference between two solutions, on a coarse and a two times finer grid.

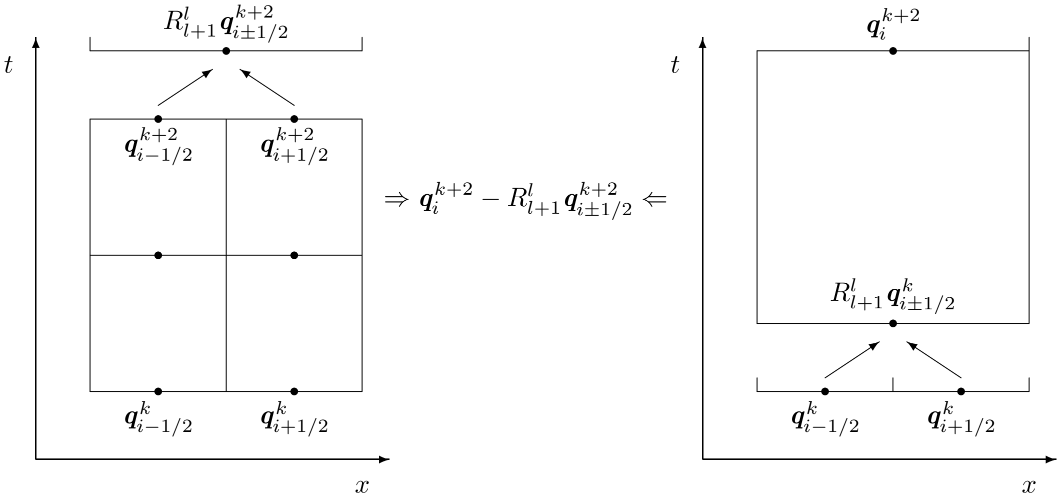

$R^l_{l+1}\,\boldsymbol{q}^{k+2}_{i\pm1/2}$
$t$
$\boldsymbol{q}^{k+2}_{i-1/2}$
$\boldsymbol{q}^{k+2}_{i+1/2}$
$\Rightarrow \boldsymbol{q}^{k+2}_i - R^l_{l+1}\,\boldsymbol{q}^{k+2}_{i\pm1/2} \Leftarrow$
$\boldsymbol{q}^{k}_{i-1/2}$
$\boldsymbol{q}^{k}_{i+1/2}$
$x$
$\boldsymbol{q}^{k+2}_i$
$t$
$R^l_{l+1}\,\boldsymbol{q}^{k}_{i\pm1/2}$
$\boldsymbol{q}^{k}_{i-1/2}$
$\boldsymbol{q}^{k}_{i+1/2}$
$x$

Figure 4.3: $C_{EE}$ refinement criterion (1D example).

This can be seen as follows. Consider a big cell $\Omega^l_{i,j}$ in a Cartesian grid (only for a convenient notation, the result is also valid on a non-uniformly refined grid), on a time level $k$. The cell has sizes $\Delta x = \Delta y = 2h$, $\Delta t = 2ch$ for some constant $c$, and four smaller cells $\Omega_{i\pm1/2,j\pm1/2}$ with $\Delta x = \Delta y = h$, $\Delta t = ch$ on the same location.

Let the exact solution in the four small cells be known. This solution is restricted to the big cell with equation (3.7) and the cell is advanced to level $k+2$. Then the small cells are advanced twice to reach $k+2$ and their states are restricted too (see figure 4.3). The difference with the state in the big cell gives the truncation error:

Develop the exact solution $\bar{\boldsymbol{q}}^k$ in a Taylor series around $\bar{\boldsymbol{q}}^k_{i,j}$, then

$$\bar{\boldsymbol{q}}^k_{i\pm1/2,j\pm1/2} = \bar{\boldsymbol{q}}^k_{i,j} \pm \frac{1}{2}h\left(\bar{\boldsymbol{q}}^k_{i,j}\right)_x \pm \frac{1}{2}h\left(\bar{\boldsymbol{q}}^k_{i,j}\right)_y + \frac{1}{8}h^2\left(\bar{\boldsymbol{q}}^k_{i,j}\right)_{xx} + \frac{1}{8}h^2\left(\bar{\boldsymbol{q}}^k_{i,j}\right)_{yy} \pm \frac{1}{4}h^2\left(\bar{\boldsymbol{q}}^k_{i,j}\right)_{xy} + \mathcal{O}\left(h^3\right). \quad (4.11)$$

Let these $\bar{\boldsymbol{q}}^k_{i\pm1/2,j\pm1/2}$ be known. The restriction $\boldsymbol{q}_{i,j}$ is, with eq. (3.7),

$$\boldsymbol{q}^k_{i,j} = R^l_{l+1}\bar{\boldsymbol{q}}^k_{i\pm1/2,j\pm1/2} = \bar{\boldsymbol{q}}^k_{i,j} + \frac{1}{8}h^2\left(\bar{\boldsymbol{q}}^k_{i,j}\right)_{xx} + \frac{1}{8}h^2\left(\bar{\boldsymbol{q}}^k_{i,j}\right)_{yy} + \mathcal{O}\left(h^3\right). \quad (4.12)$$

Now the cells are advanced. Write the Euler operator $L$ as

$$\frac{\boldsymbol{q}^{k+1}_i - \boldsymbol{q}^k_i}{\Delta t} + F\left(\boldsymbol{q}^k_i\right) = 0,$$

so

$$\boldsymbol{q}_i^{k+1} = \boldsymbol{q}_i^k - chF\left(\boldsymbol{q}_i^k\right). \tag{4.13}$$

As the discretisation is first-order accurate, the truncation error is written as $ha_i^k$, with $a_i^k$ independent of $h$. So

$$\begin{aligned}\bar{\boldsymbol{q}}_i^{k+1} &= \boldsymbol{q}_i^{k+1} - ch^2 a_i^k + \mathcal{O}\left(h^3\right), &(4.14)\\ &= \bar{\boldsymbol{q}}_i^k - chF\left(\bar{\boldsymbol{q}}_i^k\right) - ch^2 a_i^k + \mathcal{O}\left(h^3\right). &(4.15)\end{aligned}$$

The small cells are advanced to level $k+2$ in two steps, so the error in this step is halved,

$$\begin{aligned}\boldsymbol{q}_{i\pm1/2,j\pm1/2}^{k+2} &= \bar{\boldsymbol{q}}_{i\pm1/2,j\pm1/2}^{k+2} + \frac{1}{2}c(2h)^2 a_{i\pm1/2,j\pm1/2}^k + \mathcal{O}\left(h^2\right),\\ &= \bar{\boldsymbol{q}}_{i,j}^{k+2} \pm \frac{1}{2}h\left(\bar{\boldsymbol{q}}_{i,j}^{k+2}\right)_x \pm \frac{1}{2}h\left(\bar{\boldsymbol{q}}_{i,j}^{k+2}\right)_y + \frac{1}{8}h^2\left(\bar{\boldsymbol{q}}_{i,j}^{k+2}\right)_{xx} +\\ &\quad \frac{1}{8}h^2\left(\bar{\boldsymbol{q}}_{i,j}^{k+2}\right)_{yy} \pm \frac{1}{4}h^2\left(\bar{\boldsymbol{q}}_{i,j}^{k+2}\right)_{xy} + 2ch^2 a_{i,j}^k + \mathcal{O}\left(h^3\right). \end{aligned}\tag{4.16}$$

Restricting these yields

$$R_{l+1}^l \boldsymbol{q}_{i\pm1/2,j\pm1/2}^{k+2} = \bar{\boldsymbol{q}}_{i,j}^{k+2} + \frac{1}{8}h^2\left(\bar{\boldsymbol{q}}_{i,j}^{k+2}\right)_{xx} + \frac{1}{8}h^2\left(\bar{\boldsymbol{q}}_{i,j}^{k+2}\right)_{yy} + 2ch^2 a_{i,j}^k + \mathcal{O}\left(h^3\right). \tag{4.17}$$

Now the restricted state in the big cell is advanced. We know that

$$\bar{\boldsymbol{q}}_{i,j}^{k+2} = \bar{\boldsymbol{q}}_{i,j}^k - 2chF\bar{\boldsymbol{q}}_{i,j}^k - 4ch^2 a_{i,j}^k + \mathcal{O}\left(h^3\right). \tag{4.18}$$

Now advance the $\boldsymbol{q}_{i,j}^k$ from equation (4.12):

$$\begin{aligned}\boldsymbol{q}_{i,j}^{k+2} &= \boldsymbol{q}_{i,j}^k - 2chF\boldsymbol{q}_{i,j}^k,\\ &= \bar{\boldsymbol{q}}_{i,j}^k - 2chF\bar{\boldsymbol{q}}_{i,j}^k + \frac{1}{8}h^2\left(\bar{\boldsymbol{q}}_{i,j}^{k+2}\right)_{xx} + \frac{1}{8}h^2\left(\bar{\boldsymbol{q}}_{i,j}^{k+2}\right)_{yy} + \mathcal{O}\left(h^3\right),\end{aligned}\tag{4.19}$$

assuming that $F\left(\bar{\boldsymbol{q}}_{i,j}^k + \mathcal{O}\left(h^2\right)\right) = F\left(\bar{\boldsymbol{q}}_{i,j}^k\right) + \mathcal{O}\left(h^2\right)$. Using (4.18),

$$\boldsymbol{q}_{i,j}^{k+2} = \bar{\boldsymbol{q}}_{i,j}^{k+2} + \frac{1}{8}h^2\left(\bar{\boldsymbol{q}}_{i,j}^{k+2}\right)_{xx} + \frac{1}{8}h^2\left(\bar{\boldsymbol{q}}_{i,j}^{k+2}\right)_{yy} + 4ch^2 a_{i,j}^k + \mathcal{O}\left(h^3\right). \tag{4.20}$$

And therefore,

$$\boldsymbol{q}_{i,j}^{k+2} - R_{l+1}^l \boldsymbol{q}_{i\pm1/2,j\pm1/2}^{k+2} = 2ch^2 a_{i,j}^k + \mathcal{O}\left(h^3\right). \tag{4.21}$$

The terms with $\boldsymbol{q}_{xx}$ and $\boldsymbol{q}_{yy}$ cancel, because the same restriction is used in both cases. To estimate the truncation error, equation (4.21) is divided by $h = \Delta x_i$ to yield an expression that is proportional to $\tau$,

$$C_{EE} \equiv \frac{\left|\boldsymbol{q}_{i,j}^{k+2} - R_{l+1}^l \boldsymbol{q}_{i\pm1/2,j\pm1/2}^{k+2}\right|}{h} = 2cha_{i,j}^k + \mathcal{O}\left(h^2\right) = 2c\tau + \mathcal{O}\left(h^2\right). \tag{4.22}$$

The procedure to determine this refinement criterion is:

- Start with a refined grid, consider one level $\Omega^l$.
- Make a second grid, 1 level coarser (the ‘sister’ grid).
- Restrict the solution on the fine grid to the coarse grid.

- Advance the fine grid twice and the coarse grid once.
- Restrict again the solution on the fine grid to the coarse grid.
- Compare this restriction with the solution on the coarse grid.

This is done for all levels, so at each level a refinement criterion is available every two time steps $\Delta t_i$ (as opposed to the previous criteria that were available every step).

This procedure causes a number of problems: first, a coarse grid cannot be constructed for any fine grid. Like with unrefinement (section 3.2), sometimes one of the fine cells is refined itself, so the coarse grid does not fit in.

This problem is solved by basing the grid geometry not on the fine grid, but on the coarser sister grid and refining *this* grid (figure 4.4). When a coarse cell is refined, its four finer sister cells are split too, in 16 cells. This is a natural thing to do, as $C_{EE}$ is determined in every *coarse* cell, but it means that more fine cells are refined than strictly necessary.

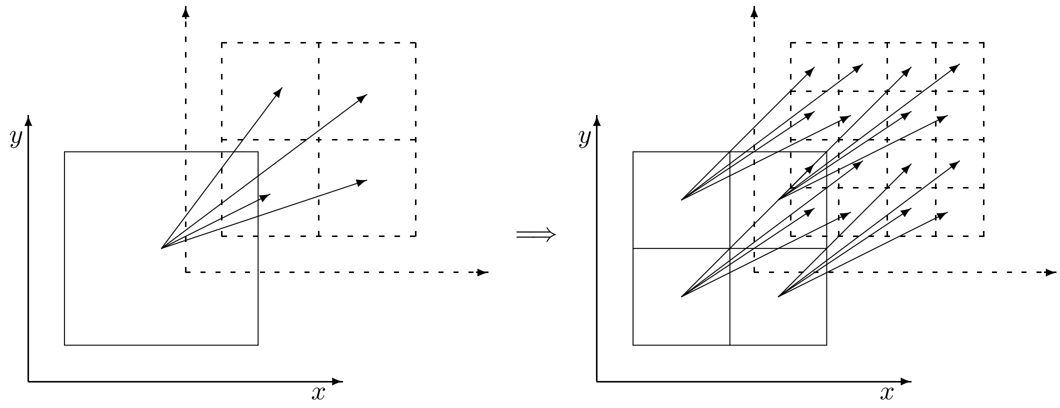


Figure 4.4: Two grids are used, a coarse 'sister' grid (drawn lines) and the finer main grid (dashed). Each coarse cell has four sisters.

Another problem is that $C_{EE}$ works well as an unrefinement criterion (keep the fine cells unless the truncation error gets so low that they can be merged), but it is not so suitable as a refinement criterion: when the truncation error is too high in one fine step, the grid is refined, but only for the *next* time step. Since the current algorithm cannot go back and do a time step again on a finer grid, we need something that predicts when $C_{EE}$ will become higher than $d_s$ and splits the cells in time. Then $C_{EE}$ is used to determine if this refinement was really necessary.

The increase of $C_{EE}$ between two time steps is used for this prediction. When $C_{EE}$ increases more than a certain value $d_{s\triangle}$ in one time step, the coarse cell is split. As an extra, a $C_{\partial\rho}$ criterion with a very high $d_s$ is used to spot discontinuities in the initial solution (like in Riemann problems) and refine quickly there.

Results show that the $C_{EE}$ criterion works, but it is far more sensitive to small disturbances than the $C_{\partial\rho}$ criterion.

## 4.4. Buffers

To ensure a properly functioning algorithm, it is necessary to split more cells than just those that have a high enough refinement criterion. These cells are called buffer cells. The different types of buffers are described here.

*Two cells per level* It is desired that two neighbour cells have a maximum level difference of one, jumps of two or more levels are not allowed. This is both more accurate (extra errors are introduced where neighbour cells have different levels and these errors become larger when the level difference is 2 or more) and allows simpler routines in the computer program. But we cannot decide not to refine cells that have a refinement criterion higher than $d_s$.

When a cell on level $l$, that has a neighbour on level $l-1$, is refined to level $l+1$, a level jump of 2 appears. But this jump can be easily removed by refining the neighbour cell too, to level $l$. Problems appear only when $l$ is the lowest level that may be split at that moment (figure 3.5b) as, in that case, it is no longer possible to split a cell at level $l-1$ if a level jump appears. But as all flow phenomena travel at a maximum rate of 1 cell per time step (the choice of $\Delta t$ ensures this, section 3.1), the zones with refined cells travel at the same speed.

So when a level $l$ that can be split is made at least 2 cells wide, it never disappears completely: after the next time step for level $l+1$, at most one of the 2 cells of level $l$ is refined to level $l+1$, after the second time step level $l$ can be split again and a new buffer of 2 cells is formed.

One extra buffer is needed: when a cell at level 0 is refined, all its neighbours are refined as well. This is needed when a new wave appears somewhere, probably at $t=0$. At that moment, cells in and around the wave are refined from level 0 to level 1. But the wave is new, so there are no level 2 cells yet to ensure a two-cell buffer of level 1 cells. Also, the direction in which the wave is going to travel is not yet known. Therefore a buffer of 2 cells in all directions is made by refining the neighbours of cells to be refined as well.

***$C_{EE}$* buffers** The error estimate criterion is rather sensitive to refinements and unrefinements. Therefore, two extra buffers are used in the final version:

1. A buffer with a width of one cell around any cell on the highest level $M$ that still has $C_{EE} > d_s$. These cells cannot be split, but it must be ensured that they never come close to cells on a lower level.

2. A buffer in the unrefinement procedure: only cells that have all neighbours on a lower level, or with $C_{EE} < d_m$, may be unrefined. This means that cells are only unrefined when they are in a relatively large area with a low $C_{EE}$, so they will not be split again the next time step. This is highly undesirable, because it introduces large wiggles in the solution and thus unnecessary refinement of the grid.

# Chapter 5
# Numerical results

In this chapter, numerical results are presented for the first-order adaptive-gridding solver. The performance of different refinement criteria is compared. To do this, two test problems are chosen, defined in section 5.1. The tests are calculated with a Fortran 95 research code [22].

## 5.1. Test problems

Two different test problems are used. The first is a one-dimensional case. The solution of this problem is known analytically, therefore it is very useful to compare the performance of different refinement criteria. The second is a 2D problem with a lot of structure in the solution, suitable for evaluating the capabilities of the solver for truly 2D problems.

### *5.1.1 The Sod problem*

This 1D problem is a Riemann problem, see section 2.3: the flow of gas in a shock tube is studied, at $t = 0$ this gas has two states, one to the left and one to the right of $x = 0$. The Sod problem is used by Toro [18] to test numerical Riemann solvers, but it is also suitable as a test problem for Euler solvers, because its solution consists of a shock, a contact discontinuity and an expansion wave. All waves that appear in Riemann problems are there.

| | $x < 0$ | $x > 0$ |
|---|---|---|
| $\rho$ | 1.000 | 0.125 |
| $u$ | 0.000 | 0.000 |
| $v$ | 0.000 | 0.000 |
| $p$ | 1.000 | 0.100 |

Table 5.1: Sod problem, initial values.

The initial states are given in table 5.1. Both states are at rest, but the pressure and density are higher to the left. The gas has $\gamma = 1.4$. The result is a left-running expansion, a right-running contact discontinuity and a right-running shock.

In 2D, this problem is modelled as a real shock tube, a many × 1 cell rectangular domain with four boundaries. The left boundary is subsonic inflow, the right boundary is subsonic outflow. (This is unimportant, by the way, because the problem is stopped before the waves reach the boundaries). The upper and lower boundaries are walls. The tangential velocity component $v$ is calculated, but it must be zero everywhere. This sounds trivial, but it is a good check for the correctness of the computer code.

As a reference for the solutions on adapted grids, the problem is solved on uniform grids, ranging from 50 cells to 800 cells. The solution at $t = 0.1$ for 50, 200 and 800 cells is given in figure 5.1. A value of 4.0 is chosen for $\Delta x/\Delta t$. With $|\lambda|_{max} = 2.19157$ for the entire problem, this corresponds to a CFL number (3.1) of 0.55.

Note that the shock is very sharp, this is usual for Godunov-type flux functions. The contact discontinuity (only visible in $\rho$) and the head of the expansion are smeared more. The first-order discretisation causes a high amount of numerical diffusion, which tends to smear the flow phenomena.

To compare the solutions, define a global error measure for each state component $p$ as

$$e_p = \frac{\sum_i |R_i \bar{q}_p - q_{i,p}| \, 2^{-2l_i}}{N^*}, \tag{5.1}$$

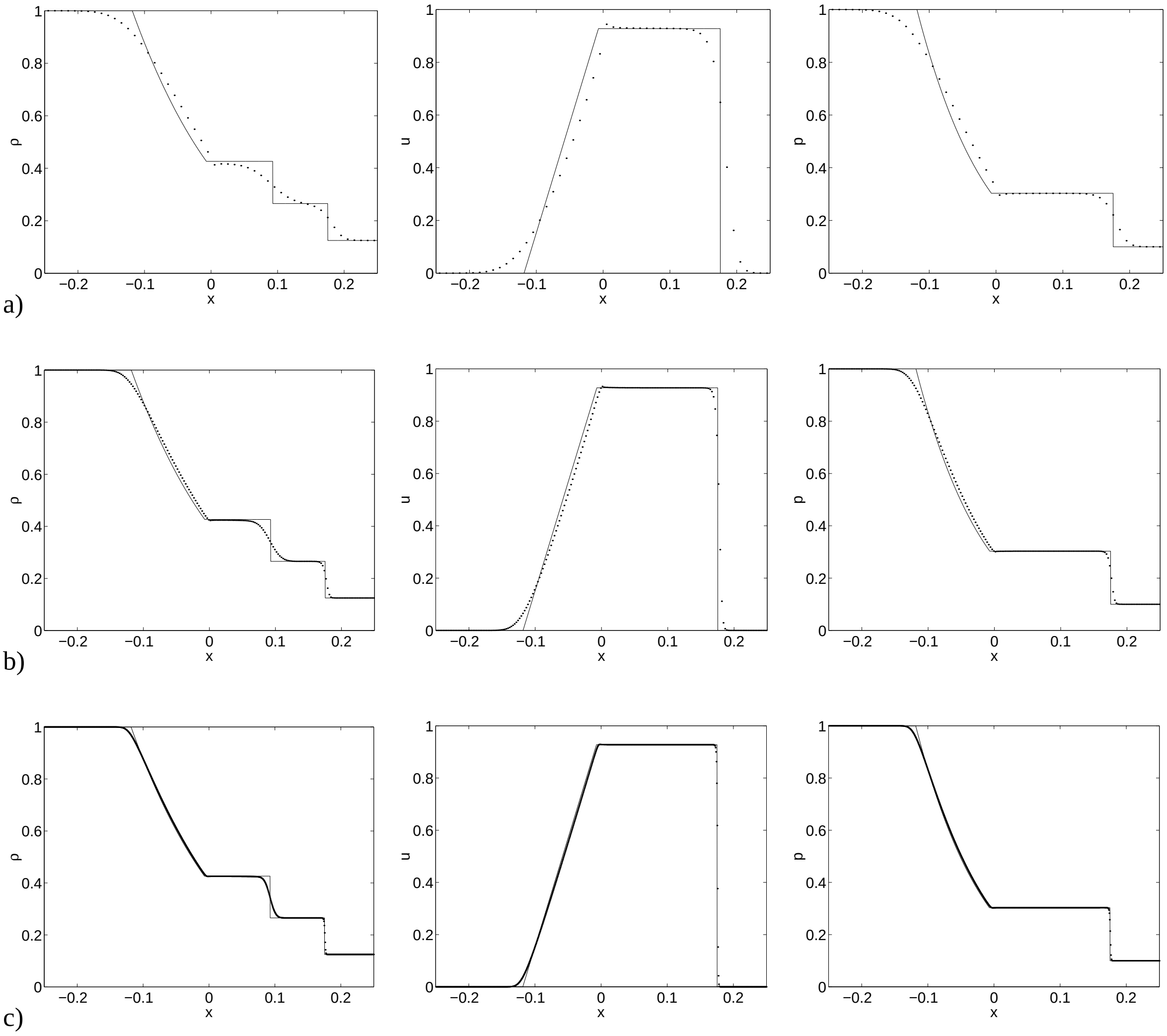


Figure 5.1: Sod problem, solution on uniform grids with 50 (a), 200 (b) and 800 (c) cells. Domain $x = [-0.25, 0.25]$, $t = 0.1$, $\Delta x/\Delta t = 4.0$. Solid lines give the exact solution.

with $R_i\bar{q}_p$ the restriction of the exact solution to the grid (see section 2.5) and $N^*$ the number of cells in the basic grid. These errors are given in table 5.2. The order of convergence is clearly not 1, but this is expected for a problem with discontinuities. The actual order lies around 0.6.

### *5.1.2 A wind tunnel with a forward-facing step*

This 2D problem was introduced by Emery [5] to compare finite-difference schemes and it was later used by Woodward and Colella [24] in their comparison of Euler solvers.

The problem starts with a wind tunnel section, 3 units long and 1 unit high, with a Mach 3 uniform flow ($\gamma = 1.4$, $\rho = 1.4$, $u = 3.0$, $v = 0.0$, $p = 1.0$). At $t = 0$, a forward-facing step materializes in the tunnel, with a height of 0.2 units, that starts at 0.6 units from the inflow boundary.

A shock wave develops in front of the step, it continues above the step. After some time, it reflects from the upper boundary and interacts with the expansion fan that develops above the step. When it reaches the lower wall, it reflects again. The reflection at the top develops into a Mach stem with a normal shock and a contact discontinuity. Figure 5.3 shows a ‘golden’ solution for the density at $t = 4.0$, obtained by Woodward and

| Cells | 'Level' | $e_\rho$ | $e_u$ | $e_p$ |
|---|---|---|---|---|
| 50 | 0 | 0.0250 | 0.0458 | 0.0230 |
| 100 | 1 | 0.0167 | 0.0283 | 0.0143 |
| 200 | 2 | 0.0106 | 0.0161 | 0.0086 |
| 400 | 3 | 0.0068 | 0.0092 | 0.0052 |
| 800 | 4 | 0.0043 | 0.0053 | 0.0031 |

Table 5.2: Sod problem, errors in the solution on uniform grids, $t = 0.1$. The number of time steps is set from $\Delta x/\Delta t = 4$. The 'level' indicated is the level of the grid cells, if they are created by splitting a 50 cell basic grid.

Colella [24] with their best solver on a fine grid.

The problem is modelled here with basic grids that have $\Delta x$ and $\Delta y$ the same. Boundary conditions are supersonic inflow on the left boundary and supersonic outflow on the right, all the other boundaries are walls. Special treatment of the upper corner of the step is needed, because the point above the step is a singular point of the solution. Therefore, the influence of numerical viscosity is very strong here: without special measures a flow pattern appears on the upper side of the step that resembles a boundary layer. This 'numerical boundary layer' has a strong influence on the reflection of the oblique shock, changing the entire flow pattern.

So, following Woodward and Colella, a correction is used to minimize the effect of the singular point: in the first four cells on the first row above the step and in the first two cells on the next row, the density is reset after every time step to give the same entropy $z_{\text{ref}}$ as in the cell just below and left of the corner (see figure 5.2). In the same cells, the magnitude of the velocity is reset to give the same total enthalpy $H_{\text{ref}}$. So for each of the six cells $\Omega_i$,

$$\begin{aligned}
p_{i,\text{new}} &= p_{i,\text{old}}, \\
\rho_{i,\text{new}} &= \exp\left(\frac{\ln\left(p_{i,\text{old}}\right) - z_{\text{ref}}}{\gamma}\right) && \text{(same entropy)}, \\
u_{i,\text{new}} &= u_{i,\text{old}}\sqrt{\frac{H_{\text{ref}} - \frac{\gamma}{\gamma-1}\frac{p_{i,\text{new}}}{\rho_{i,\text{new}}}}{\frac{1}{2}\left(u_{i,\text{old}}^2 + v_{i,\text{old}}^2\right)}} && \text{(same total enthalpy)}, \\
v_{i,\text{new}} &= v_{i,\text{old}}\sqrt{\frac{H_{\text{ref}} - \frac{\gamma}{\gamma-1}\frac{p_{i,\text{new}}}{\rho_{i,\text{new}}}}{\frac{1}{2}\left(u_{i,\text{old}}^2 + v_{i,\text{old}}^2\right)}} && \text{(same total enthalpy)}.
\end{aligned} \tag{5.2}$$

The conservative state vector in $\Omega_i$ is set with these values of $p$, $\rho$, $u$ and $v$.

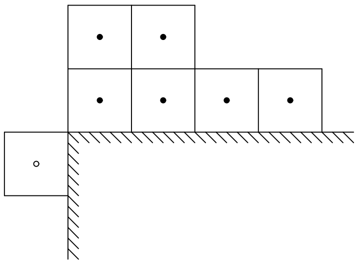


Figure 5.2: Correction above the step for forward-facing step problem. •: cell that is changed, ∘: reference cell.

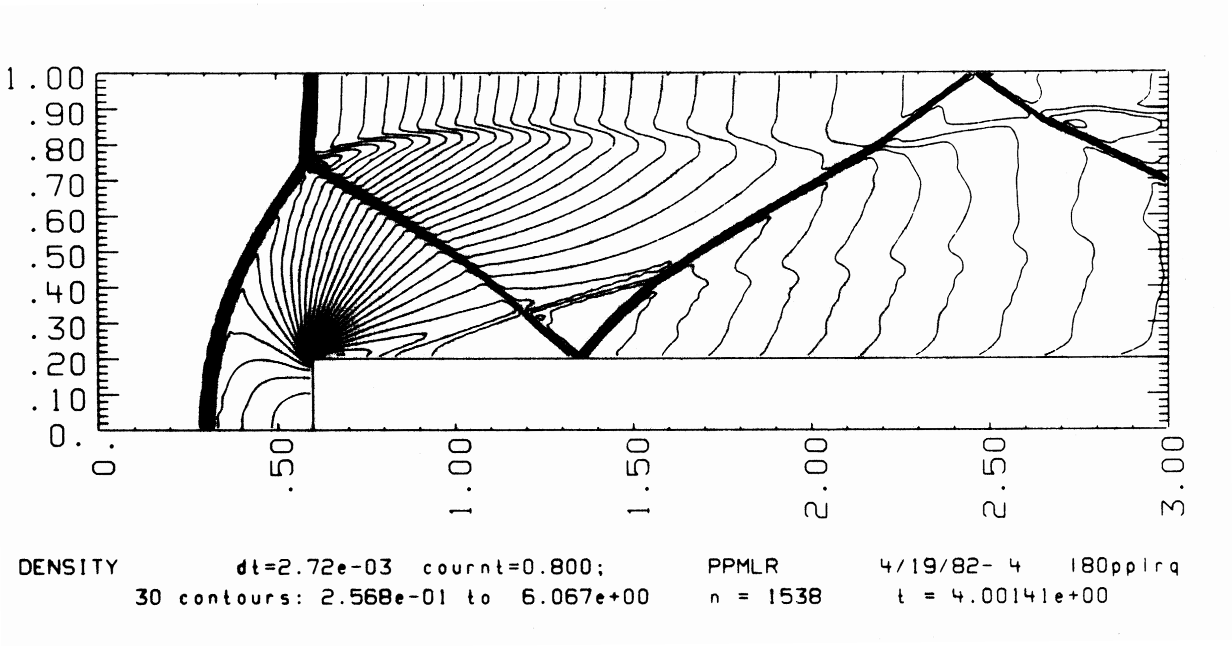


Figure 5.3: Forward-facing step, 'golden' solution, from [24].

As a reference, solutions for the density $\rho$ are presented in figure 5.4 on uniform grids with $\Delta x = 1/20$, $1/40$, $1/80$ and $1/160$. $M$, $p$ and $z$ are given for the finest grid in figure 5.5. To get a better understanding of the effects of adaptive gridding, later on, it is interesting to study these solutions in detail.

The main features of the flow at $t = 4$ are a bow shock in front of the step, that is reflected three times, and an expansion fan originating from the step. The first shock reflection has developed into a Mach stem: the oblique shocks join a normal shock from the wall. This normal shock is stronger than the two oblique shocks combined, the flow behind the normal shock is subsonic, while the flow behind the oblique shocks is still supersonic (figure 5.5a). Therefore, a contact discontinuity originates from the meeting point of the shocks. This contact discontinuity is visible in the 'golden' solution for the density (figure 5.3) and in the entropy solution (figure 5.5c). Like the contact discontinuity in the Sod problem, it is smeared in the density plots. Note that the pressure plot 5.5b shows no sign of the wave, so it is a true contact discontinuity.

Another interesting feature can be seen behind the expansion. For numerical reasons, a slight overexpansion occurs in the expansion fan, that is corrected with a weak oblique shock. The 'golden' solution shows how this shock interacts with the reflected shocks: it crosses the first and merges with the second, causing a (very weak) contact discontinuity. The oblique shock is resolved on the finer grids of figure 5.4, but the contact discontinuity is smeared completely. The contact discontinuity from the Mach stem interacts with the reflected shocks too, changing their direction notably.

Several errors are visible in the solutions 5.4. The Mach stem is not resolved on the coarsest grids and is too far back and too small on the finer grids. Also, despite the correction (5.2), a viscous layer appears above the step (especially well visible in figure 5.5c) and this layer interacts with the reflecting shock. The resulting pattern is probably an incorrect second Mach stem (note the subsonic area behind the reflection).

A smaller error type are the 'wiggles' visible behind the normal shock on the two finer grids. Van Leer [12] explains this as a mesh effect: when the normal shock moves to the left, it crosses the cell face between left and right neighbour cells sometimes. When this happens and the shock is not exactly perpendicular to the wall, then pairs of cells appear behind the shock where the upper cell is part of the shock, but the lower cell is not or vice versa. Obviously, the state in these two cells is different, so vertical flow starts between these cells and it continues for some time after the shock has crossed the cell face. This causes the wiggles.

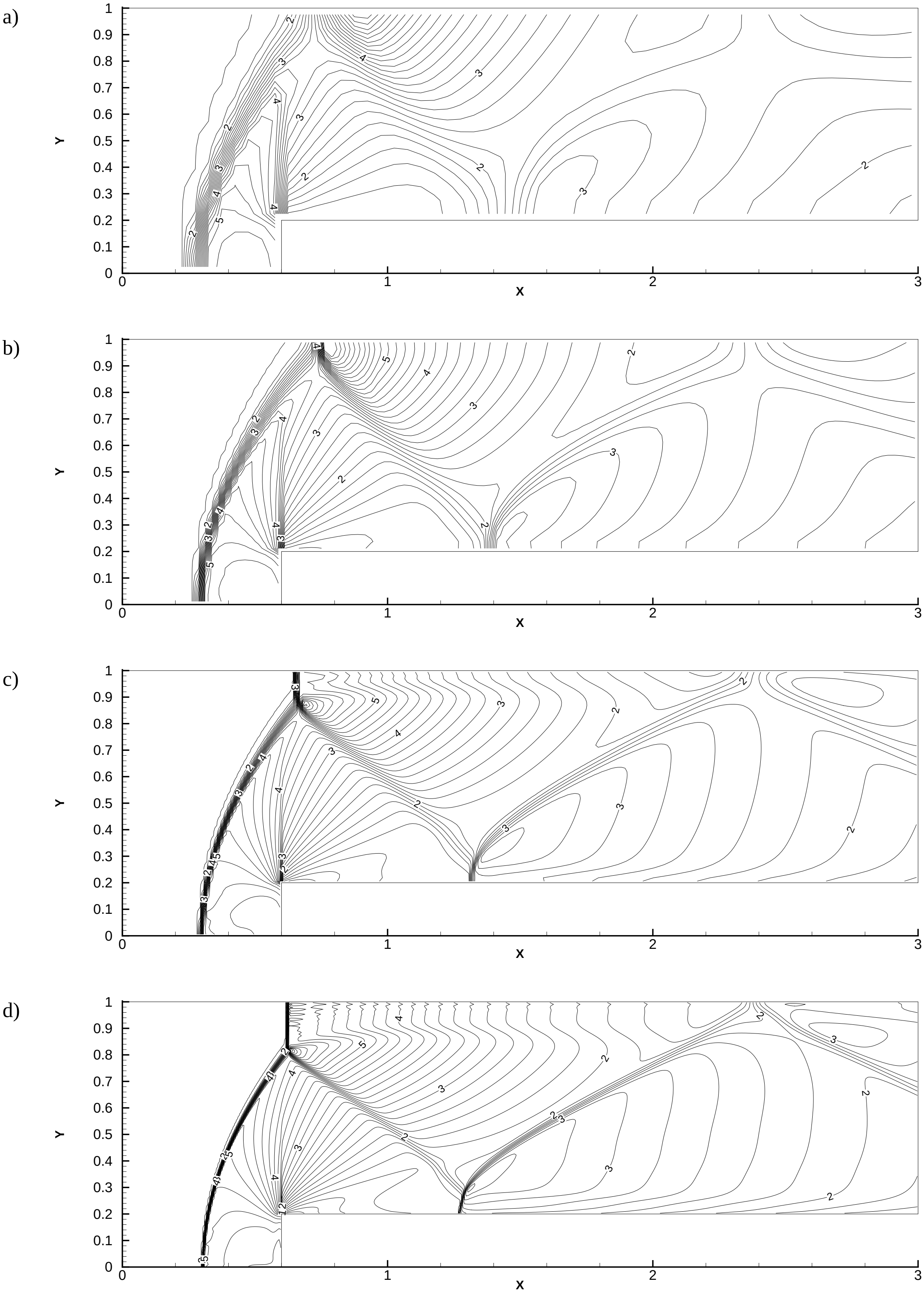


Figure 5.4: Forward-facing step, solution on uniform grids. Density $\rho$ at $t = 4.0$. Iso-lines, step 0.2 between the levels. $\Delta x = 1/20$ (a), $1/40$ (b), $1/80$ (c) and $1/160$ (d), $\Delta x/\Delta t = 6.25$.

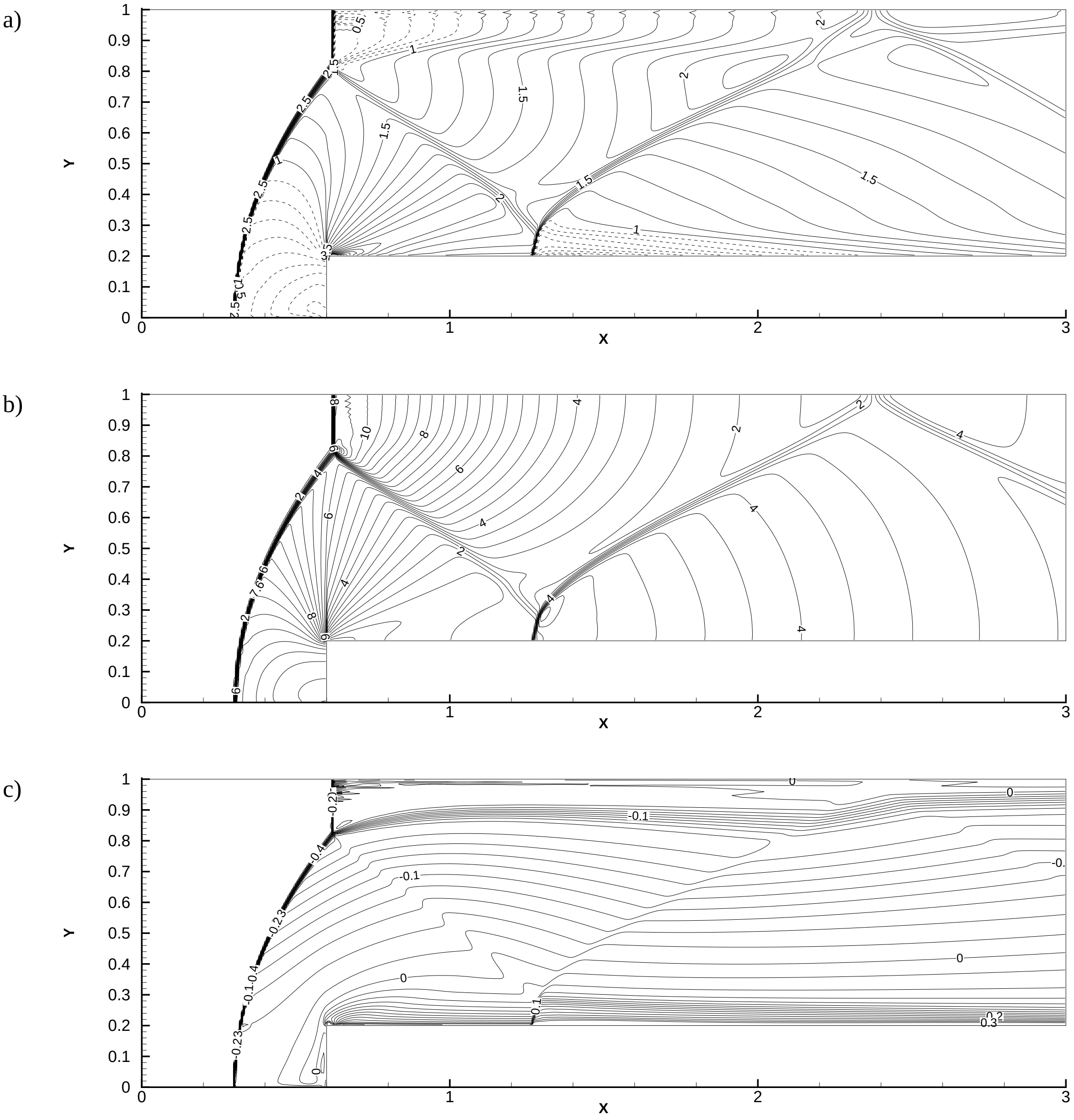


Figure 5.5: Forward-facing step, solution on uniform grid $\Delta x = 1/160$. Iso-lines for Mach number $\frac{\sqrt{u^2+v^2}}{c}$, 0.1 between the levels (a), pressure $p$, 0.4 between the levels (b) and unscaled entropy $z$, 0.02 between the levels (c).

The last error is the shock-like structure right above the step. Woodward and Colella observe the same error for their Godunov solution and call it an ‘expansion shock’. However, it is *not* a shock: it is not visible in the entropy solution 5.5c. According to Van Leer, it is an error that Riemann solvers make when they encounter a sonic point at $x = 0$. For the first-order discretisation, the input states $\boldsymbol{q}_L$ and $\boldsymbol{q}_R$ are equal to the states in the cells, which are either subsonic or supersonic, therefore the sonic points are *always* on the cell faces and this error appears always when a sonic line is aligned with the grid. The Mach plot 5.5a shows that the sonic line lies in the same position as the error in the density.

To show the dramatic effects of omitting the correction (5.2) at the step, one solution is given without this correction (figure 5.6). The entire flow pattern is changed: the first lower reflection is changed to an X-shaped shock and all shocks have moved. This effect does not disappear on finer grids, as may be expected for numerical phenomena. Contrarily, it is less pronounced on *coarser* grids. The phenomena are caused by numerical viscosity and they resemble the $\lambda$-shocks from viscous flow. The flow in figure 5.6 can be compared with e.g. figure 278 in Van Dyke [20].

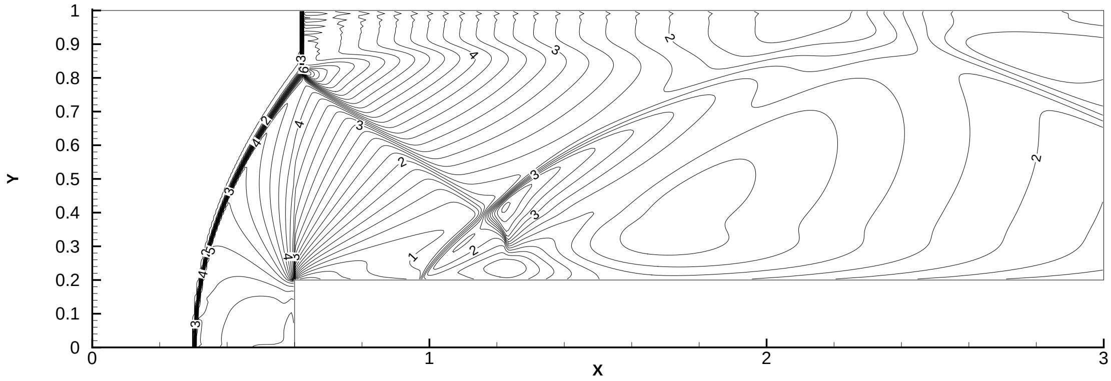


Figure 5.6: Forward-facing step, solution on uniform grid $\Delta x = 1/160$, iso-lines for density $\rho$. This solution was calculated without the correction (5.2) at the step corner.

To compare the computational costs for the solutions, CPU times are measured on a SUN E250, see table 5.3. The code used is the adaptive-gridding code, but on only one level, so there is no overhead time for refinement checks. Each solution requires about 8 times more CPU time than the solution on a two times coarser grid. This is logical, as a grid with a two times smaller $\Delta x$ has four times more cells and twice more time steps.

| $\Delta x$ | ‘Level’ | Cells | CPU time |
|---|---|---|---|
| 1/20 | 0 | 1008 | 13 s |
| 1/40 | 1 | 4032 | 104 s |
| 1/80 | 2 | 16128 | 833 s |
| 1/160 | 3 | 64512 | 6802 s |

Table 5.3: Forward-facing step, CPU times for the solution on uniform grids (figure 5.4). The ‘level’ indicated is the level of the grid cells, if they are created by splitting a $\Delta x = 1/20$ basic grid.

## 5.2. Gradient $\rho$ refinement criterion

The gradient $\rho$ refinement criterion, described in section 4.1, is tested here. The criterion is based on a finite-difference approximation to the first spatial derivative of the density.

*Sod problem* The first test is done on the 1D Sod problem. The basic grid is a $50 \times 1$ grid, as described in the previous section, but the solutions are calculated with the normal adaptive-gridding code, which refines cells in two dimensions. Therefore, when cells are refined, the grid gets two cells in $y$-direction, or four, eight etc. As the problem is one-dimensional, all cells with the same $x$-coordinate must have the same state and the same level, which is a good test for the code.

Because of this, no CPU times are recorded, as it is unfair to compare an $800 \times 1$ uniform grid with a refined grid with, say 150 cells in $x$-direction, but maybe 3000 cells totally. Therefore, the size of the problem is expressed as the number of cells in $x$-direction at $t = 0.1$. This value is not truly representative for the amount of computation needed, but gives some indication.

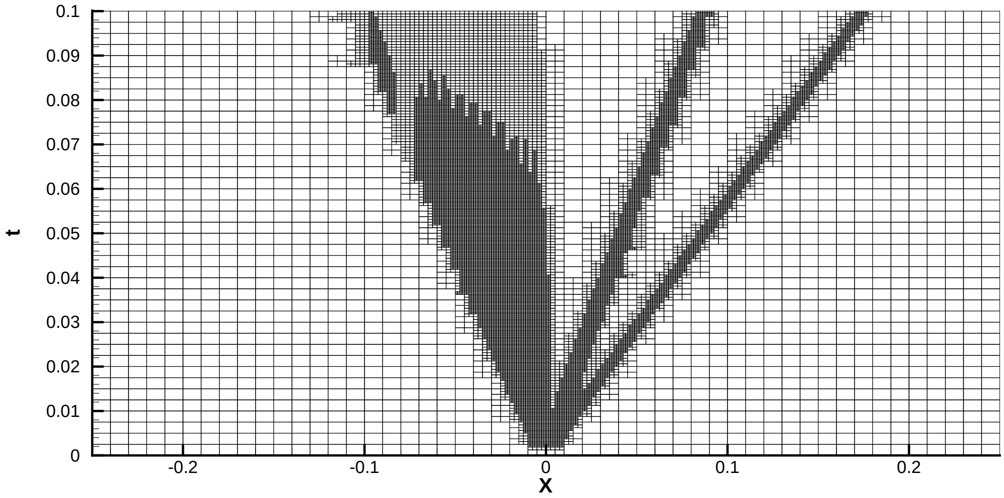


Figure 5.7: Sod problem, $C_{\partial\rho}$ criterion. An example of the grid for the entire problem, $d_s = 0.4$, $d_m = 0.16$, maximum level $M = 3$.

In figure 5.7, the development of the grid in time is shown for one case. We see here that the refined area starts around $x = 0$, that it becomes broader and how the refined areas for the three waves separate. From that moment, the pattern of refined cells in the shock and the contact discontinuity remains the same, but more cells are refined in the expansion when it becomes broader. As the change in the state quantities over the expansion remains the same, the expansion becomes less steep when it spreads. Therefore, the cell level of refinement is reduced in this case, shortly before $t = 0.1$.

When all this is taken into account, we see that the grid at $t = 0.1$ gives a good indication of the shape of the grid in the entire problem. Therefore, the grids at $t = 0.1$ are given for all cases in the remainder of this report.

Grid levels of the solutions for different maximum levels and different values of $d_s$ are shown in figure 5.8. It is obvious at once that the grids are smooth: the level increases from 0 to a maximum level and back, without ‘wiggles’.

The shock usually has cells at the highest level, except for the highest value of $d_s$. Shocks are self-steepening: they become sharper in time without any specific numerical treatment. So when a shock has cells on some level, it steepens and if $d_s$ is low enough, the middle of the shock reaches a slope higher than $d_s$. Then the cells are refined and the shock steepens more, until $C_{\partial\rho}$ is again higher than $d_s$ etc.

The contact discontinuity is also resolved on either the highest or the lowest grid level, although it is not self-steepening. This is probably determined in the first few iterations: if $d_s$ is high, the contact discontinuity is smeared at once and stays smeared, so it does not require refined cells, but when it is resolved on refined cells in the first steps, it stays sharp and thus keeps the fine cells. The examples of figure 5.8 show only contact discontinuities with cells at level 0 or $M$, but other solutions have discontinuities on intermediate levels. The range of $d_s$ in which this happens is very small.

The expansion has a slope that is determined physically, so the level of the expansion is determined only by $d_s$ and not by $M$ (if $M$ is high enough). The expansion level is 1 for $d_s = 8.0$, 2 for $d_s = 4.0$ and probably 4 for $d_s = 1.0$. This cannot be checked because no calculations were made with $M = 5$ or more, so the

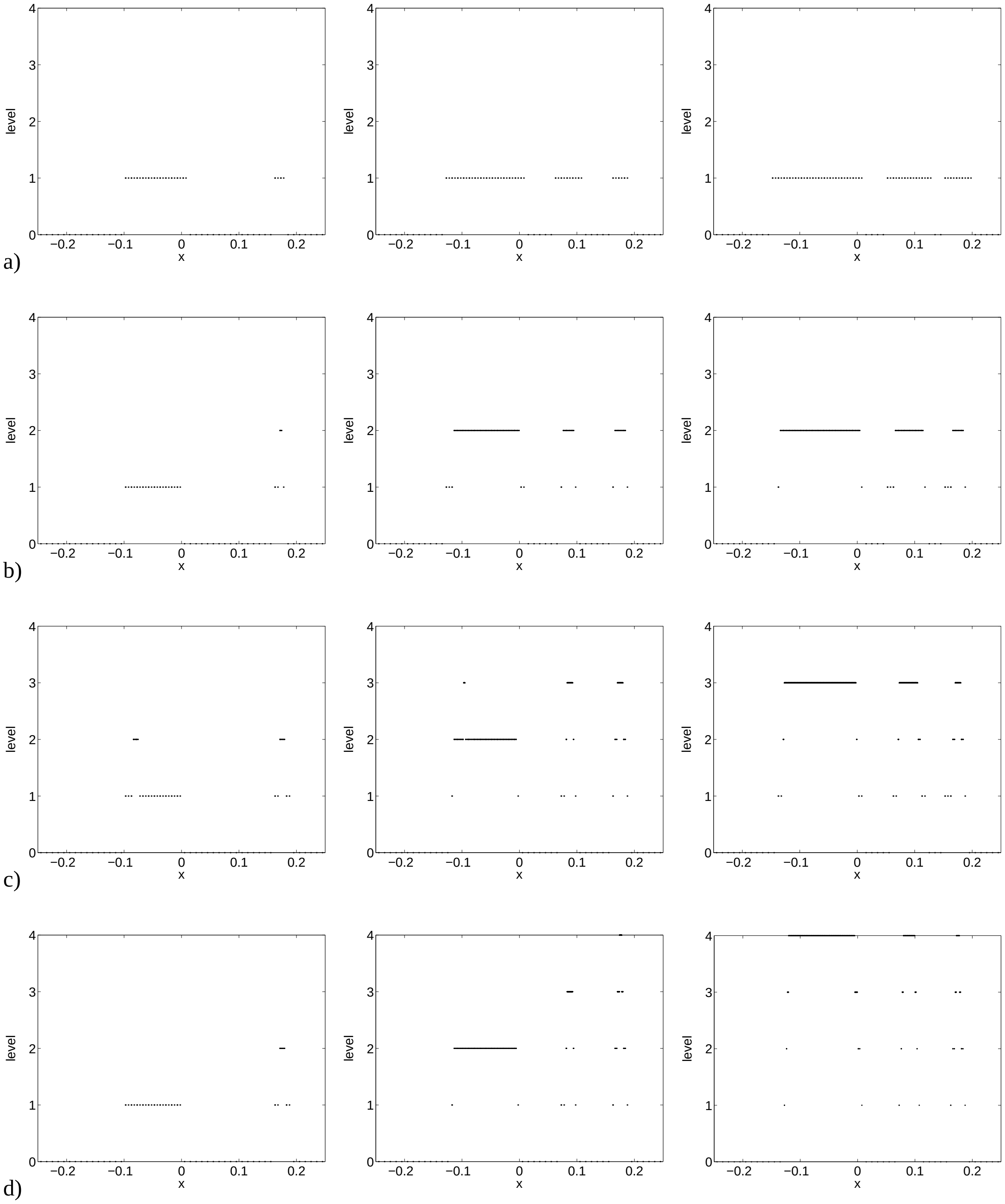


Figure 5.8: Sod problem, $C_{\partial\rho}$ refinement criterion. Cell level distributions with different settings of $d_s$: 8.0, 4.0 and 1.0. $d_m = 0.4d_s$. Maximum level 1 (a), 2 (b), 3 (c) and 4 (d).

expansion level is always $M$ for $d_s = 1.0$.

Plots of the solutions for selected tests (figure 5.9) show an accuracy comparable with the solutions on uniform grids, only the 'bump' in the solution at the end of the expansion is bigger.

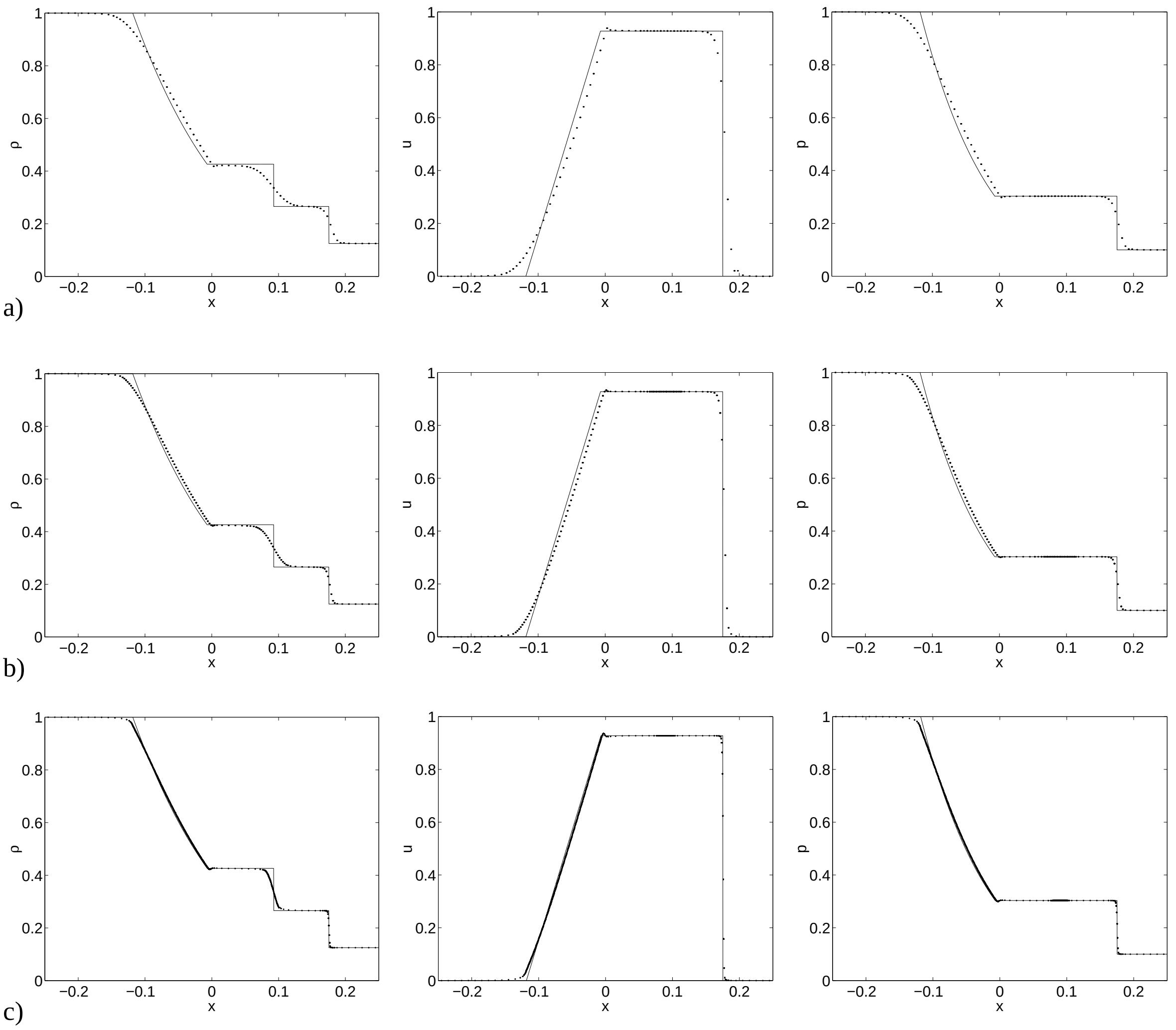


Figure 5.9: Sod problem, $C_{\partial\rho}$ refinement criterion. Solution for maximum levels 1 (a), 2 (b) and 4 (c). $d_s$ = 1.0, $d_m$ = 0.4. Domain $x = [-0.25, 0.25]$, $t = 0.1$, $\Delta x/\Delta t = 4.0$.

Errors as calculated with equation (5.1) are given in figure 5.10. It is obvious that the refinement criterion works, the same accuracy is reached with far less cells. If we look at the solid lines, that show the error on different levels for the same setting of $d_s$, we see that the error converges at least at the same rate as for the uniform grids, even a bit faster. For the higher $d_s$, the error drops rapidly for higher levels. This happens because the expansion is not refined above a certain level at the end, so the number of cells increases only a bit with the maximum level, while for higher $M$ the grid is refined more in the beginning when the expansion is steeper, thus reducing the errors. So here it is not entirely fair to use the number of cells at $t = 1$ to compare the solutions: those with higher maximum levels do require more computation in the beginning.

Looking at the dotted lines, the error per level, we see that this error converges excellently towards the error for the same level on a uniform grid, we can come close to these errors by setting $d_s$ sufficiently small. But as

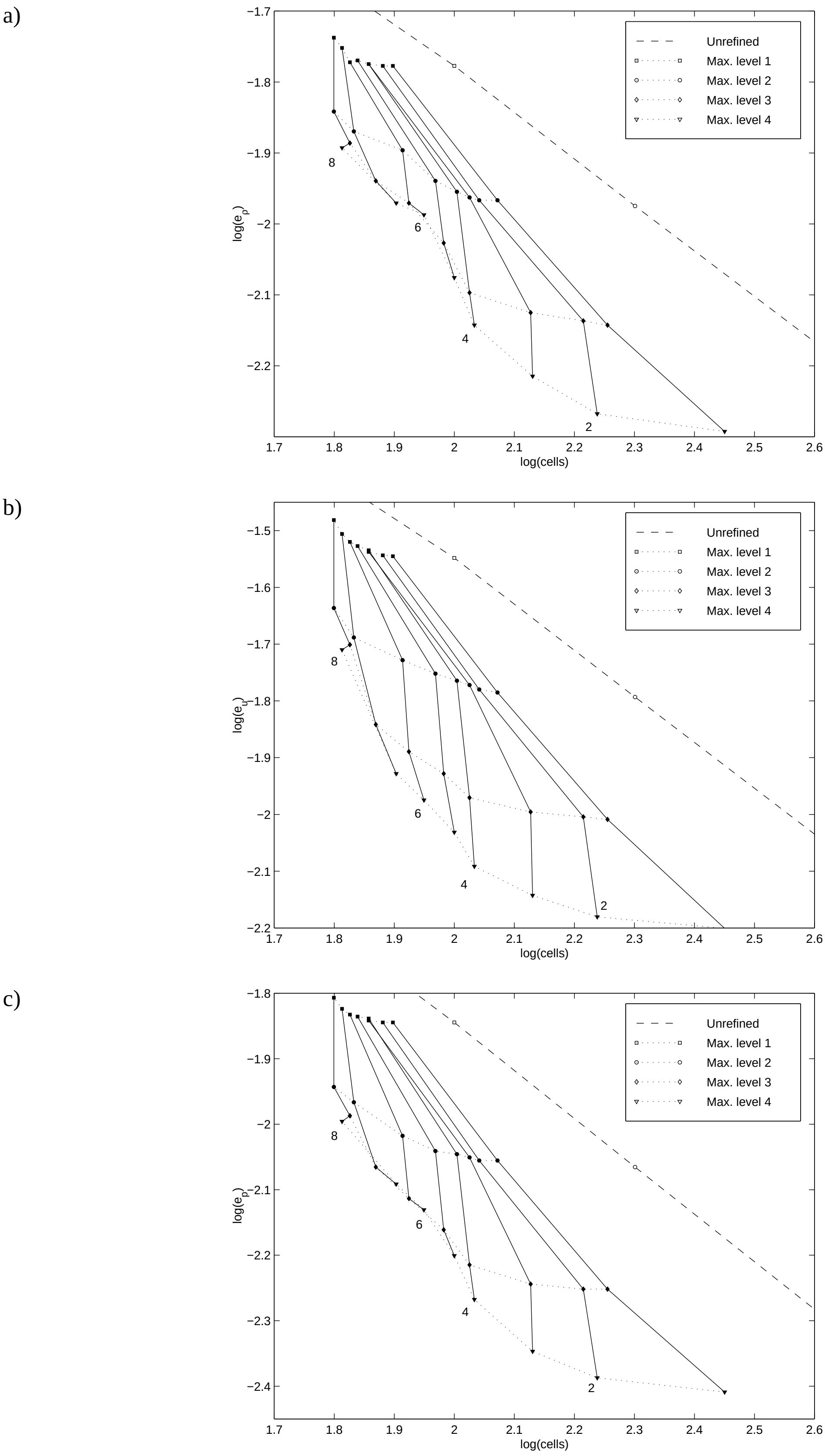


Figure 5.10: Sod problem, $C_{\partial\rho}$ refinement criterion. Errors in $\rho$ (a), $u$ (b) and $p$ (c). Dashed line gives the errors for uniform grids, solid lines the errors for $d_s = 1 \dots 8$ (indicated on the plots). Note the double-log scales.)

the last part of the convergence is slow, it is probably more efficient to choose a higher value of $d_s$ and a higher maximum level to reach the same errors.

The ratio between $d_s$ and $d_m$ is kept constant at 2.5. Experiments have shown that all ratios between 2 and 3 give almost the same results. Only lowering the ratio below 2 causes 'wiggles' in the level distribution and larger errors.

A last remark, the effect of the value of $d_s$ depends on the flow problem. If a problem has stronger shocks than the Sod test problem, the same types of grids may occur for higher values of $d_s$.

*Forward-facing step problem* The second test case was run with one value for $d_s$. $d_s = 4.0$ is chosen, because this value gives waves with cells almost at the maximum level for the Sod problem. As the shocks in the forward-facing step problem are a little stronger, this choice works well.

Results for three maximum levels are given in figures 5.11 and 5.13. Comparing them with the solutions on uniform grids (figures 5.4 and 5.5) shows that the results are excellent: shocks are in the same positions, they are as sharp as on the uniform grids and (almost) all details are resolved. Only the oblique shock on the level 3 grid is not resolved and a few bumps appear in the iso-curves. They are caused by the plotting program that has to cope with changes in cell size. On the other hand, the wiggles behind the Mach stem have almost disappeared.

Looking at the refined grids (figure 5.12), we see that the refinement criterion recognizes all the flow features. The shocks, the expansion fan and the contact discontinuity are resolved. Even the incorrect second Mach stem that showed up in the uniform results is refined. Note also the continuation of the contact discontinuity in the upper right corner on the finest grid: it is not visible in the $\rho$ plots, but the location of the refined grids corresponds to the position of the contact discontinuity as found by Woodward and Colella and to the enthalpy solution 5.13c.

So the solutions on refined grids are comparable in accuracy with those on uniform fine grids. The CPU times required are much lower, even with the extra costs for grid refinement and unrefinement. A reduction of almost five times is achieved on the finest grid (table 5.4).

| Max. level | CPU time | % of unif. |
|---|---:|---:|
| 1 | 63 s | 61% |
| 2 | 333 s | 40% |
| 3 | 1479 s | 22% |

Table 5.4: Forward-facing step, CPU times for the solution on refined grids, $C_{\partial\rho}$ refinement criterion (figure 5.11). The last column gives the CPU time, divided by the time for the solution on a uniform grid on the maximum level.

## 5.3. Second derivative $\rho$ refinement criterion

The $C_{\partial^2\rho}$ refinement criterion is based on a finite-difference approximation of $\frac{\partial^2\rho}{\partial x^2}$ (section 4.2). It was expected that this criterion would refine the grid only in places with a large curvature in the solution. But it appears that the criterion is sensitive to small wiggles in the solution, caused by the refining and unrefining of grids and by the extra errors where grids with different sizes meet.

The result is that the criterion refines the grid there were it is already refined or worse, there were it was just unrefined. The grid gets many changes between levels and inaccurate solutions result on unnecessarily fine grids. Therefore, the criterion is not investigated in-depth; a few results for the Sod problem are given here.

Tests were run on the Sod problem, with a maximum level of 2. The level of the cells for three different settings of $d_s$ is given in figure 5.14. For high values of $d_s$, the grid for the expansion is ugly: there are far too many changes between level 1 and 2. Only for a low setting of $d_s$ the grids become smooth, but there are unnecessarily wide refinements.

Looking at a solution, figure 5.15, we see the effect of the wiggles in the grid: wiggles in the solution appear where the grid changes size, they are most obvious near the start of the expansion wave.

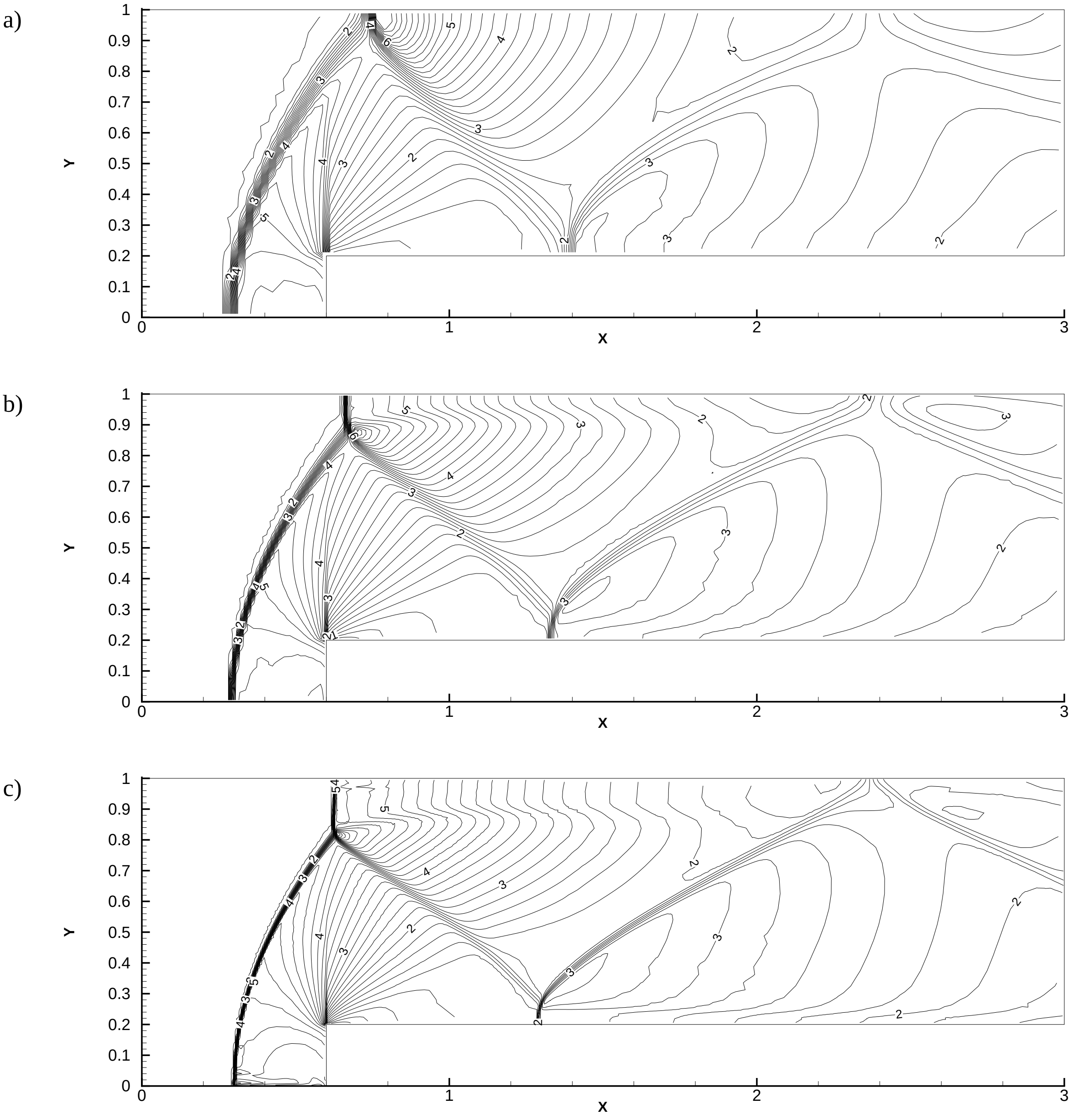


Figure 5.11: Forward-facing step, solution on adapted grids with $C_{\partial\rho}$ refinement criterion, $d_s = 4.0$, $d_m = 1.6$. Density $\rho$ at $t = 4.0$. Iso-lines, step 0.2 between the levels. $\Delta x = 1/20$ for the basic grid, maximum level $M = 1$ (a), 2 (b) and 3 (c).

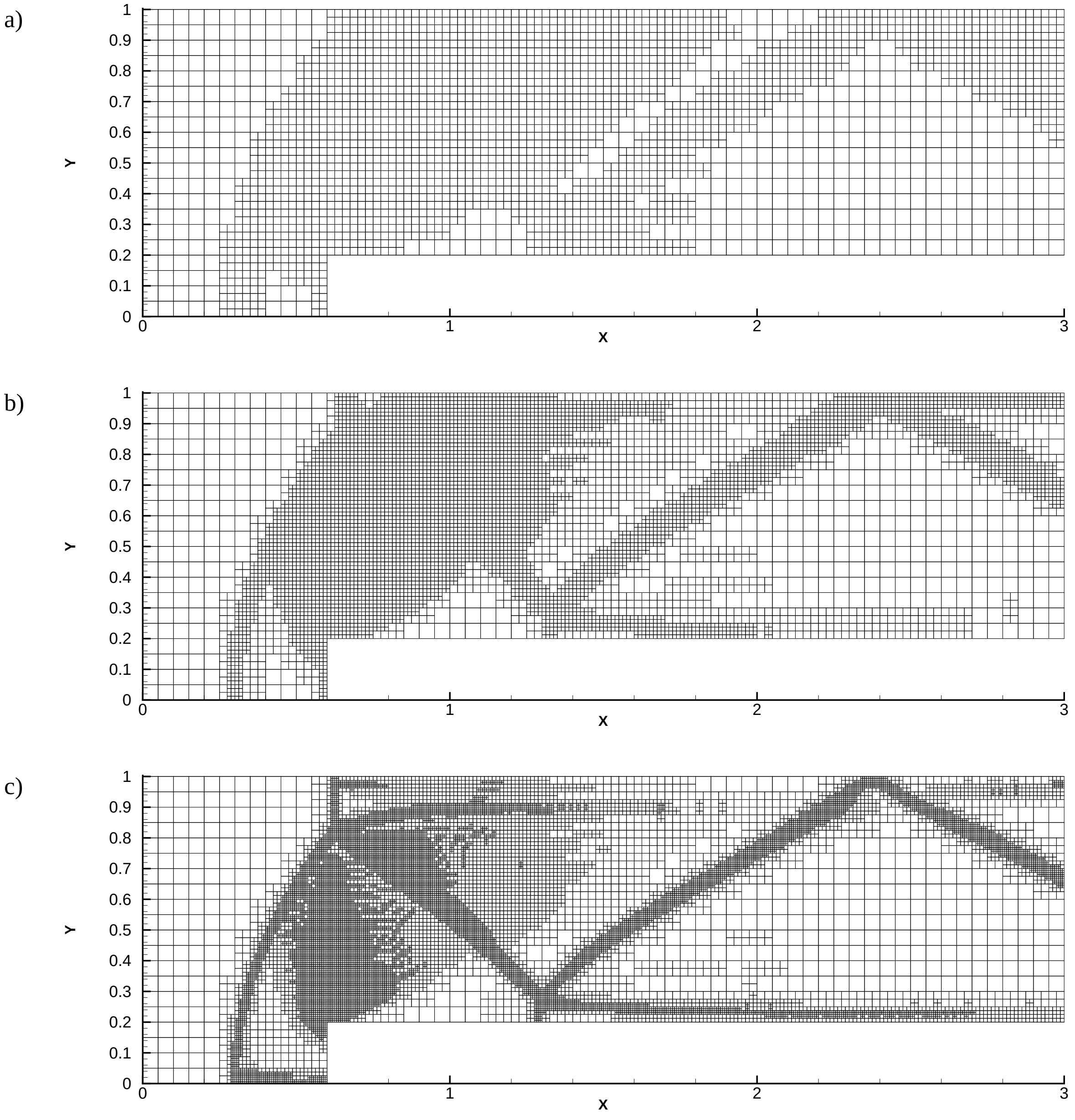


Figure 5.12: Forward-facing step, adapted grids. $C_{\partial\rho}$ refinement criterion, $d_s = 4.0$, $d_m = 1.6$. $\Delta x = 1/20$ for the basic grid, maximum level $M = 1$ (a), 2 (b) and 3 (c).

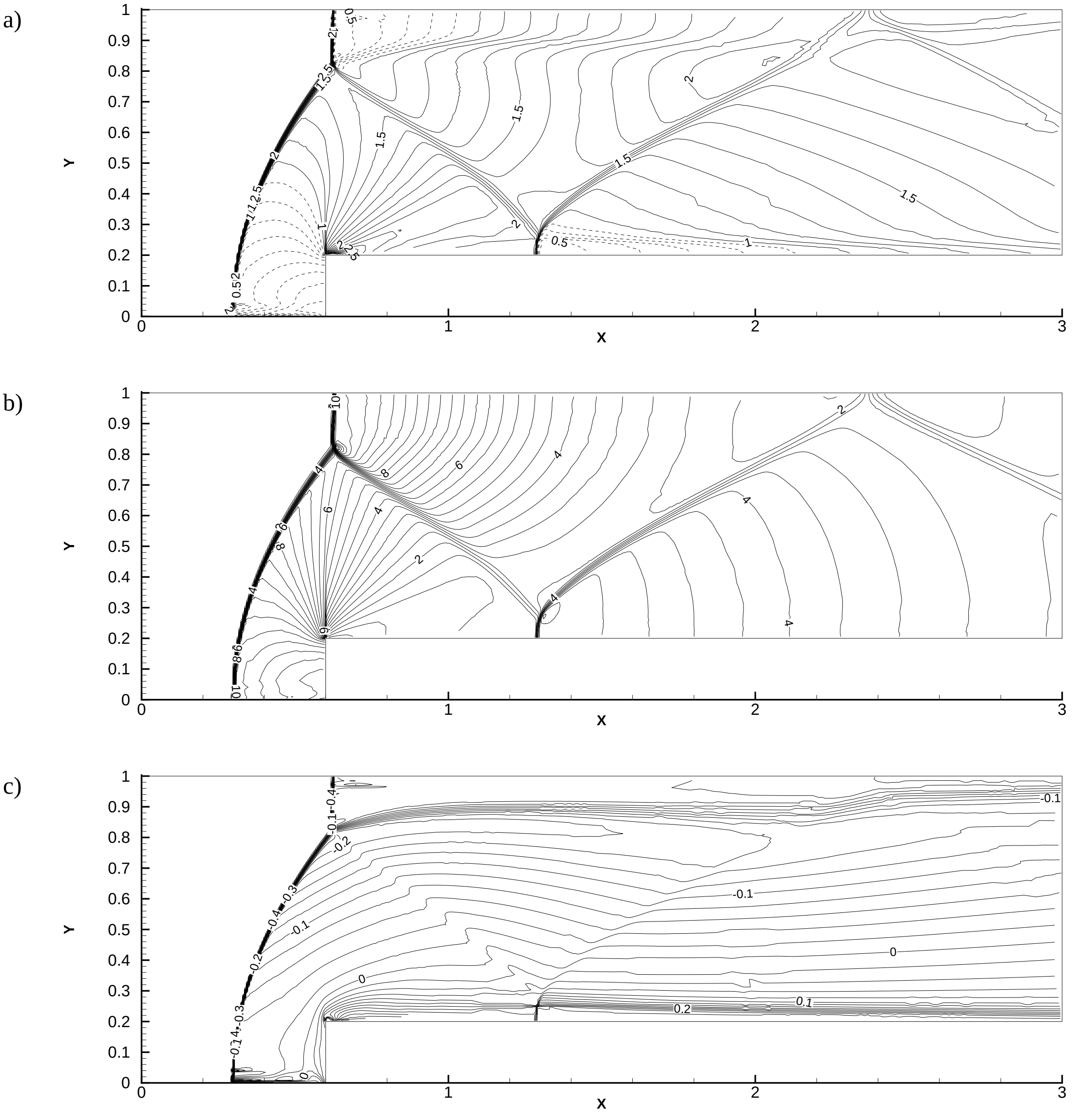


Figure 5.13: Forward-facing step, solution on adapted grids with $C_{\partial\rho}$ refinement criterion, $M = 3$. Iso-lines for Mach number $\frac{\sqrt{u^2+v^2}}{c}$, 0.1 between the levels (a), pressure $p$, 0.4 between the levels (b) and unscaled entropy $z$, 0.02 between the levels (c).

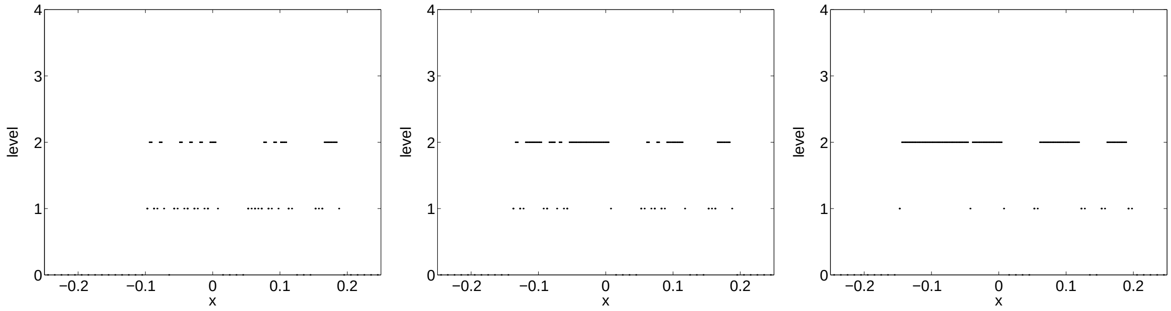


Figure 5.14: Sod problem, $C_{\partial^2\rho}$ refinement criterion. Cell level distributions with different settings of $d_s$, $d_s$ = 40, 120 and 200, $d_m = 0.4d_s$. $M = 2$.

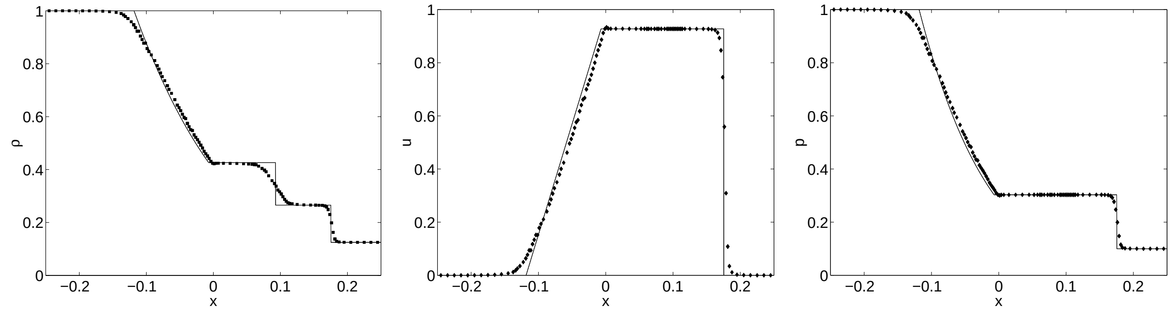


Figure 5.15: Sod problem, $C_{\partial^2\rho}$ refinement criterion. Solution for maximum level 2, $d_s = 120$, $d_m = 64$.

Lastly, figure 5.16 shows the values of the estimate of $\frac{\partial^2\rho}{\partial x^2}$ used in the refinement criterion. For a solution calculated with the $C_{\partial\rho}$ refinement criterion (a), $\frac{\partial^2\rho}{\partial x^2}$ indicates the global curvature of the solution, so this solution is smooth. $\frac{\partial^2\rho}{\partial x^2}$ for a solution calculated with $C_{\partial^2\rho}$ itself has more wiggles (b), the global curvature is unrecognizable due to the ‘noise’ in the solution.

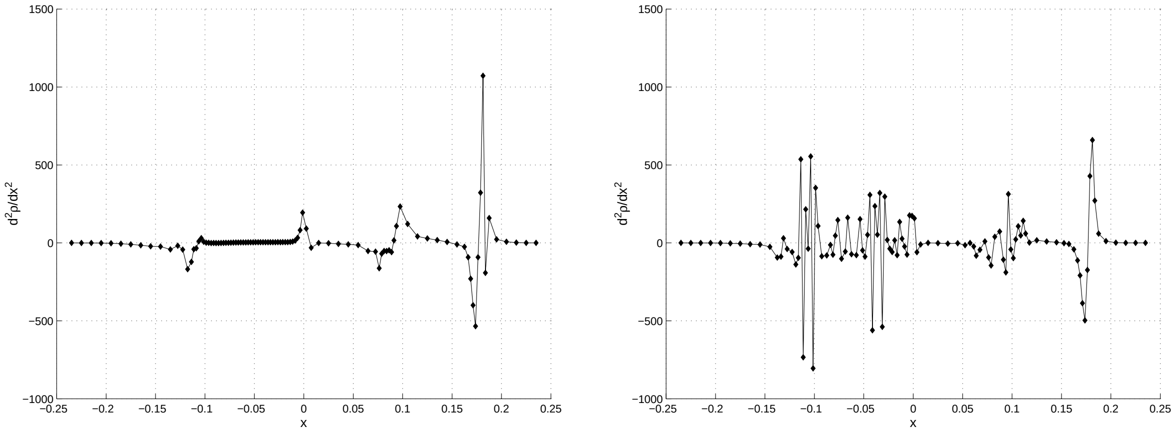
## 5.4. Error estimate refinement criterion

The $C_{EE}$ refinement criterion is defined in section 4.3. The problem is solved on two grids simultaneously, a coarse and a fine grid, and both grids are refined when the difference in the solution between the two grids becomes too high.

*Sod problem* To study the behaviour of $C_{EE}$, this criterion was calculated first for uniform grids, see figure 5.17. As can be expected, the error is not first-order: $C_{EE}$ does not become two times smaller when the grid is made two times finer. The tail end of the expansion comes close, but the front end converges much slower. For the contact discontinuity $C_{EE}$ does not reduce at all! It is interesting to note that the truncation error here keeps exactly the same value, only it occurs in a smaller zone. The error in the shock increases for smaller cell widths, but, again, the peak becomes narrower. Note that the curves resemble the second derivative of $\rho$ (in absolute value).

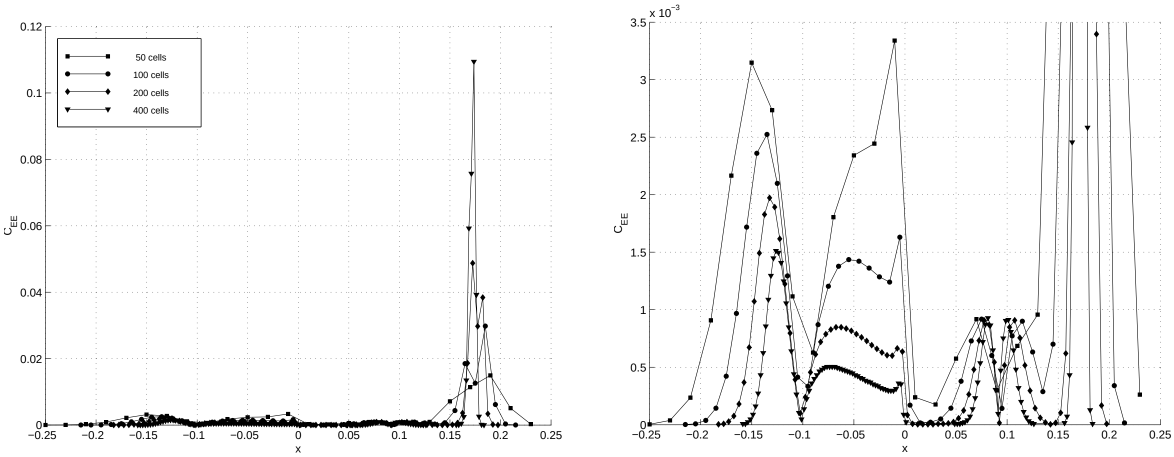
It is thus expected that the grid level distributions are wiggly: when a grid is refined, $C_{EE}$ reduces only a little, so the grid may be split again soon. It may even be split after an unrefinement, that causes some extra error. Therefore, extra buffers are used (section 4.4).

Looking at grid level distributions for selected cases (figure 5.18), we see the wiggles appear, especially in the expansion. However, there are fewer than for the $C_{\partial^2\rho}$ criterion. The grids are refined more around the



a) b)

Figure 5.16: Sod problem, the value of $\frac{\partial^2\rho}{\partial x^2}$. Solution a) is made with the $C_{\partial\rho}$ criterion ($d_s = 4.0$), solution b) with the $C_{\partial^2\rho}$ criterion ($d_s = 120$). $M = 2$.



a) b)

Figure 5.17: Sod problem, the value of $C_{EE}$ on uniform grids.

expansion, compared with the $C_{\partial\rho}$ results, a good feature of the $C_{EE}$ criterion as it obviously pays to refine the ‘head’ and ‘tail’ of the expansion.

The contact discontinuity, as expected from figure 5.17, is either fully refined or not at all. As $C_{EE}$ in the contact discontinuity does not change after refining, it is fully refined if $d_s$ is chosen lower than the error in the discontinuity on the lowest level. Note, however, that for high values of $d_s$ the edges of the contact discontinuity are sometimes on level 1.

The shock is always fully refined. The refined zone around the shock is broader than for the gradient $\rho$ criterion, partially because of the extra buffers.

Selected solutions are given in figure 5.19.

The errors (from equation (5.1)) for the $C_{EE}$ criterion are given in figure 5.20. Except for the error in $\rho$, which is influenced much by the refinement of the contact discontinuity, the errors converge approximately like the

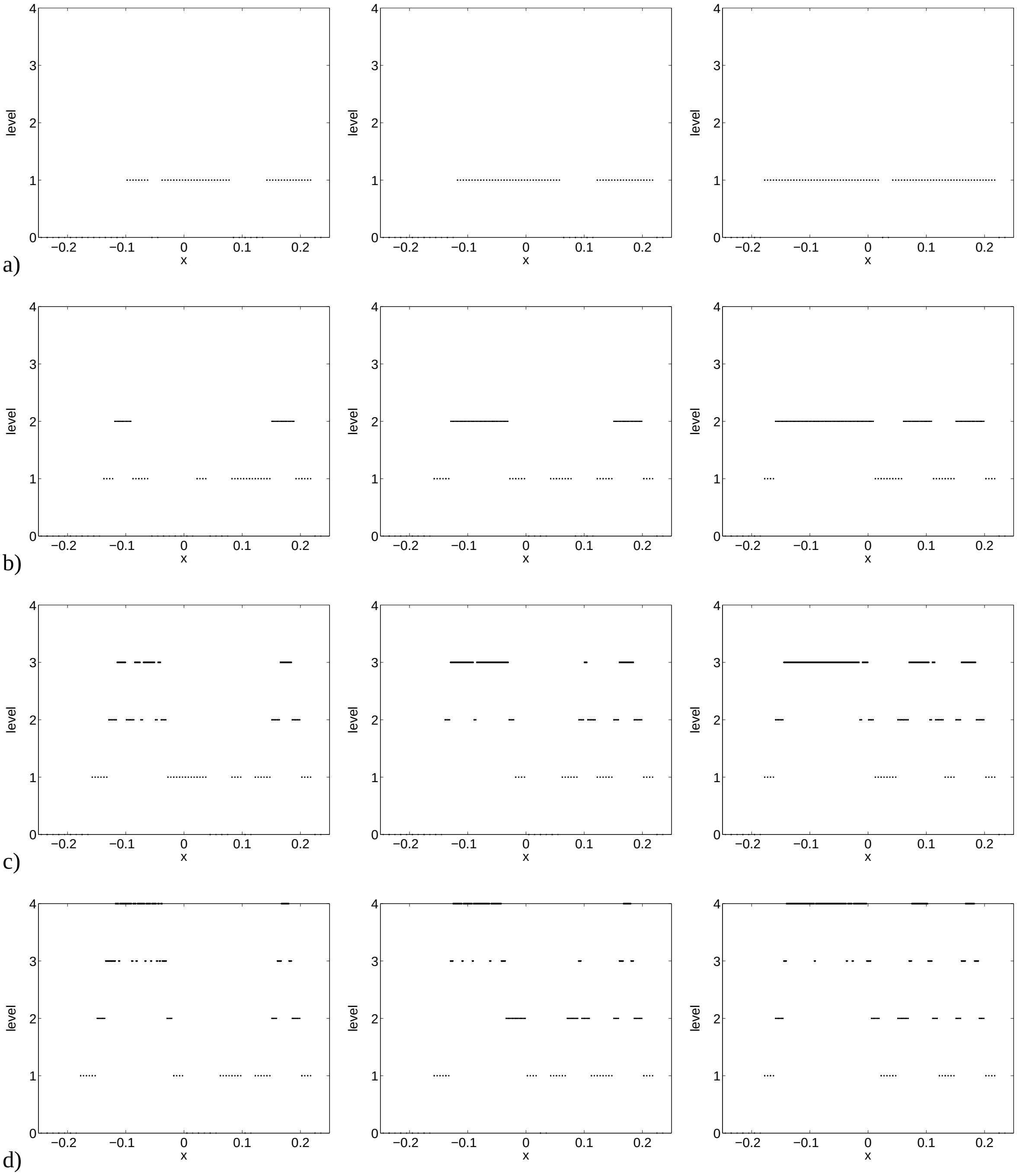


Figure 5.18: Sod problem, $C_{EE}$ refinement criterion. Cell level distributions with different settings of $d_s$: 0.008, 0.004 and 0.001. $d_m = 0.4d_s$, $d_{\Delta s} = 0.5d_s$. Maximum level 1 (a), 2 (b), 3 (c) and 4 (d).

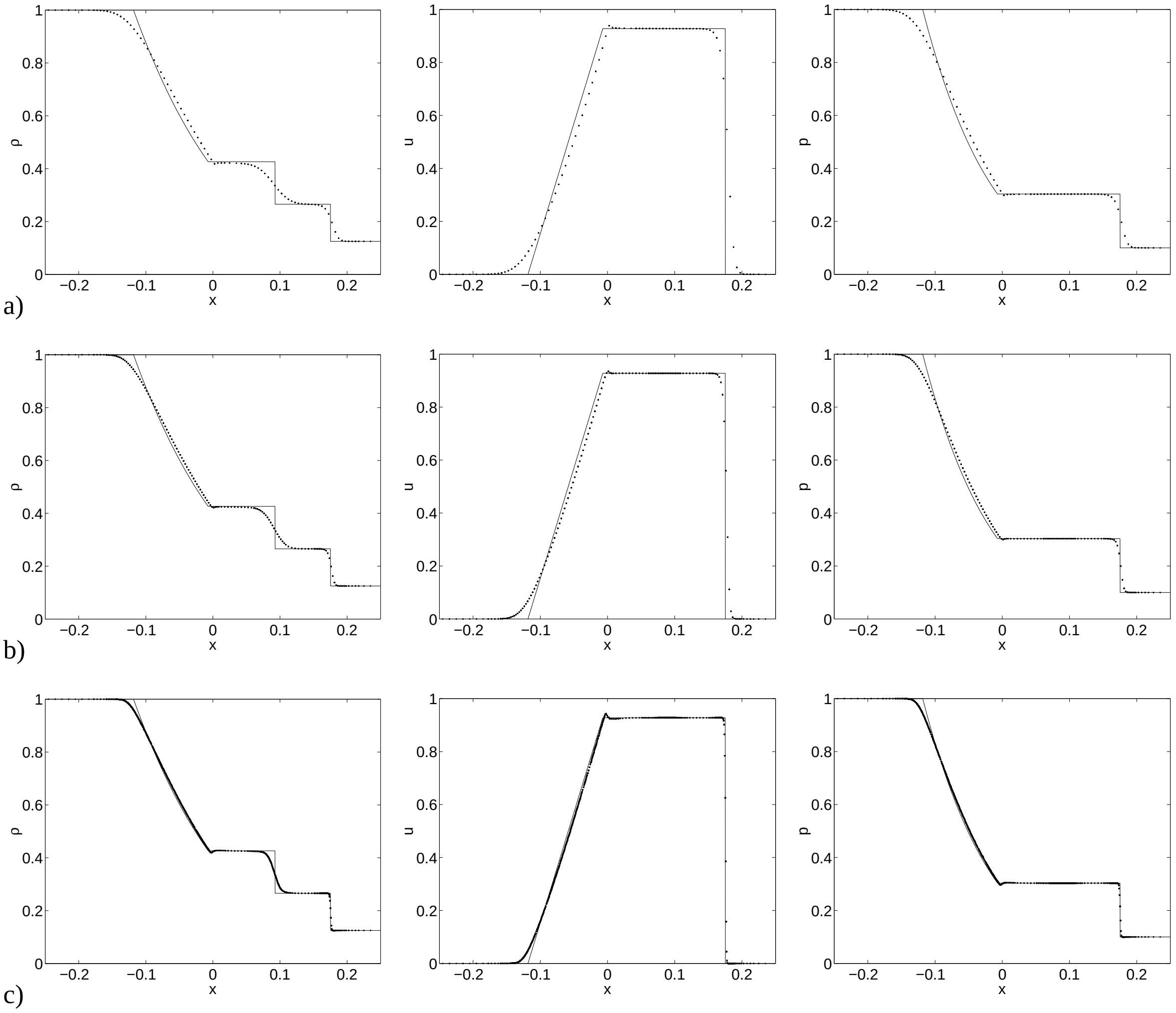


Figure 5.19: Sod problem, $C_{EE}$ refinement criterion. Solution for maximum levels 1 (a), 2 (b) and 4 (c). $d_s$ = 0.001, $d_m$ = 0.0004, $d_{s\Delta}$ = 0.0005. Domain $x = [-0.25, 0.25]$, $t = 0.1$, $\Delta x/\Delta t = 4.0$.

uniform case. The error on one level does not decrease when $d_s$ is decreased: it is assumed that the errors made on the highest grid level $M$ contribute most to the global error, so once the critical areas are refined on this level, further refining of other areas does not help. Note also that the errors do not converge to the uniform error.

The errors for the $C_{EE}$ criterion are higher than those for the $C_{\partial\rho}$ criterion with the same number of cells (compare figure 5.20 with figure 5.10). Most probably the expansions are refined unnecessarily, because of the wiggles in the grid distribution, so too many cells are used.

*Forward-facing step problem* Based on the experience with the Sod problem, a high value (0.16) was chosen for $d_s$, because the forward-facing step problem has strong shocks and because it appeared that solutions do not improve if $d_s$ is chosen low. The solutions in figure 5.21 look excellent. All features are resolved, also the oblique shock on the finest grid. One disadvantage is the waviness of the lines in the lower right corner. Also, the wiggles behind the Mach stem are back.

Looking at the grids (figure 5.22) we can understand this: the grids are almost uniformly refined behind the first bow shock, but the grid changes between levels at unexpected places (causing the waviness of the lines).

Also, all shocks are fully refined (note especially the beautiful transition from level 0 to level $M$ in front of the first bow shock), but the contact discontinuities are not: $d_s$ is higher than the error in the contact discontinuities.

The CPU times for the calculations are much higher than those for the $C_{\partial\rho}$ criterion, see table 5.5. The times are still lower than in the uniform case, but the gain is not high (12% – 25%). This is not strange, as many more cells are refined. And due to the extra calculation on the coarse grid, the computational costs for fluxes and advancing are about 9/8 of the costs for a $C_{\partial\rho}$ calculation.

| Max. level | CPU time | % of unif. |
|---|---:|---:|
| 1 | 90 s | 87% |
| 2 | 615 s | 74% |
| 3 | 5868 s | 86% |

Table 5.5: Forward-facing step, CPU times for the solution on refined grids, $C_{EE}$ refinement criterion (figure 5.21).

## 5.5. Conclusion

In this chapter, the first-order algorithm is tested with three refinement criteria. The simple $C_{\partial\rho}$ criterion is very robust and it gives solutions that have the same accuracy as first-order solutions on uniform grids. However, it requires quite many cells to accurately resolve expansion fans.

The $C_{\partial^2\rho}$ criterion is unsuitable for first-order calculations, as it is more sensitive to small errors in the solution than to the general shape of the solution itself. The grids produced are very wiggly and they are refined in useless places.

The $C_{EE}$ criterion looks promising, as it gives very good solutions. But it suffers from wiggly grids too, so it must be made more stable before it becomes useful. Also, it is probably more suitable for problems with smooth flow than for problems with strong discontinuities, as these cause strong irregularities in the solution.

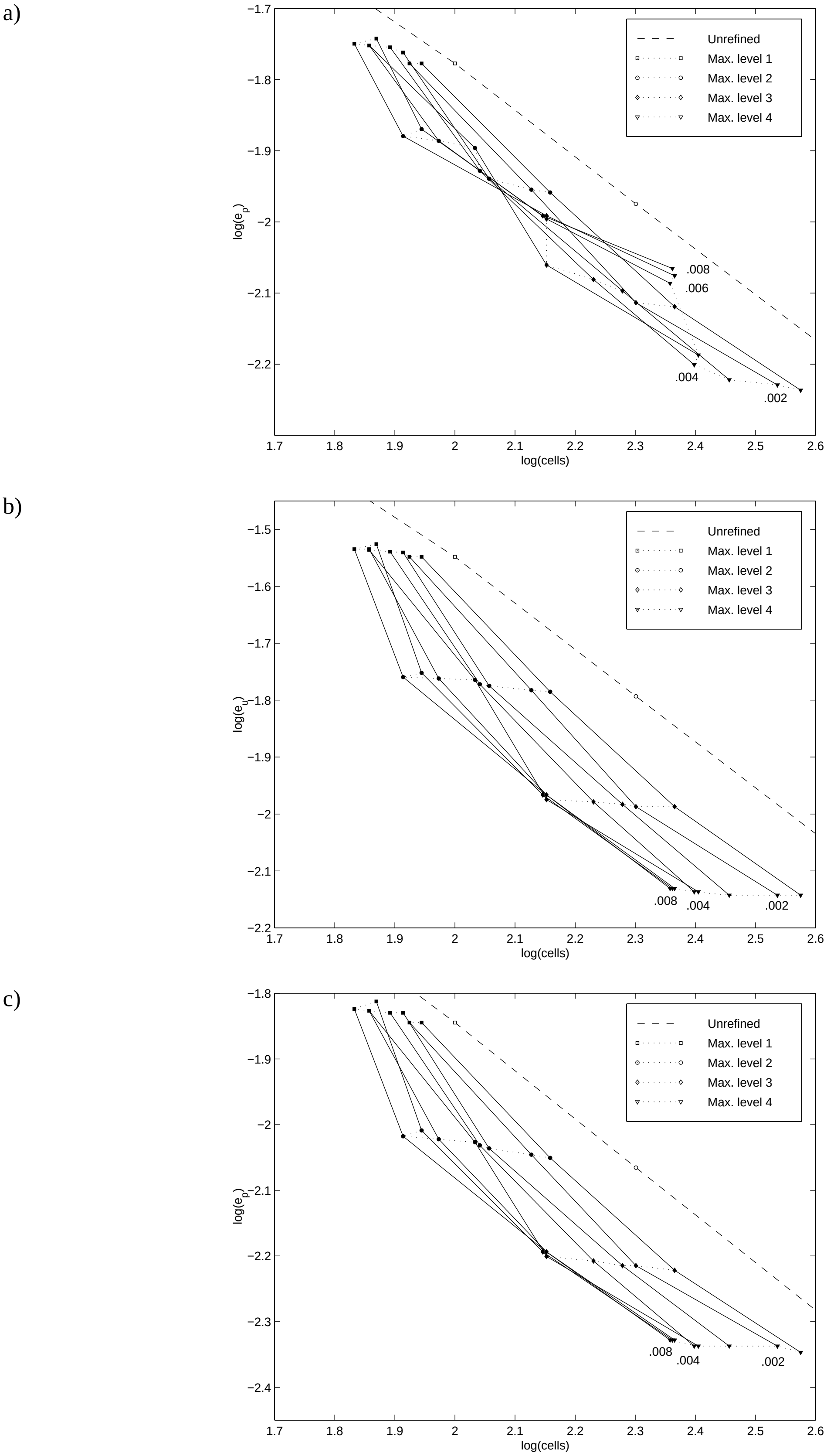


Figure 5.20: Sod problem, $C_{EE}$ refinement criterion. Errors in $\rho$ (a), $u$ (b) and $p$ (c). Dashed line gives the errors for uniform grids, solid lines the errors for $d_s = 0.001 \ldots 0.008$ (indicated on the plots). Note the double-log scales.

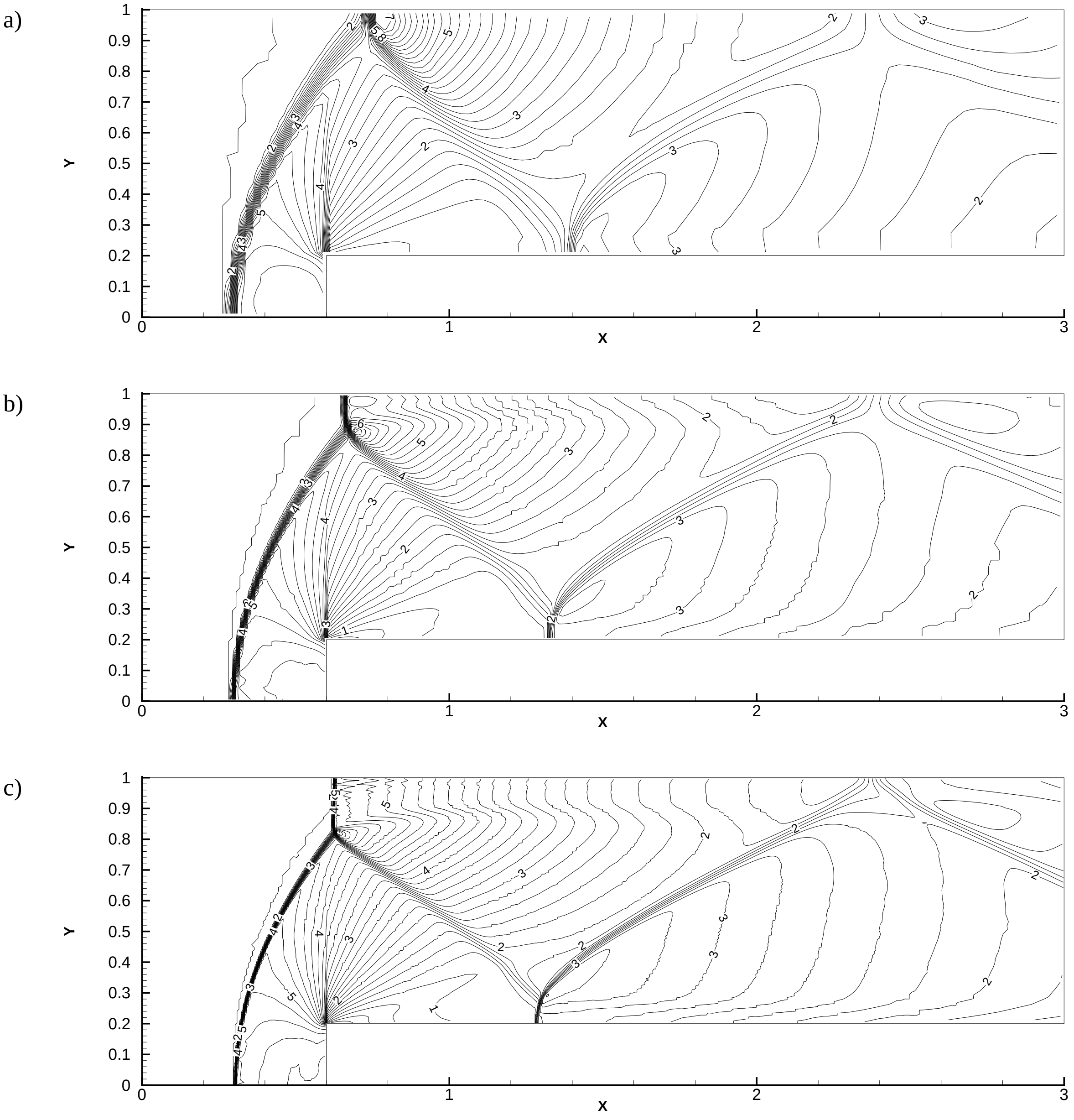


Figure 5.21: Forward-facing step, solution on adapted grids with $C_{EE}$ refinement criterion, $d_s = 0.16$, $d_m = 0.064$, $d_{s\Delta} = 0.08$. Density $\rho$ at $t = 4.0$. Iso-lines, step 0.2 between the levels. $\Delta x = 1/20$ for the basic grid, $1/10$ for the basic sister grid. Maximum level $M = 1$ (a), 2 (b) and 3 (c).

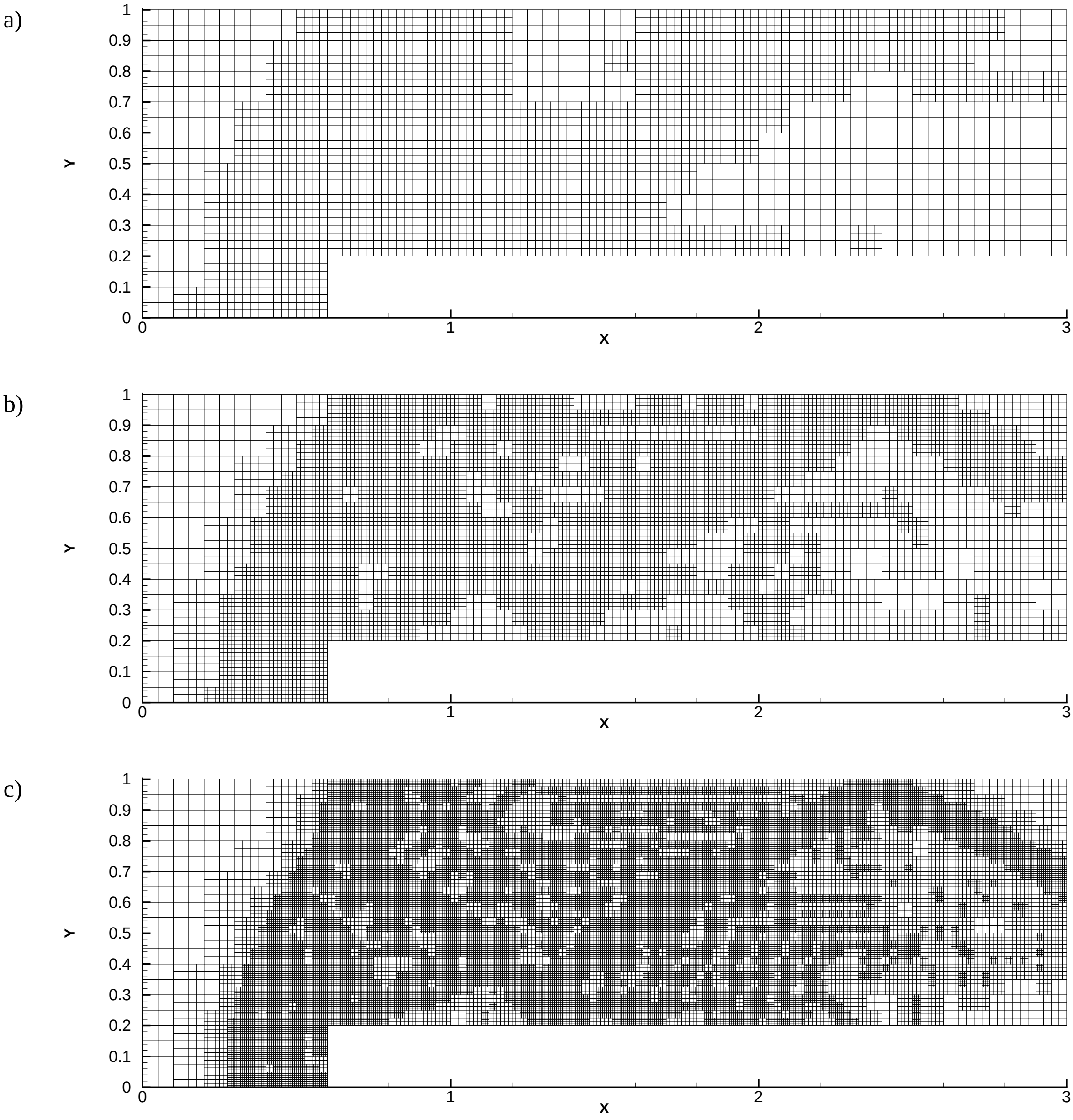


Figure 5.22: Forward-facing step, adapted grids. $C_{EE}$ refinement criterion, $d_s = 0.16$, $d_m = 0.064$, $d_{s\Delta} = 0.08$. $\Delta x = 1/20$ for the basic grid, $1/10$ for the basic sister grid. Maximum level $M = 1$ (a), 2 (b) and 3 (c).

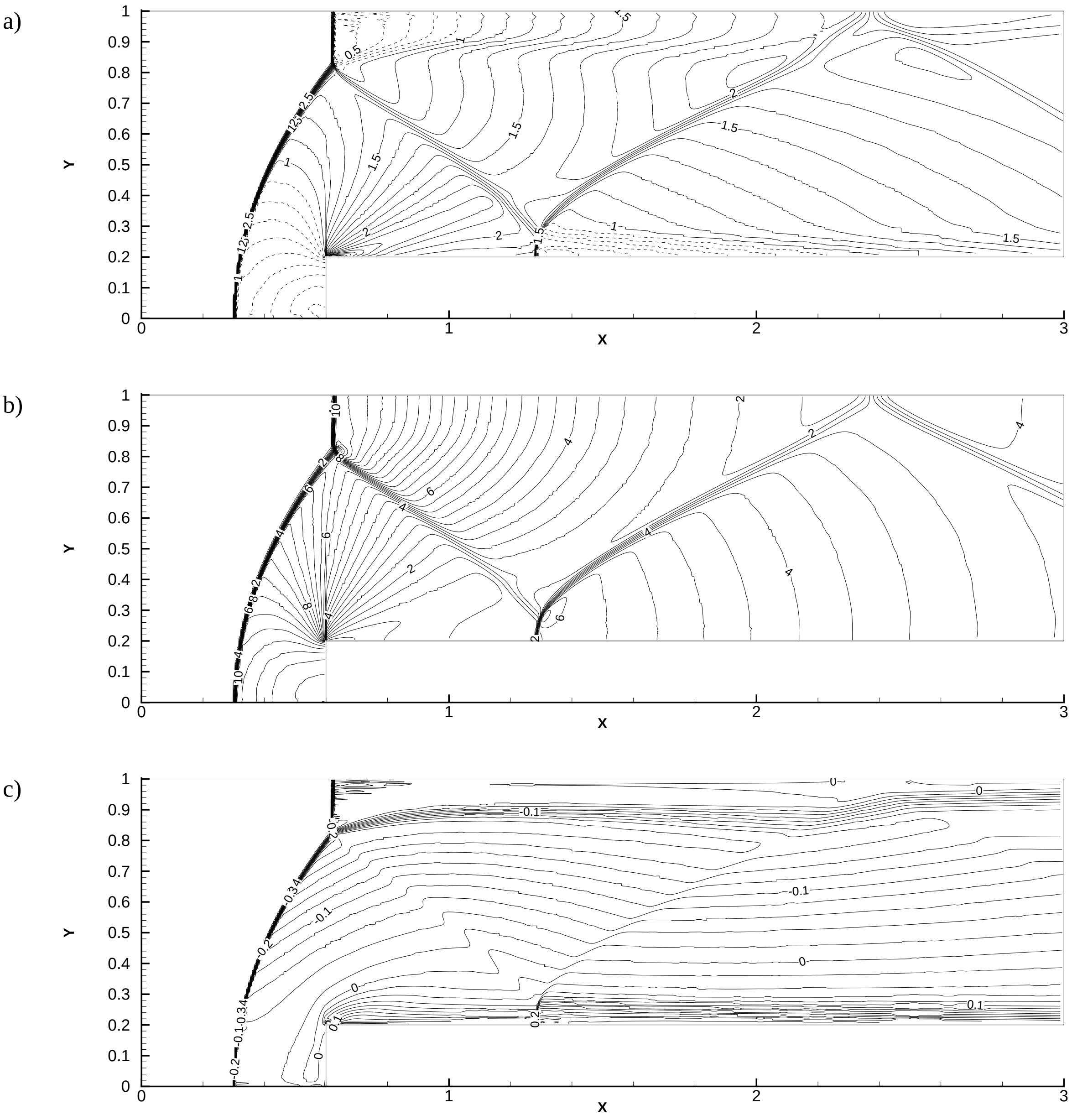


Figure 5.23: Forward-facing step, solution on adapted grids with $C_{EE}$ refinement criterion, $M = 3$. Iso-lines for Mach number $\frac{\sqrt{u^2+v^2}}{c}$, 0.1 between the levels (a), pressure $p$, 0.4 between the levels (b) and unscaled entropy $z$, 0.02 between the levels (c).

# Chapter 6
# Second-order accurate discretisation

Solving the Euler equations with a first-order accurate discretisation on uniform grids requires a lot of computation. The previous chapters showed that adaptive gridding can decrease this amount. Another well-known way to increase computational efficiency is to use a second-order accurate discretisation. In the following chapters, both approaches are combined.

A basic second-order accurate discretisation of the Euler equations is described here, comparable with the first-order discretisation from chapter 2. Section 1 gives a second-order discretisation of the fluxes, section 2 describes a second-order scheme for the time derivatives. The stability of the obtained scheme is studied in section 3 and finally, the treatment of boundary conditions is given in section 4. All equations are given here for uniform grids, so cartesian cell numbering is used for clarity.

## 6.1. Space discretisation

It follows from section 2.2 that a second-order accurate discretisation of the fluxes is achieved when the states at the cell faces are determined with second-order accuracy. Therefore, setting these states from just one cell is not accurate enough, some form of interpolation or extrapolation between two or more cells is needed.

Two basic possibilities exist for a second-order accurate calculation of the states at the cell faces: central and upwind discretisation. Both use a linear approximation of $\boldsymbol{q}$, but the central scheme interpolates between two cells (figure 6.1a),

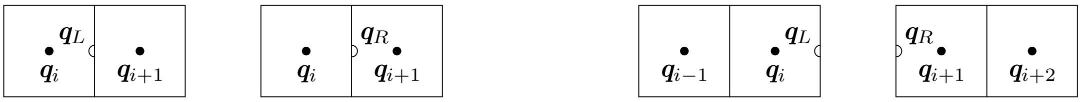
$$\begin{aligned} \boldsymbol{q}_L &= \tfrac{1}{2}\boldsymbol{q}_{i+1} + \tfrac{1}{2}\boldsymbol{q}_i, \\ \boldsymbol{q}_R &= \tfrac{1}{2}\boldsymbol{q}_i + \tfrac{1}{2}\boldsymbol{q}_{i+1}, \end{aligned} \tag{6.1}$$

while the upwind scheme extrapolates (figure 6.1b),

$$\begin{aligned} \boldsymbol{q}_L &= \tfrac{3}{2}\boldsymbol{q}_i - \tfrac{1}{2}\boldsymbol{q}_{i-1}, \\ \boldsymbol{q}_R &= \tfrac{3}{2}\boldsymbol{q}_{i+1} - \tfrac{1}{2}\boldsymbol{q}_{i+2}. \end{aligned} \tag{6.2}$$

Both are second-order accurate, but when the central scheme is used, the solution becomes usually unstable (note that $\boldsymbol{q}_L$ and $\boldsymbol{q}_R$ are equal for this scheme). The fully one-sided upwind scheme usually gives stable solutions, but they are not monotone: solutions tend to have spurious oscillations ('wiggles') in the neighbourhood of shocks and contact discontinuities. This means that the state in one cell is below the exact value, the state in the next cell is above the exact value, etc.

It can be shown [6] that any linear discretisation for the states at the cell faces that gives monotone solutions, is at most first-order accurate. Therefore, the monotonicity problem is solved by using a nonlinear, limited



a) Central scheme

b) Upwind scheme

Figure 6.1: Cells needed to determine cell face state (the halve circle).

scheme. In this scheme, the coefficients in the interpolation formula depend on the solution. Limiters are described in detail in [10] and [16], an overview is given here.

The limited scheme uses three cell states to determine a cell face state (figure 6.2). The equations are, per component $q_p$ of $\boldsymbol{q}$,

$$\begin{aligned} q_{L,p} &= q_{i,p} + \frac{1}{2}\phi\left(r_L\right)\left(q_{i,p} - q_{i-1,p}\right), \\ q_{R,p} &= q_{i+1,p} + \frac{1}{2}\phi\left(r_R\right)\left(q_{i+1,p} - q_{i+2,p}\right), \end{aligned} \tag{6.3}$$

with

$$\begin{aligned} r_L &= \frac{q_{i+1,p} - q_{i,p}}{q_{i,p} - q_{i-1,p}}, \\ r_R &= \frac{q_{i,p} - q_{i+1,p}}{q_{i+1,p} - q_{i+2,p}}. \end{aligned} \tag{6.4}$$

The nonlinear limiter function $\phi(r)$ is used to change the discretisation: if $\phi(r) = 1$, then the scheme is equal to the upwind scheme (6.2), if $\phi(r) = r$, then the scheme is the central scheme (6.1). For $\phi(r) = 0$, the scheme is the first-order accurate scheme (3.4). For other values of $\phi$, intermediate schemes appear.

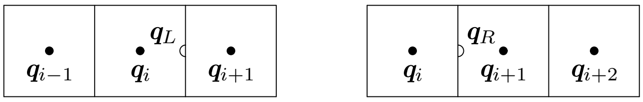


Figure 6.2: Cells needed to determine cell face state. Limited scheme.

The parameter $r$ is an indication of the curvature of the solution (figure 6.3). Different values of $r$ have the following significance:

- $r \approx 1$. The most common situation, $r \approx 1$ indicates a smooth solution.
- $r \approx 0$. A discontinuity on the upwind side.
- $r \to \infty$. Almost the same as $r \approx 0$, this also indicates a discontinuity, but now across the cell face.
- $r < 0$. A maximum or minimum in the solution, usually a numerical error, e.g. a wiggle (a normal Euler solution has only a few physical maxima or minima).

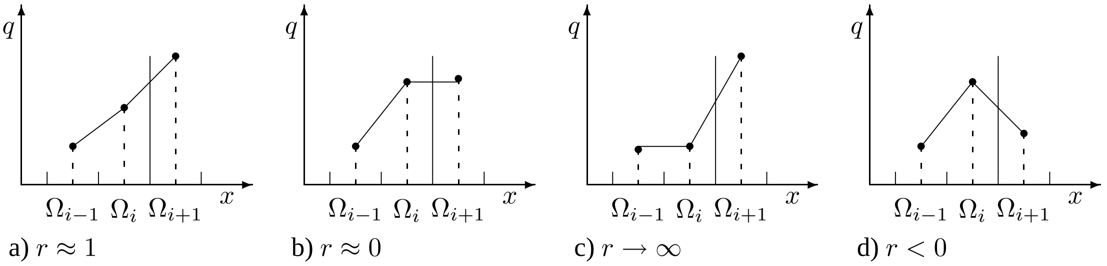

$q$
$x$
$\Omega_{i-1}$ $\Omega_i$ $\Omega_{i+1}$
a) $r \approx 1$
$q$
$x$
$\Omega_{i-1}$ $\Omega_i$ $\Omega_{i+1}$
b) $r \approx 0$
$q$
$x$
$\Omega_{i-1}$ $\Omega_i$ $\Omega_{i+1}$
c) $r \to \infty$
$q$
$x$
$\Omega_{i-1}$ $\Omega_i$ $\Omega_{i+1}$
d) $r < 0$

Figure 6.3: Curvature function $r$, different values. $r_L$ is shown here, for $r_R$ the situation is reversed. The vertical bar indicates the cell face.

A function $\phi(r)$ must be chosen. The limiter function can be chosen freely by the code developer, as it has no physical meaning. However, there are some limitations. Sweby [17] has proposed a monotonicity domain, an area in the $r$ – $\phi$ plane in which a limiter must lie in order to have monotone solutions (no wiggles). Later this domain was extended, e.g. by Spekreijse [16]. His domain has a shape as shown in figure 6.4.

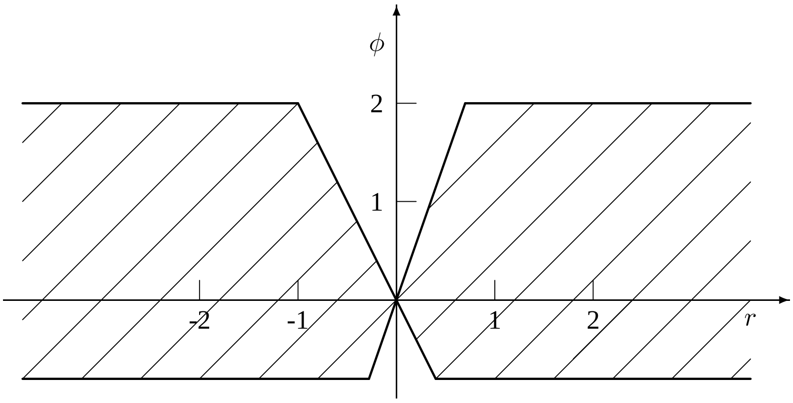


Figure 6.4: Monotonicity domain in the $r$ – $\phi$ plane.

Considering the shape of the domain, we see that $\phi$ must become very small for $r \approx 0$ and a constant value for $r \to \infty$. Such a limiter chooses the discretisation that sets the cell face state mostly from the central cell and the neighbour cell with the smallest difference in states (the central scheme for $r \approx 0$, the upwind scheme for $r \to \infty$). So the cell face states near discontinuities stay close to the state in their own cell. No places occur where, say, two neighbour cells have a positive difference while their cell face states have a negative difference, as this is a sure cause of wiggles.

In order to yield a second-order accurate discretisation, a limiter function $\phi(r)$ must pass through the point $(1, 1)$. Apart from the demand that $|\phi^{'}|$ and $|\phi^{''}|$ never go to $\infty$, this is the only condition for second-order accuracy. Note that the limiter function $\phi = 0$, the first-order scheme, does not fulfil this condition (as expected).

In the literature, many different limiters are proposed (see e.g. [16], section 2.5). In this report, two limiters are used:

*Minmod limiter* Limiter functions near $(1, 1)$ are often picked from the set of lines with slope $[0, 1]$ through $(1, 1)$. The Minmod limiter has the lowest possible values for $\phi$ that still lie in this domain. It is given by

$$\phi_{MM}(r) = \begin{cases} 0 & r < 0, \\ r & 0 \leq r < 1, \\ 1 & r \geq 1. \end{cases} \tag{6.5}$$

(This can be read as: use the first-order scheme for $r < 0$, the central scheme for $0 \leq r < 1$ and the upwind scheme for $r \geq 1$.) It can be proved through Taylor expansion that a low value of $\phi$ corresponds to high artificial viscosity, so this limiter is suitable for problems with very strong shocks, which are likely to cause wiggles. A graph is given in figure 6.5.

*Modified Van Albada limiter* The original Van Albada limiter is a smooth function that has a limit value 1 for $r \to \pm\infty$. It is modified here to have a value 0 for $r < 0$, because the first-order scheme with its high artificial

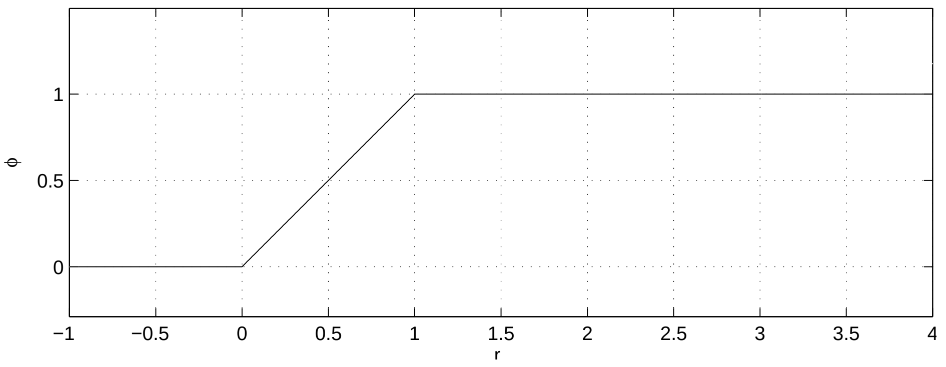


Figure 6.5: Minmod limiter.

viscosity is the most effective scheme to remove wiggles. The limiter is

$$\phi_{MVA}(r) = \begin{cases} 0 & r < 0, \\ \frac{r^2+r}{r^2+1} & r \geq 0, \end{cases} \tag{6.6}$$

(see also figure 6.6). It is interesting to note that $\lim_{r\downarrow 0}\phi_{MVA} = r$ and $\lim_{r\to\infty}\phi_{MVA} = 1$, the same values as for the Minmod limiter.

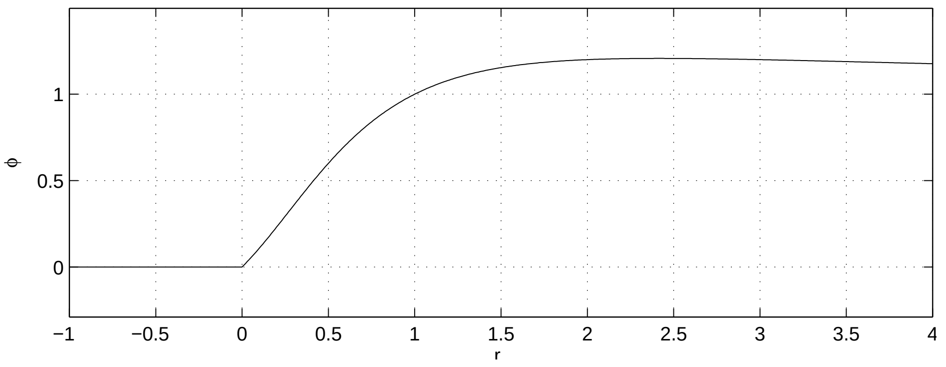


Figure 6.6: Modified Van Albada limiter.

One last important remark must be made about the limited scheme: the limiter works *per component* of $\boldsymbol{q}$, to make sure that all components of $\boldsymbol{q}$ are sensible (for example, their difference must have the same sign as the difference between the states in the left and right cell). The limited states are used as input for the Osher flux function, so it is essential that the input state vector $(u, v, c, z)$ for the Osher scheme is limited. Therefore, the conservative states in the cells are converted to the state vector $(u, v, c, z)$ and these states are limited to give $\boldsymbol{q}_L$ and $\boldsymbol{q}_R$.

## 6.2. Time discretisation

Two different approaches are available to discretise the time derivatives in the Euler equations with higher-order accuracy. A widely used family of methods are the multi-stage methods, like the various Runge-Kutta methods (see e.g. [10]). These methods do not calculate a time step in one stage, but several intermediate stages are calculated using, for the flux calculation, a combination of $\boldsymbol{q}^k$ and the previous intermediate stages. The final solution $\boldsymbol{q}^{k+1}$ is taken as a combination of these stages.

These methods are unsuitable for the adaptive-gridding algorithm, because they cannot easily handle different time steps for neighbouring cells in a consistent way. Consider, for instance, a time step for a cell with a larger neighbour. To calculate the different intermediate stages, the fluxes in the cell faces are calculated with states based on previous stages. Intermediate stages for the new state in the bigger cell cannot be calculated until the smaller cells are advanced twice, but they are already needed for the *first* step in the small cells. It is not easy to solve this problem.

Another series of methods are the so called multi-step methods: $\boldsymbol{q}^{k+1}$ is not calculated using the current state $\boldsymbol{q}^{k}$ only, but the *previous* states $\boldsymbol{q}^{k-1}$, $\boldsymbol{q}^{k-2}$ etc. are used too. One of the best known methods is the Leapfrog scheme, a finite-difference scheme that uses central discretisations in both time and space (figure 6.7a). For the Euler equations in one space dimension, the scheme is

$$\frac{\boldsymbol{q}_i^{k+1}-\boldsymbol{q}_i^{k-1}}{2\Delta t}+\frac{\boldsymbol{f}_{i+1}^{k}-\boldsymbol{f}_{i-1}^{k}}{2\Delta x}=0. \tag{6.7}$$

The central discretisation of the time derivative is second-order accurate,

$$\frac{\partial \boldsymbol{q}}{\partial t}=\frac{\boldsymbol{q}^{k+1}-\boldsymbol{q}^{k-1}}{2\Delta t}+\mathcal{O}\left(\Delta t^2\right). \tag{6.8}$$

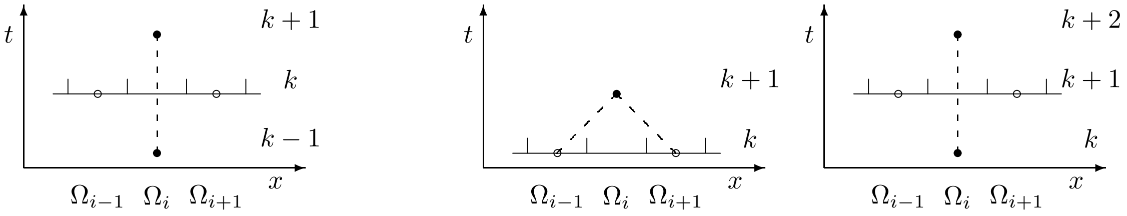
A disadvantage of this scheme is its marginal stability for linear equations, making it usually unstable for nonlinear equations. This disadvantage can be removed by combining the Leapfrog scheme (6.7) with the Lax scheme that is first-order accurate in time: the resulting scheme is called the two-step Lax-Wendroff or Richtmyer scheme (figure 6.7b),

$$\begin{aligned}\frac{\boldsymbol{q}_i^{k+1}-\frac{1}{2}\left(\boldsymbol{q}_{i-1}^{k}+\boldsymbol{q}_{i+1}^{k}\right)}{\Delta t}+\frac{\boldsymbol{f}_{i+1}^{k}-\boldsymbol{f}_{i-1}^{k}}{2\Delta x}&=0,\\ \frac{\boldsymbol{q}_i^{k+2}-\boldsymbol{q}_i^{k}}{2\Delta t}+\frac{\boldsymbol{f}_{i+1}^{k+1}-\boldsymbol{f}_{i-1}^{k+1}}{2\Delta x}&=0.\end{aligned} \tag{6.9}$$

So the two schemes are alternated. During the first step, the states are advanced as for the first-order algorithm, giving the scheme stability. For the second step, the fluxes are calculated with these advanced states, but the time derivative with the old states. This step makes the entire scheme $\mathcal{O}\left(\Delta t^2\right)$ accurate.

• state used for time derivative

○ state used for fluxes


t
k + 1
k
k − 1
x
$\Omega_{i-1}$ $\Omega_i$ $\Omega_{i+1}$
t
k + 1
k
x
$\Omega_{i-1}$ $\Omega_i$ $\Omega_{i+1}$
t
k + 2
k + 1
k
x
$\Omega_{i-1}$ $\Omega_i$ $\Omega_{i+1}$

a) Leapfrog

b) Richtmyer

Figure 6.7: Stencils for second-order accurate finite-difference schemes. Leapfrog scheme (a) and the two steps of the Richtmyer scheme, Lax (left graph) and Leapfrog (right graph) (b).

Both the Leapfrog and the Richtmyer scheme are finite-difference schemes that use central differences for the flux evaluation. But we want to use a finite-volume scheme with the limited upwind flux discretisation of the previous section, to make sure the scheme is conservative and to keep the good shock capturing of the Osher flux function. Therefore, the finite-volume equation (2.7) is combined with the time discretisations of the Richtmyer scheme

$$\begin{aligned} \left.\frac{\partial \boldsymbol{q}}{\partial t}\right|_{k} &= \frac{\boldsymbol{q}^{k+1}-\boldsymbol{q}^{k}}{\Delta t} + \mathcal{O}\left(\Delta t\right), \\ \left.\frac{\partial \boldsymbol{q}}{\partial t}\right|_{k+1} &= \frac{\boldsymbol{q}^{k+2}-\boldsymbol{q}^{k}}{2\Delta t} + \mathcal{O}\left(\Delta t^2\right). \end{aligned} \tag{6.10}$$

The result is a new scheme for two spatial dimensions, (see also figure 6.8),

$$\begin{aligned} \boldsymbol{q}_i^{k+1} &= \boldsymbol{q}_i^{k} - \left(\boldsymbol{F}_{O\,i,r}^{k} - \boldsymbol{F}_{O\,i,l}^{k}\right)\frac{\Delta t}{\Delta x} - \left(\boldsymbol{G}_{O\,i,a}^{k} - \boldsymbol{G}_{O\,i,b}^{k}\right)\frac{\Delta t}{\Delta y}, \\ \boldsymbol{q}_i^{k+2} &= \boldsymbol{q}_i^{k} - 2\left(\boldsymbol{F}_{O\,i,r}^{k+1} - \boldsymbol{F}_{O\,i,l}^{k+1}\right)\frac{\Delta t}{\Delta x} - 2\left(\boldsymbol{G}_{O\,i,a}^{k+1} - \boldsymbol{G}_{O\,i,b}^{k+1}\right)\frac{\Delta t}{\Delta y}. \end{aligned} \tag{6.11}$$

If equation (6.3) is used for the fluxes, then the scheme is second-order accurate in both time and space.

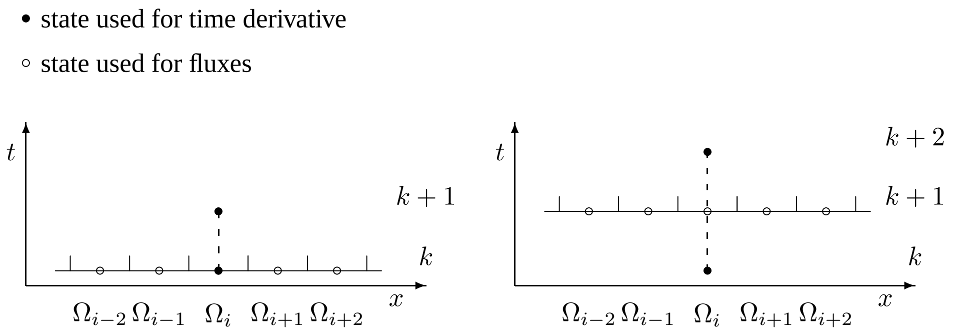

• state used for time derivative
◦ state used for fluxes
$t$
$k+1$
$k$
$x$
$\Omega_{i-2}$ $\Omega_{i-1}$ $\Omega_i$ $\Omega_{i+1}$ $\Omega_{i+2}$
$t$
$k+2$
$k+1$
$k$
$x$
$\Omega_{i-2}$ $\Omega_{i-1}$ $\Omega_i$ $\Omega_{i+1}$ $\Omega_{i+2}$

Figure 6.8: Stencil for the new, second-order accurate upwind scheme (in 1D). Two steps, the first has a forward-Euler time derivative and the second a Leapfrog time derivative.

This scheme is a natural choice for the adaptive-gridding algorithm of section 3.2, since this algorithm already has a two-step structure. Therefore, it is easy to implement the scheme. But its biggest advantage is, that it is self-starting: no separate discretisation is needed to calculate the *first* time step. (This is not the case for the Leapfrog scheme, since this scheme always needs a previous step. Of course, there is no ‘previous step’ for the first time step.) For the adaptive-gridding algorithm, a ‘start-up’ situation occurs not only at the first time step, but every time a cell is split into four small cells. With the scheme (6.11), these new cells just start with step 1 and therefore it is not necessary to interpolate previous states from the old big cell.

It is possible that a combination of the Leapfrog scheme with the limited upwind flux discretisation is also stable. However, this scheme is not self-starting, therefore it is not investigated here.

## 6.3. Constraints on the time step

In section 3.1, we saw that stability puts a constraint on the maximum value of $\Delta t/\Delta x$, the CFL criterion (3.1). For the second-order scheme, there still is a stability constraint on $\Delta t$, but monotonicity also puts a maximum on the time step. It can be shown [10] that this constraint has the same form as the first-order CFL condition, yet it is more severe. For the forward-Euler scheme applied to first-order upwind linear advection equations, the monotonicity requirement reads

$$|\lambda|_{max}\frac{\Delta t}{\Delta x} \leq \frac{1}{2}. \tag{6.12}$$

For nonlinear equations this constraint cannot be determined analytically, but it is usually the same.

To determine the stability constraint for the present scheme, let us apply a Von Neumann stability analysis to the discretisation (6.11). For simplicity, a linear model equation is analysed:

$$\frac{\partial q}{\partial t} + a\frac{\partial q}{\partial x} = 0. \tag{6.13}$$

Also, the space-derivative is modelled with a pure one-sided upwind scheme. With cartesian indices:

$$\begin{aligned} q_{i-\frac{1}{2}} &= \tfrac{3}{2}q_{i-1} - \tfrac{1}{2}q_{i-2}, \\ q_{i+\frac{1}{2}} &= \tfrac{3}{2}q_{i} - \tfrac{1}{2}q_{i-1}. \end{aligned} \tag{6.14}$$

Therefore, the finite-volume discretisation of (6.13) becomes

$$\begin{aligned} q_i^{k+1} &= q_i^k - a\frac{\Delta t}{\Delta x}\left(\tfrac{3}{2}q_i^k - 2q_{i-1}^k + \tfrac{1}{2}q_{i-2}^k\right), \\ q_i^{k+2} &= q_i^k - 2a\frac{\Delta t}{\Delta x}\left(\tfrac{3}{2}q_i^{k+1} - 2q_{i-1}^{k+1} + \tfrac{1}{2}q_{i-2}^{k+1}\right). \end{aligned}$$

Substituting the first part in the second part and writing $A$ for $a\frac{\Delta t}{\Delta x}$, this becomes

$$\begin{aligned} q_i^{k+2} = q_i^k &- A\left(3q_i^k - 4q_{i-1}^k + q_{i-2}^k\right) \\ &+ A^2\left(4\tfrac{1}{2}q_i^k - 12q_{i-1}^k + 11q_{i-2}^k - 4q_{i-3}^k + \tfrac{1}{2}q_{i-4}^k\right). \end{aligned} \tag{6.15}$$

Now it is assumed that the solution at one time level can be Fourier decomposed into waves with different wavelengths. The development in time of these waves is studied and if waves of all frequencies are damped in time, then the scheme is stable. To do this, let us introduce

$$q_i^k = Q\zeta^k e^{I\omega i\Delta x}. \tag{6.16}$$

$Q$ is a constant, $\zeta$ is a frequency-dependent amplification factor, $\omega$ is a measure for the wavelength, $\omega \in [-\pi, \pi]$. $I$ is used for the imaginary unit, $I^2 = -1$, because $i$ is already in use to number the cells. After substitution of (6.16) into (6.15) and division by $Q\zeta^k e^{I\omega i\Delta x}$, an expression is found for $\zeta$,

$$\begin{aligned} \zeta^2 = 1 &- A\left(3 - 4e^{-I\omega\Delta x} + e^{-2I\omega\Delta x}\right) \\ &+ A^2\left(4\tfrac{1}{2} - 12e^{-I\omega\Delta x} + 11e^{-2I\omega\Delta x} - 4e^{-3I\omega\Delta x} + \tfrac{1}{2}e^{-4I\omega\Delta x}\right). \end{aligned} \tag{6.17}$$

The norm of this expression is taken and, after much rewriting, it is found that

$$\begin{aligned} \left|\zeta^2\right|^2 = 1 &+ (1-\cos(\omega\Delta x))^2\left(-4A + 16A^2\right) + (1-\cos(\omega\Delta x))^3\left(-16A^3 + 48A^4\right) \\ &+ (1-\cos(\omega\Delta x))^4\left(8A^2 - 24A^3 + 36A^4\right). \end{aligned} \tag{6.18}$$

For a stable scheme, $|\zeta|$ must be smaller than 1 for every $\omega$. It was assumed, and numerically confirmed, that the highest value for $|\zeta|$ occurs for $\cos(\omega\Delta x) = -1$, i.e. for the highest grid frequency, $\omega\Delta x = \pm\pi$. Then

$$\begin{aligned} &1 - 16A + 128A^2 - 512A^3 + 1024A^4 \leq 1, \\ &A \leq \frac{1}{4}. \end{aligned}$$

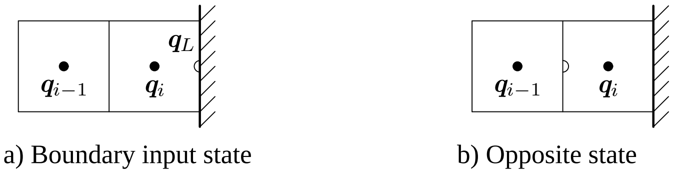
So the new stability requirement is even stricter than the monotonicity requirement (6.12),

$$|\lambda|_{max}\frac{\Delta t}{\Delta x} \leq \frac{1}{4}. \tag{6.19}$$

Numerical experiments in chapter 8 show, however, that the nonlinear scheme (6.11) is sometimes stable at higher values of $A$. Especially for uniform grids, the scheme can be run with $A$ up to $\frac{1}{2}$. This is probably caused by the stabilising influence of the limiter. But for solutions on adapted grids, the CFL number must be chosen below $\frac{1}{4}$ to ensure stability.

## 6.4. Boundary conditions

Boundary conditions for the second-order algorithm are imposed as in section 2.4, a boundary state $\boldsymbol{q}_B$ is calculated with Osher's scheme and the boundary flux is set from this $\boldsymbol{q}_B$. Only the input state at the boundary $\boldsymbol{q}_L$ or $\boldsymbol{q}_R$ has to be determined with second-order accuracy. No cell state behind the boundary is known in advance, so the only way to determine $\boldsymbol{q}_L$ or $\boldsymbol{q}_R$ is with the upwind scheme (figure 6.9a).



a) Boundary input state
b) Opposite state

Figure 6.9: Cells needed to determine boundary input state and opposite state.

Another problem is the state on the cell face on the opposite side of the boundary. The normal, limited scheme cannot be used because there is no cell behind the boundary. Therefore, the central scheme is used here. Experience shows that this usually has no influence on the stability or monotonicity of the solution, since the number of boundary cells is small compared to the total number of cells.

When the test program was run, the upwind scheme caused problems in the neighbourhood of shocks on the wall. When very steep gradients occur at the wall, the input state from the upwind scheme was not physical sometimes. To prevent this, the first-order scheme is used here when the pressure becomes nearly zero, that means when $c^2 < 0.01\left(u^2 + v^2\right)$. The number 0.01 was chosen arbitrarily to assure some margin. On a typical 16.000 cell problem, about four cells per time step were calculated like this and the results were not notably affected.

# Chapter 7
# Second-order accuracy and adaptive gridding

The second-order accurate discretisation of the Euler equations from the previous chapter is combined here with the adaptive-gridding algorithm from chapter 3. Most of the procedures from the algorithm, like the refinement and unrefinement procedure, do not depend on the discretisation used, so no fundamental changes in the algorithm are needed. However, the adaptation of the second-order discretisation to the algorithm is not straightforward: some extra procedures have to be added to the discretisation. The most important of these is the use of virtual states for the flux calculation, as described in section 1. The second section explains how this principle is used for the flux calculation, how the states are advanced and how states in newly refined or unrefined cells are set. The last section gives the revised algorithm.

## 7.1. Virtual states

When we introduced cell face states in section 2.2, it was assumed that the two cells sharing a cell face have equal sizes. The accuracy of the equations that determine $\boldsymbol{q}_L$ and $\boldsymbol{q}_R$ (like equation (2.20)) depend on this assumption: the replacement of an integral of $\boldsymbol{q}$ over a cell face by a single extrapolated value $\boldsymbol{q}_L$ or $\boldsymbol{q}_R$ is only second-order accurate when the integral runs over the entire cell face. This is not the case for a cell lying next to a smaller cell.

A correct way to handle the case of two neighbour cells with different sizes is the concept of virtual states, as described by Van der Maarel [13]. When the fluxes are calculated, the big cell is thought to be divided in four smaller 'virtual' cells (figure 7.1). The states in these cells are determined by interpolation between the states in the surrounding 'real' cells. Unlike the big cell, these virtual cells have the same size as the small neighbour cell, so they can be used for the flux calculation.

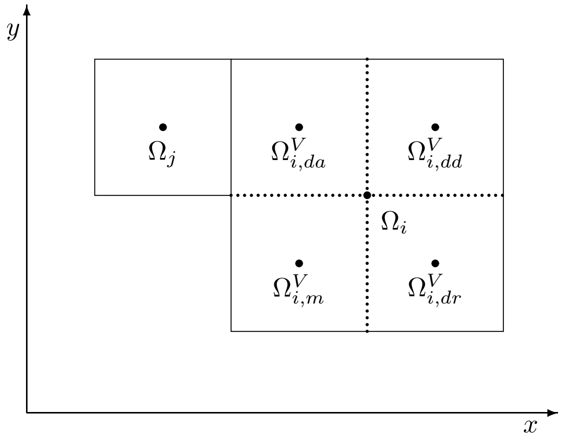


Figure 7.1: Small cell $\Omega_j$ and large cell $\Omega_i$ with four virtual cells.

Although virtual states were not mentioned for the first-order accurate algorithm, they were in fact used there. The state in the big cell was used without interpolation for the flux calculation, but the first-order accurate approximation to the virtual states *is* the state in the big cell,

$$\boldsymbol{q}^V_{i,d} = \boldsymbol{q}_i. \tag{7.1}$$

However, for higher-order accuracy a more complex interpolation is needed.

*Second-order virtual states* To improve the accuracy of the virtual states to second-order, the state is interpolated between two cells: $\Omega_i$ and the cell $\Omega_{d_i}$ in the diagonal direction of the virtual cell (figure 7.2). The centre of the virtual cell lies on the line between these two cells, so a linear interpolation for the virtual state is second-order accurate

$$\boldsymbol{q}_{i,d}^V = \boldsymbol{q}_i + (\boldsymbol{q}_{d_i} - \boldsymbol{q}_i) \frac{x_{i,d}^V - x_i}{x_{d_i} - x_i}. \tag{7.2}$$

How to calculate the virtual state precisely depends on the relative sizes of the cell $\Omega_i$ and the diagonal cell $\Omega_{d_i}$. Let us take the distance between the centres of the big cell and the virtual cell as a unit length. Then the distance from the centre to the corner of $\Omega_i$ is always two units. Furthermore, if $\Omega_{d_i}$ has the same size as $\Omega_i$, then the total distance between the centres is 4 units, so

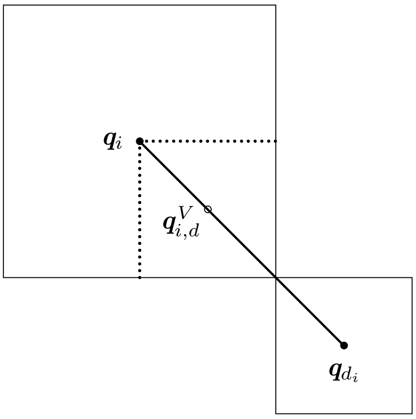


Figure 7.2: Second-order virtual state: interpolation between $\Omega_i$ and diagonal cell $\Omega_{d_i}$.

$$\boldsymbol{q}_{i,d}^V = \boldsymbol{q}_i + \tfrac{1}{4} (\boldsymbol{q}_{d_i} - \boldsymbol{q}_i) .$$

This is the virtual state used by Van der Maarel [13]. When the two cells may have different sizes, the distance between the centre and the corner of $\Omega_{d_i}$ can be expressed in the level difference between $\Omega_i$ and $\Omega_{d_i}$: it is $2^{l_i - l_{d_i} + 1}$ unit lengths. So the virtual state can be calculated, for any cell sizes, as

$$\boldsymbol{q}_{i,d}^V = \boldsymbol{q}_i + \frac{1}{2 + 2^{l_i - l_{d_i} + 1}} (\boldsymbol{q}_{d_i} - \boldsymbol{q}_i) . \tag{7.3}$$

As an example, consider the situation from figure 7.2. Here, the diagonal cell is one level smaller than $\Omega_i$, so $l_{d_i} = l_i + 1$. Therefore, $\frac{1}{2+2^{l_i - l_{d-i}+1}}$ is $\frac{1}{3}$. This is correct, as the distance from the centre of $\Omega_i$ to the centre of $\Omega_{d_i}$ is three unit lengths.

*Diagonal cell is virtual* When the cell $\Omega_{d_i}$ is larger than $\Omega_i$ and when $\Omega_{d_i}$ is a *neighbour* of $\Omega_i$, then the required diagonal state is again a virtual state. In a computer code, this virtual diagonal state can be calculated with the normal virtual state routine, if this routine is made to invoke itself recursively. But let us look at the situations where a diagonal state is virtual:

Whether a diagonal state is virtual, depends on the position of the original virtual cell. To calculate the fluxes in a cell face, all the states in the stencil from figure 6.8 are needed. Depending on the adapted grid, all cells in figure 6.8 may be virtual, except $\Omega_i$ itself. As an example, take the cells on the right side of $\Omega_i$. If $\Omega_{i+1}$

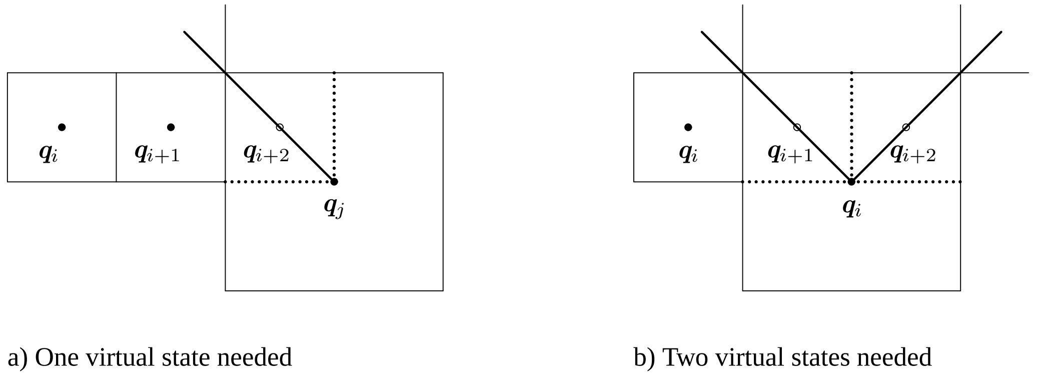


Figure 7.3: Flux calculation, one or two virtual states are needed. Compare this figure with the stencil from figure 6.8.

has the same size as $\Omega_i$, but its right neighbour is bigger, then $\Omega_{i+2}$ is a virtual state (figure 7.3). The diagonal cell needed to calculate this virtual state is the upper neighbour of $\Omega_{i+1}$ and, as this cell may be only one level bigger than $\Omega_{i+1}$, it has at least the same level as the big cell $\Omega_j$ in figure 7.3a. Therefore, this diagonal cell is *never* virtual, as a virtual diagonal state appears only in a cell that is bigger than $\Omega_j$.

When both $\Omega_{i+1}$ and $\Omega_{i+2}$ lie in a big cell $\Omega_j$ (figure 7.3b), two virtual states need to be calculated. The virtual state for $\Omega_{i+1}$ is the same as the virtual state for $\Omega_{i+2}$ in figure 7.3a and is already discussed. But the virtual state for $\Omega_{i+2}$ is new and this virtual state could have a virtual diagonal state. This happens in two cases, in this example

- when the upper neighbour $\Omega_{a_j}$ of $\Omega_j$ is larger than $\Omega_j$ itself, figure 7.4a,
- when the right neighbour $\Omega_{r_j}$ of $\Omega_j$ is larger than $\Omega_j$ itself, figure 7.4b.

In both these figures, we see the original virtual state in $\Omega_j$ and the diagonal virtual state, which is one level bigger, in $\Omega_{n_j}$.

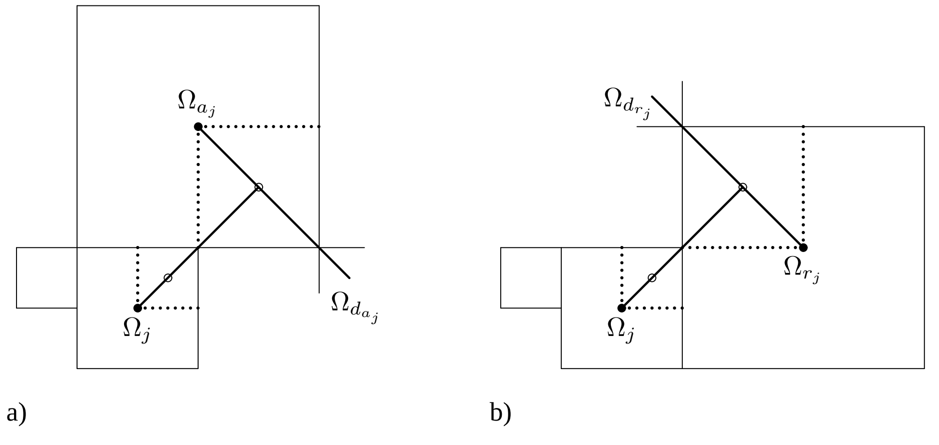
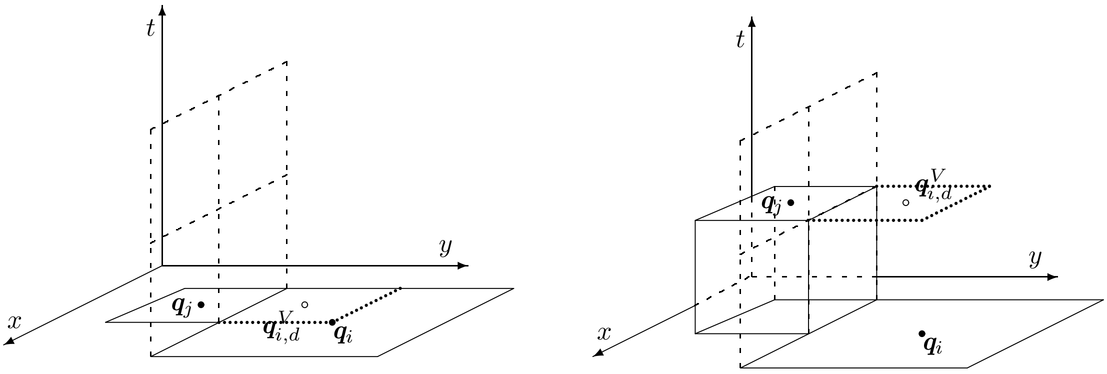


Figure 7.4: The diagonal cell is virtual: two possibilities.

The diagonal cell for this diagonal virtual state is *never* a virtual cell itself. To understand this, look at this cell ($\Omega_{d_{a_j}}$ or $\Omega_{d_{r_j}}$, drawn partially in figure 7.4). Between this diagonal cell and $\Omega_j$ lies only one cell, that has the same size as $\Omega_j$. It cannot be smaller than $\Omega_j$, because the larger cell $\Omega_{r_j}$ or $\Omega_{a_j}$ is its neighbour and it cannot be bigger than $\Omega_j$, because it would not fit in. The diagonal cell $\Omega_{d_{a_j}}$ or $\Omega_{d_{r_j}}$ is a neighbour of this cell, so the diagonal cell can be only one level bigger than this cell, i.e. the same size as $\Omega_{r_j}$ or $\Omega_{a_j}$. It cannot be bigger than $\Omega_{d_{r_j}}$ or $\Omega_{d_{a_j}}$, so the diagonal cell cannot be virtual.

This leads to an important conclusion: to calculate a virtual state, it is sometimes necessary to calculate one more virtual state. But *more* than one virtual state is never needed.

*Virtual timestepping* The virtual state of equation (7.3) is, in fact, only valid for steady problems. But for the current unsteady algorithm, when the fluxes for a small level are calculated for the second of two time steps, the state in the neighbouring big cell is not known at that time level (see figure 7.5, that can be seen as a time-dependent version of figure 7.1).



a) First time step for $\boldsymbol{q}_j$

b) Second time step for $\boldsymbol{q}_j$

Figure 7.5: A virtual state in a big cell $\Omega_i$ at two time levels.

The solution to this problem is to advance the virtual cell as if it were a normal cell, with flux differences. But this gives one problem: the fluxes into the virtual cells are not known. Therefore, a trick is used. The virtual cells are not advanced, but the big cell $\Omega_i$ is advanced half a time step (one step for the small cells) and the resulting $\Delta\boldsymbol{q}^V_i$ is used to advance all four virtual states. This is allowed: $\Delta\boldsymbol{q}^V_i$ is of $\mathcal{O}(\Delta t)$, so its spatial variation in cell $\Omega_i$ is of $\mathcal{O}(\Delta t \Delta x)$. Therefore, $\Delta\boldsymbol{q}^V_i$ is a second-order accurate approximation to the $\Delta\boldsymbol{q}$ that appears when the virtual cells are advanced themselves.

Combining equations (7.3) and (3.4), the virtual state equation becomes

$$\boldsymbol{q}^V_{i,d} = \begin{cases} \boldsymbol{q}_i + \frac{1}{2+2^{l_{d_i}-l_i+1}}\left(\boldsymbol{q}_{d_i} - \boldsymbol{q}_i\right) & \text{first time step,} \\ \boldsymbol{q}_i + \frac{1}{2+2^{l_{d_i}-l_i+1}}\left(\boldsymbol{q}_{d_i} - \boldsymbol{q}_i\right) - \frac{1}{2}\left(\boldsymbol{F}_{O\,i,r} - \boldsymbol{F}_{O\,i,l}\right)\frac{\Delta t}{\Delta x} & \\ \qquad -\frac{1}{2}\left(\boldsymbol{G}_{O\,i,a} - \boldsymbol{G}_{O\,i,b}\right)\frac{\Delta t}{\Delta y} & \text{second time step.} \end{cases} \tag{7.4}$$

Note that the first-order accurate Euler discretisation is used for $\Delta\boldsymbol{q}^V_i$, but for a single $\Delta\boldsymbol{q}$ of $\mathcal{O}(\Delta t)$ this discretisation gives an error of $\mathcal{O}(\Delta t^2)$, which is small enough.

A last question is which fluxes to use. All fluxes into neighbour cells that are the same as or bigger than $\Omega_i$ are known, since they have been calculated after the last time step for $\Omega_i$. But the fluxes in those cell faces that lie next to smaller cells are not known yet, as these fluxes are averaged over the first and second time step of the small cells. Therefore, on a face that lies next to two small cells, the average flux in the two small cells for their first time step is used.

*Cells at the boundary* When a virtual state is needed in a cell that lies next to the boundary, the diagonal cell may be a boundary cell (figure 7.6). In that case, a special procedure is used to calculate the diagonal state $\boldsymbol{q}_{d_i}$.

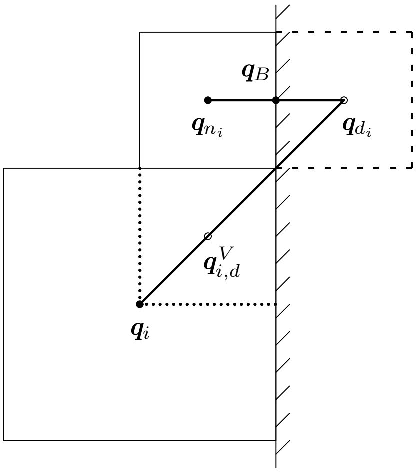


Figure 7.6: Diagonal state for boundary cells.

The calculation starts with a neighbour cell $\Omega_{n_i}$ of $\Omega_i$ that also lies next to the boundary. The state at the boundary $\boldsymbol{q}_B$ in this cell is calculated as usual (see sections 2.4 and 6.4) but the boundary state is not used to calculate a boundary flux. Instead, a state in a mirror cell behind the boundary is extrapolated,

$$\boldsymbol{q}_{d_i} = 2\boldsymbol{q}_B - \boldsymbol{q}_{n_i}. \tag{7.5}$$

This state is used in equation (7.4), no other changes are needed. $l_{d_i}$ is the same as the level of the neighbour cell $\Omega_{n_i}$.

It may seem dangerous to use such an extrapolation, as it is not certain that $\boldsymbol{q}_{d_i}$ is physical: if $\boldsymbol{q}_{n_i} \gg \boldsymbol{q}_B$, then $\rho$ and $\rho E$ may be negative in $\boldsymbol{q}_{d_i}$. But $\boldsymbol{q}_{d_i}$ is not used for flux calculation itself, it is only used to set $\boldsymbol{q}^V_{i,d}$ and no problems arise as long as $\boldsymbol{q}^V_{i,d}$ is physically possible. The present method is equivalent to setting $\boldsymbol{q}^V_{i,d}$ from a plane through $\boldsymbol{q}_i$, $\boldsymbol{q}_{n_i}$ and $\boldsymbol{q}_B$ and $\boldsymbol{q}^V_{i,d}$ is not far away from the line between $\boldsymbol{q}_i$ and $\boldsymbol{q}_B$, so $\boldsymbol{q}^V_{i,d}$ is nearly always physically possible.

## 7.2. State and flux treatment

### 7.2.1 Flux evaluation

In the previous section, the virtual state is described, which is the most important change in the flux calculation algorithm. But there are more changes, both in the way the fluxes are calculated and in the way in which fluxes from small cells are summed to give the flux into a big cell.

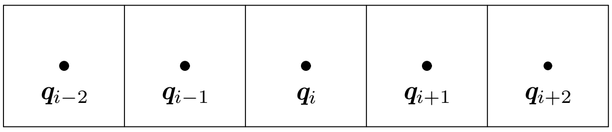


Figure 7.7: Five-cell basic stencil for flux calculation.

The principle of the flux calculation remains the same, like it was described in section 3.4: fluxes are calculated per face and stored per face. The procedure depends on the size of the neighbour cells, the flux is calculated and stored in the neighbour face too if the cell has the same size, it is calculated and added to the flux in the neighbour cell face when this cell is bigger. In this way, the flux in the big cell is averaged over the small cells. This means that nothing has to be done when the neighbour cells are smaller.

But with the second-order flux discretisation, more cells are needed for the flux calculation: four cells per cell face instead of two. In the computer code, the fluxes at two opposite faces are calculated at the same time. Together, they need five cell states (figure 7.7, c.f. equation (6.3)). In the code, these five states are determined first and then the fluxes at the two cell faces are calculated. Determination of these five states is not always straightforward on an adapted grid, because any of the four cells $\Omega_{i-2}$, $\Omega_{i-1}$, $\Omega_{i+1}$ and $\Omega_{i+2}$ could lie in a bigger cell or be split into finer cells. All these cell arrangements must be transformed to the uniform five-point stencil of figure 7.7. When any of the neighbour cells is split into smaller cells, it is not necessary to calculate the flux at one of the faces, which influences the way the stencil is filled too. How the states in the stencil are determined, for all possible configurations, is described below.

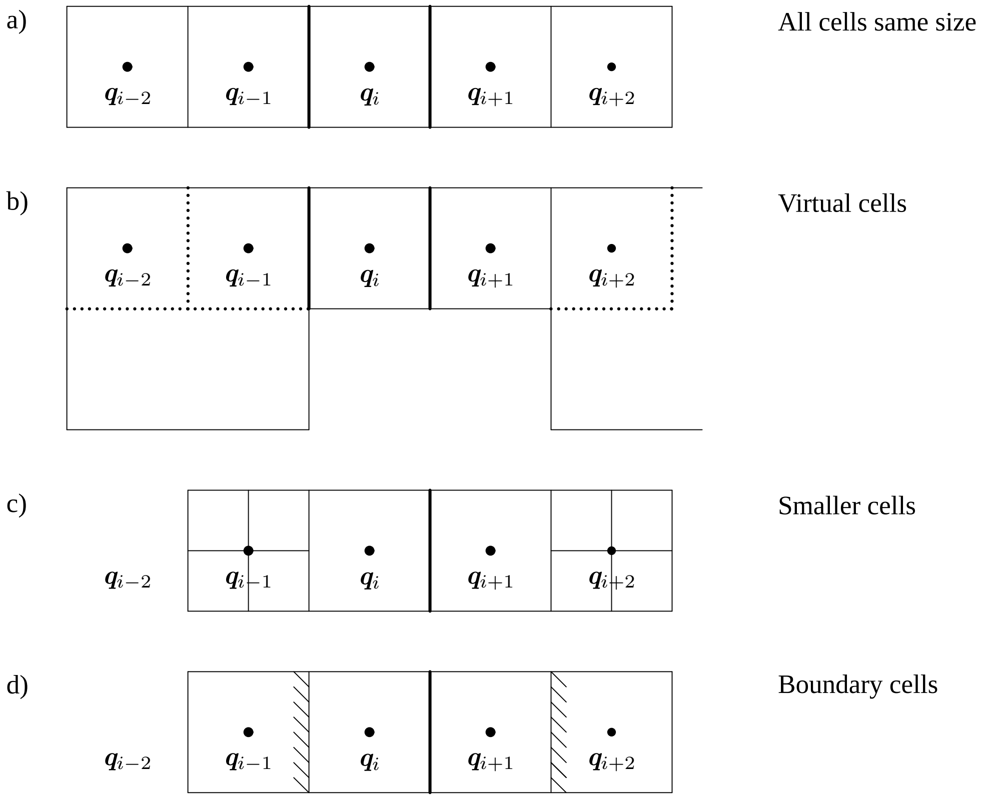


Figure 7.8: All possible configurations for the five-point stencil of figure 7.7. Any combination of the left and right sides of each picture may appear in a refined grid. ●: state is needed, |: flux is calculated.

*Cells have same size* When all cells in the stencil have the same size, the stencil can be filled in a straightforward way (figure 7.8a). The fluxes across both cell faces are calculated.

*Virtual cells* When the neighbour cells are larger, the stencil is filled with virtual cells (figure 7.8b). As already shown in figure 7.3, there are two possible configurations: both cells on one side of $\Omega_i$ are virtual ($\boldsymbol{q}_{i-2}$

and $\boldsymbol{q}_{i-1}$ on the left side of figure 7.8b) or only the outer cell is virtual ($\boldsymbol{q}_{i+2}$ on the right side of figure 7.8b). In both cases, the flux at the cell face is calculated.

*Smaller cells* When the neighbour cells on one cell face of $\Omega_i$ are smaller than $\Omega_i$, no fluxes are calculated at that cell face (figure 7.8c, left side). But the stencil must be filled there anyway. Actually, the outer state $\boldsymbol{q}_{i-2}$ in figure 7.8c is not needed, but the inner state $\boldsymbol{q}_{i-1}$ is, to calculate the flux at the right cell face of $\Omega_i$. Also, if one of the outer cells in the stencil is filled with smaller cells (figure 7.8c, right side), then the state ($\boldsymbol{q}_{i+2}$) in this stencil position is still needed. The way to determine the states in these stencil positions is to average the states in the small cells, as if they are being unrefined. When all the cells in a stencil position are only one level smaller than $\Omega_i$, their average state can really be calculated as if the four cells were merged, with equation (3.7). But two of the four cells may be refined themselves and some of those cells may be refined again etc. Therefore, when one of the four cells is refined, the average state on that half of the imaginary big cell is interpolated between three cells, see figure 7.9, middle image (the other refined cells are not used, because they may be refined again). There, the average state in the upper cells is estimated as

$$\begin{aligned}\boldsymbol{q}_{av,\text{upper}} &= \tfrac{2}{3}\left(\tfrac{1}{2}\left(\boldsymbol{q}_3 + \boldsymbol{q}_4\right)\right) + \tfrac{1}{3}\boldsymbol{q}_5, \\ &= \tfrac{1}{3}\left(\boldsymbol{q}_3 + \boldsymbol{q}_4 + \boldsymbol{q}_5\right), \end{aligned} \tag{7.6}$$

so the average state in the big cell is

$$\boldsymbol{q}_{av} = \tfrac{1}{4}\left(\boldsymbol{q}_1 + \boldsymbol{q}_2\right) + \tfrac{1}{6}\left(\boldsymbol{q}_3 + \boldsymbol{q}_4 + \boldsymbol{q}_5\right). \tag{7.7}$$

If *all* outer cells are refined again (figure 7.9, lower image) then equation (7.6) is used twice,

$$\boldsymbol{q}_{av} = \tfrac{1}{6}\left(\boldsymbol{q}_1 + \boldsymbol{q}_2 + \boldsymbol{q}_3 + \boldsymbol{q}_4 + \boldsymbol{q}_5 + \boldsymbol{q}_6\right). \tag{7.8}$$

The average state for any combination of refined and unrefined small cells can be calculated with these equations.

*Boundary cells* The equations to handle the fluxes near a boundary have been discussed in section 6.4. If a cell face lies on the boundary (figure 7.8d, left), the flux is calculated with a special set of equations and the state on the opposite cell face is set with a central scheme. But with a trick, the normal limited scheme can be used to set this central state, which makes the procedure easier to implement in a computer code. This trick is to set a ‘dummy’ state in the stencil position behind the boundary (like $\boldsymbol{q}_{i-1}$ in figure 7.8d). This state contains no new information, but it is chosen such that the limited scheme gives the same cell face state that a central scheme would have given. The chosen state is a linear extrapolation from the cell next to the boundary and the next cell. For example, $\boldsymbol{q}_{i-1}$ in figure 7.8d is set as

$$\boldsymbol{q}_{i-1} = 2\boldsymbol{q}_i - \boldsymbol{q}_{i+1}. \tag{7.9}$$

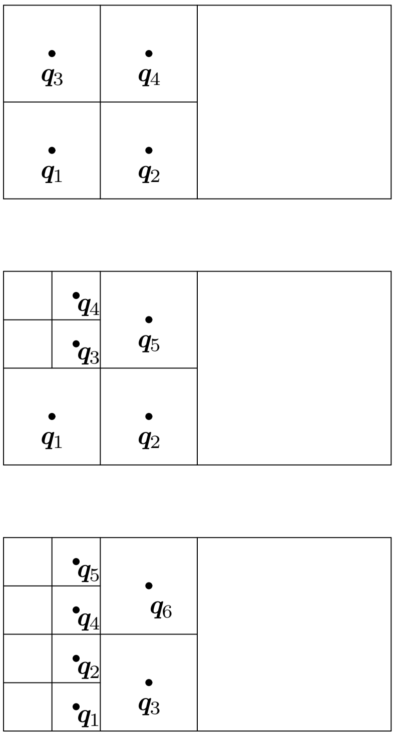
Then $\boldsymbol{q}_i - \boldsymbol{q}_{i-1} = \boldsymbol{q}_{i+1} - \boldsymbol{q}_i$ and $r_L$ for $\boldsymbol{q}_L$ on the face between $\Omega_i$ and $\Omega_{i+1}$ is always 1, which means that $\phi = 1$. The limited scheme (6.3) becomes

$$\begin{aligned} q_{L,p} &= q_{i,p} + \tfrac{1}{2}\left(q_{i,p} - q_{i-1,p}\right), \\ &= q_{i,p} + \tfrac{1}{2}\left(q_{i+1,p} - q_{i,p}\right), \\ &= \tfrac{1}{2}\left(q_{i,p} + q_{i-1,p}\right). \end{aligned}$$

The limited scheme is equivalent to the central scheme.

All the necessary states in the stencil are now filled. The cell face states are calculated with the limited scheme (6.3) and then the fluxes are calculated with Osher’s scheme.

A last point of concern is the flux into a big cell next to two smaller cells. This flux must be conservative, the total inflow in the big cell must equal the total outflow from the two small cells. But for the modified Richtmyer



$$\boldsymbol{q}_{av} = \tfrac{1}{4}\left(\boldsymbol{q}_1 + \boldsymbol{q}_2 + \boldsymbol{q}_3 + \boldsymbol{q}_4\right)$$

$$\boldsymbol{q}_{av} = \tfrac{1}{4}\left(\boldsymbol{q}_1 + \boldsymbol{q}_2\right) + \tfrac{1}{6}\left(\boldsymbol{q}_3 + \boldsymbol{q}_4 + \boldsymbol{q}_5\right)$$

$$\boldsymbol{q}_{av} = \tfrac{1}{6}\left(\boldsymbol{q}_1 + \boldsymbol{q}_2 + \boldsymbol{q}_3 + \boldsymbol{q}_4 + \boldsymbol{q}_5 + \boldsymbol{q}_6\right)$$

Figure 7.9: State averaging in smaller cells, three possible cases.

scheme, the state after the first step is only used to calculate the fluxes for the second step. The state change that is kept, is the change in the second step. Therefore, only the fluxes for this step have to be strictly conservative.

Let us again consider the situation of section 3.4: two small cells $\Omega_i$ and $\Omega_{i_2}$ on level $l$ lie next to cell $\Omega_{n_i}$ on level $l-1$. All cells are at time level $k$. The evaluation of the fluxes is shown in figure 7.10: first, the small cells are advanced to $k+1$ and the fluxes $\boldsymbol{f}^{k+1}$ are calculated. Nothing is done with the big cell. In the second, ‘Leapfrog’ step, the states $\boldsymbol{q}_i^{k+2}$ and $\boldsymbol{q}_{i_2}^{k+2}$ are set from $\boldsymbol{q}_i^k$ and $\boldsymbol{q}_{i_2}^k$ with the fluxes $\boldsymbol{f}^{k+1}$. At the same time, $\Omega_{n_i}$ is advanced for the first time with a forward-Euler step. The flux used is chosen conservative: the inflow and outflow $\boldsymbol{f}\Delta y_i \Delta t_i$ must match:

$$\boldsymbol{f}^k_{n_i,n+2}\, 2^{-(l-1)}\Delta y\, 2^{-(l-1)}\Delta t = 2\left(\boldsymbol{f}^{k+1}_{i,n} + \boldsymbol{f}^{k+1}_{i_2,n}\right) 2^{-l}\Delta y\, 2^{-l}\Delta t,$$

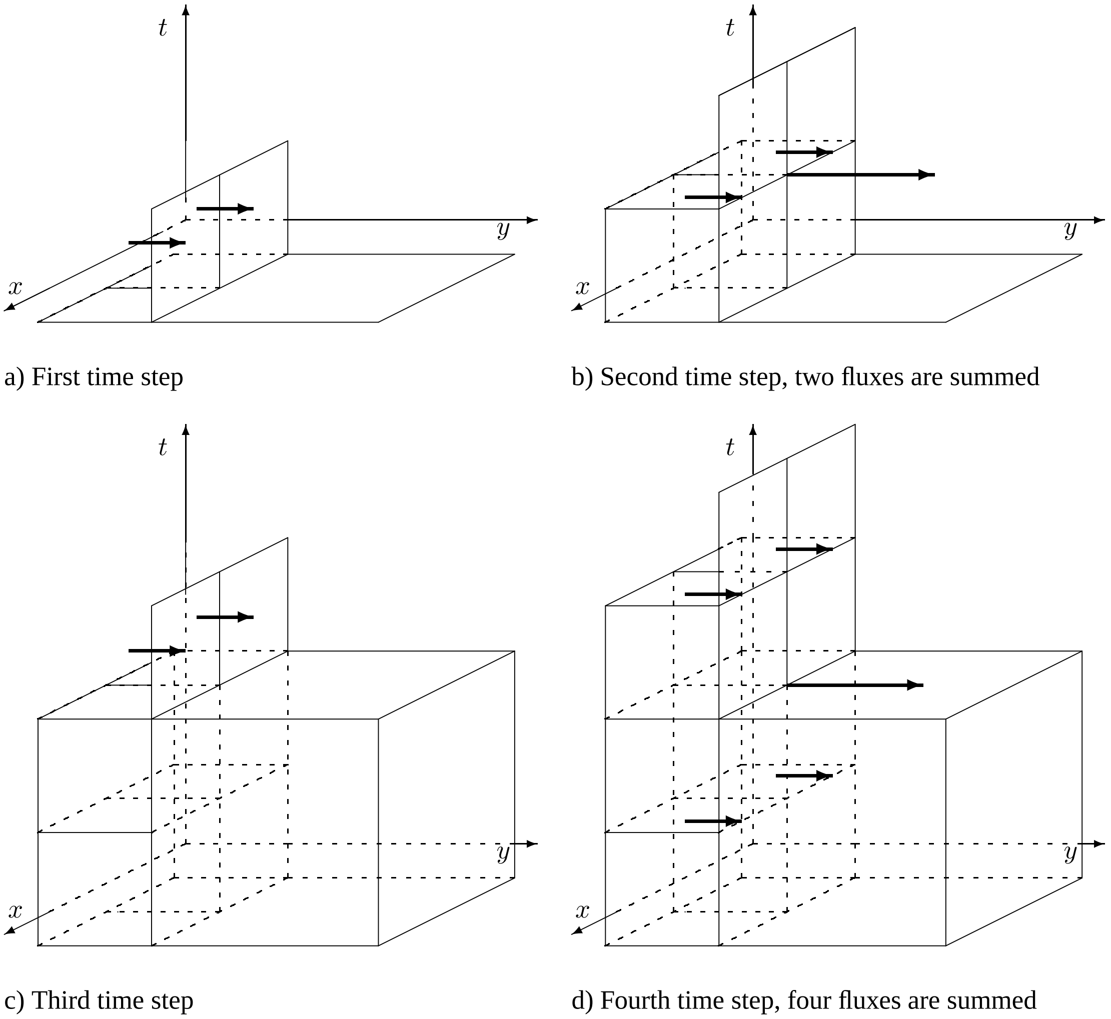
so

$$\boldsymbol{f}^k_{n_i,n+2} = \tfrac{1}{2}\left(\boldsymbol{f}^{k+1}_{i,n} + \boldsymbol{f}^{k+1}_{i_2,n}\right). \tag{7.10}$$

Then the small cells are advanced again, $\boldsymbol{q}_i^{k+3}$ and $\boldsymbol{q}_{i_2}^{k+3}$ are set, the $\boldsymbol{f}^{k+3}$ are calculated and $\boldsymbol{q}_i^{k+4}$ and $\boldsymbol{q}_{i_2}^{k+4}$ are set from $\boldsymbol{q}_i^{k+2}$ and $\boldsymbol{q}_{i_2}^{k+2}$. Now $\boldsymbol{q}_{n_i}^{k+4}$ is set with one huge Leapfrog step from $\boldsymbol{q}_{n_i}^k$ and it is essential that the fluxes used are conservative. *All* the inflow from the two second steps of the small cells must be summed,

$$\begin{aligned}
&2\boldsymbol{f}^{k+2}_{n_i,n+2}\, 2^{-(l-1)}\Delta y\, 2^{-(l-1)}\Delta t = 2\left(\boldsymbol{f}^{k+1}_{i,n} + \boldsymbol{f}^{k+1}_{i_2,n} + \boldsymbol{f}^{k+3}_{i,n} + \boldsymbol{f}^{k+3}_{i_2,n}\right) 2^{-l}\Delta y\, 2^{-l}\Delta t,\\
&\boldsymbol{f}^{k+2}_{n_i,n+2} = \tfrac{1}{4}\left(\boldsymbol{f}^{k+1}_{i,n} + \boldsymbol{f}^{k+1}_{i_2,n} + \boldsymbol{f}^{k+3}_{i,n} + \boldsymbol{f}^{k+3}_{i_2,n}\right).
\end{aligned} \tag{7.11}$$

Concluding, the first flux into the big cell is the average of the small fluxes. In the computer code, the fluxes from the small cells are calculated, stored in the small cell and added to the flux in the big cell, but divided by



a) First time step

b) Second time step, two fluxes are summed

c) Third time step

d) Fourth time step, four fluxes are summed

Figure 7.10: Summation of fluxes in a cell face between two cells with different sizes.

two (for the first step in the big cell). After the first step for the big cell, the flux in the big cell face is halved and the fluxes from the small cells are again added, but now divided by four. The second step for the big cell is calculated with this flux.

### *7.2.2 State initialisation*

When a cell is split in four, or when four cells are merged into one cell, the state in the new cell(s) must be set. The procedures to do this are described here.

*Refinement* In section 3.4, the state in newly refined cells was set with a first-order approximation: the states are chosen equal to the state in the old big cell. For a second-order discretisation, a more accurate approximation is needed.

The tool to achieve this is already discussed in this chapter: virtual states. The most straightforward solution is to set the states of the new cells at level $l+1$ equal to the four virtual states in the old cell at level $l$,

$$\boldsymbol{q}_i^{l+1} = \boldsymbol{q}_{i,m}^V \qquad \boldsymbol{q}_{dr}^{l+1} = \boldsymbol{q}_{i,dr}^V \qquad \boldsymbol{q}_{dd}^{l+1} = \boldsymbol{q}_{i,dd}^V \qquad \boldsymbol{q}_{da}^{l+1} = \boldsymbol{q}_{i,da}^V. \tag{7.12}$$

But this discretisation has a great disadvantage. It is not necessarily conservative. The cells are refined between time steps, so nothing flows out of the big cell during refinement. Therefore, the total amount of matter, momentum and energy in the four new cells *must* be equal to the total amount in the old big cell, otherwise these quantities are not conserved. In other words, the average of the four states in the new cells must be equal to the state in the old cell. This is not true for the virtual states, so another equation is needed.

Let us look at the virtual state in more detail. What happens is, that a linear interpolation is constructed between $\Omega_i$ and the diagonal state $\Omega_{d_i}$ and then the state in the virtual cell is calculated by substituting the coordinate of the virtual cell in this approximation (equation (7.2)). Take, for example, the state in the upper right corner $\boldsymbol{q}^V_{i,dd}$. If the solution is sufficiently smooth, equation (7.2) can be written as

$$\begin{aligned} \boldsymbol{q}^V_{i,dd} &= \boldsymbol{q}_i + \Big( (\boldsymbol{q}_i)_x + (\boldsymbol{q}_i)_y + \tfrac{1}{2} \Big( (\boldsymbol{q}_i)_{xx} + 2\,(\boldsymbol{q}_i)_{xy} + (\boldsymbol{q}_i)_{yy} \Big) (x_{dd_i} - x_i) + \mathcal{O}\left(h^2\right) \Big) \\ &\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \left(x^V_{i,dd} - x_i\right) , \\ &= \boldsymbol{q}_i + \boldsymbol{d}_{i,dd} \left(x^V_{i,dd} - x_i\right) , \end{aligned} \tag{7.13a}$$

For the virtual cell in the lower left corner, equation (7.2) becomes

$$\begin{aligned} \boldsymbol{q}^V_{i,m} &= \boldsymbol{q}_i + \Big( (\boldsymbol{q}_i)_x + (\boldsymbol{q}_i)_y - \tfrac{1}{2} \Big( (\boldsymbol{q}_i)_{xx} + 2\,(\boldsymbol{q}_i)_{xy} + (\boldsymbol{q}_i)_{yy} \Big) (x_i - x_{m_i}) + \mathcal{O}\left(h^2\right) \Big) \\ &\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \left(x^V_{i,m} - x_i\right) , \\ &= \boldsymbol{q}_i + \boldsymbol{d}_{i,m} \left(x^V_{i,m} - x_i\right) . \end{aligned} \tag{7.13b}$$

As both $(x^V_{i,dd} - x_i)$ and $(x^V_{i,m} - x_i)$ are $\mathcal{O}(h)$, these equations are second-order accurate. And both linear approximations are valid in the entire cell $\Omega_i$, so by using $x_{i,m}$ in equation (7.13a), instead of $x_{i,dd}$, this equation can be used too for calculating $\boldsymbol{q}^V_{m,i}$. We may conclude that there is a range of linear approximations $\boldsymbol{q}^V_{i,d} = \boldsymbol{q}_i + \boldsymbol{d}(x^V_{i,d} - x_i)$, with $\boldsymbol{d} \in [\boldsymbol{d}_m, \boldsymbol{d}_{dd}]$, that give $\boldsymbol{q}^V_{i,m}$ and $\boldsymbol{q}^V_{i,dd}$ with second-order accuracy.

Now another important conclusion is drawn: if $\boldsymbol{q}^V_{i,m}$ and $\boldsymbol{q}^V_{i,dd}$ are set with the *same* linear interpolation, then their average is equal to $\boldsymbol{q}_i$. So if one interpolation is used for both $\boldsymbol{q}^V_{i,m}$ and $\boldsymbol{q}^V_{i,dd}$ and another for both $\boldsymbol{q}^V_{i,dr}$ and $\boldsymbol{q}^V_{i,da}$, then the four virtual states are conservative. Therefore, equation (7.12) is used first to set the states in newly split cells, but it is followed by another step, where the new states are made conservative: each pair of diagonal virtual states is set from the interpolation for one of these states. For this, the interpolation with the smallest $|\boldsymbol{d}|$ is chosen. And if the two $\boldsymbol{d}$'s have opposite signs, i.e. $\boldsymbol{q}_i$ is a minimum or a maximum, then an intermediate $\boldsymbol{d}$ is used: $\boldsymbol{d} = 0$ (note that this procedure is similar to the Minmod limiter). So, per component of $\boldsymbol{q}$ (dropping the superscripts $l$),

$$q^{l+1}_{d,p} = \begin{cases} q_{i,p} - \min\left(\left(q^V_{i,d+2,p} - q_{i,p}\right), \left(q_{i,p} - q^V_{i,d,p}\right)\right) & \text{if } \frac{q^V_{i,d+2,p} - q_{i,p}}{q_{i,p} - q^V_{i,d,p}} \geq 0, \\ q_{i,p} & \text{otherwise,} \end{cases} \tag{7.14a}$$

$$q^{l+1}_{d+2,p} = 2q_{i,p} - q^{l+1}_{d,p}, \tag{7.14b}$$

$$d = m, dr \qquad m + 2 = dd \qquad dr + 2 = da.$$

After this second step, the average of the states in the four new cells is equal to the state in the old cell.

To allow the scheme to follow waves as efficiently as possible, it is essential that cells can be split after every time step. And to keep a sufficient accuracy, the state in the big cells must be second-order accurate before they are split. But the two-step scheme for the time-stepping works such that the states set with the second, Leapfrog step are second-order accurate but the states set with the forward-Euler scheme are first-order accurate. So if a cell is refined when the most recent state in a cell is set with the forward-Euler scheme, then $\boldsymbol{q}_i$ is only first-order accurate. This is corrected by recalculating the fluxes into $\Omega_i$ and changing the state $\boldsymbol{q}^{k+1}_i$ to

$$\boldsymbol{q}^{k+1}_{i,\text{corr}} = \boldsymbol{q}^k_i + \frac{1}{2}\left(\boldsymbol{q}^{k+1}_i - \boldsymbol{q}^k_i\right) - \frac{1}{2}\left(\boldsymbol{f}^{k+1}_{i,r} - \boldsymbol{f}^{k+1}_{i,l}\right)\frac{\Delta t}{\Delta x} - \frac{1}{2}\left(\boldsymbol{g}^{k+1}_{i,a} - \boldsymbol{g}^{k+1}_{i,b}\right)\frac{\Delta t}{\Delta y}. \tag{7.15}$$

$\boldsymbol{q}_{i,\text{corr}}$ is second-order accurate. These states are calculated for all cells on a level that have to be split, before the new states in the small cells are found with equation (7.14). This correction is needed formally, but numerical results show that it has almost no effect. It is expected that this correction can be removed without consequences.

*Unrefinement* Setting the state in a newly unrefined cell is straightforward. Equation (3.7) is already second-order accurate, so it can be used without changes. Only one problem is encountered: when cells at level $l$ are merged into one cell at level $l-1$ and the other cells in $\Omega^{l-1}$ were just advanced with a forward-Euler step, then a ‘previous’ state is needed for the new big cell. This problem is solved simply by not unrefining cells when the next bigger level just had a forward-Euler step.

## 7.3. The new algorithm

The second-order accurate adaptive-gridding discretisation described in the previous sections is specially designed to be used in the existing algorithm of section 3.2. Therefore, only a few changes to this algorithm are needed. The most important change is that two states are stored per cell, instead of one: $\boldsymbol{q}(1)$ is the second-order accurate state at time level $k$, $k+2$, etc. in equation (6.11), $\boldsymbol{q}(2)$ is the intermediate state that is calculated with the forward-Euler time discretisation. Per level, the operations are performed alternatingly on $\boldsymbol{q}(1)$ and $\boldsymbol{q}(2)$.

The algorithm, now called Sagseo[1] is again given in pseudo-code.

```
Program Sagseo
do l = 0 to M times_advanced(l) = 1
do k = 1 to k_max
    call Timestep(M)
end do
end Sagseo

Subroutine Timestep(l_now)
if times_advanced(l_now) = 1 then
    advance $\Omega^{\text{l\_now}}$, $\boldsymbol{q}(1) \Rightarrow \boldsymbol{q}(2)$ with forward-Euler
else
    advance $\Omega^{\text{l\_now}}$, $\boldsymbol{q}(2) \Rightarrow \boldsymbol{q}(1)$ with Leapfrog
end if
if times_advanced(l_now) = 1 or l_now = 0 then
    if l_now = 0 then
        change times_advanced(0) from 1 to 2 or v.v.
    else
        times_advanced(l_now) = 2
    end if
    do i = l_now to M - 1 refine_check $\Omega^i$ with $\boldsymbol{q}$(times_advanced(i))
    do i = l_now + 2 to M unrefine_check $\Omega^i$ with $\boldsymbol{q}$(times_advanced(i))
    do i = l_now to M fluxes $\Omega^i$ with $\boldsymbol{q}$(times_advanced(i))
    if l_now = 0 return, time step finished
    call Timestep(M)
else
    times_advanced(l_now) = 1
    call Timestep(l_now - 1)
end if
end Timestep
```

[1] Solution-Adaptive Gridding, SEcond Order accuracy.

# Chapter 8
# Second-order results

The second-order accurate algorithm developed in the previous chapters is tested here with the same test problems that were used for the first-order algorithm (chapter 5). The three refinement criteria of chapter 4 are tested and the effect of the two limiters from section 6.1 is determined in two cases: for uniform grids (section 8.1) and for the $C_{\partial\rho}$ refinement criterion (section 8.2). The code used is given in [22].

## 8.1. Test problems

The two test problems of section 5.1 are used again, but some changes in the implementation are needed for second-order problems. These changes are described here and reference results for uniform grids are given.

### *8.1.1 Sod problem*

In section 5.1, this problem was modelled on a 1 × many grid. But the current interpolation procedures at the boundary (section 6.4) cannot handle cells with a boundary on two opposite sides. Therefore, the problem is solved here on a 2 × many cells basic grid. This doubles the amount of computation needed, but the number of cells in $x$-direction is counted in one row only, so it does not change.

$\Delta x/\Delta t$ is changed from 4 to 8 for the second-order problems, this value corresponds to a CFL number of 0.28. This is higher than the theoretical maximum (section 6.3), but still low: all calculations on uniform grids were stable up to CFL = 0.5. This may have two causes: the limited flux function adds dissipation and stabilizes the scheme, furthermore it is possible that the most critical error frequencies are not present in the solution. The solutions on the refined grids were less stable, therefore all calculations were made with $\Delta x/\Delta t$ = 8.

Results for the Sod problem are given in figure 8.1 and 8.2. These solutions are significantly more accurate than the first-order results, but their general characteristics are very much the same. The shock is sharp, but the contact discontinuity and the head of the expansion are smeared. This is caused by the limited flux discretisation, which adds numerical viscosity like the first-order scheme (only less). However, there are other errors with a second-order character, like the wiggle behind the shock (visible in the $u$-plots with 50 cells) and the overshoot behind the expansion.

| Cells | 'Level' | $e_\rho$ | | $e_u$ | | $e_p$ | |
|---|---|---|---|---|---|---|---|
| | | $\phi_{MVA}$ | $\phi_{MM}$ | $\phi_{MVA}$ | $\phi_{MM}$ | $\phi_{MVA}$ | $\phi_{MM}$ |
| 50 | 0 | 0.0107 | 0.0135 | 0.0205 | 0.0254 | 0.0087 | 0.0116 |
| 100 | 1 | 0.0061 | 0.0079 | 0.0123 | 0.0151 | 0.0045 | 0.0062 |
| 200 | 2 | 0.0032 | 0.0043 | 0.0061 | 0.0076 | 0.0022 | 0.0031 |
| 400 | 3 | 0.0018 | 0.0024 | 0.0031 | 0.0039 | 0.0011 | 0.0016 |
| 800 | 4 | 0.0010 | 0.0014 | 0.0017 | 0.0020 | 0.0006 | 0.0008 |

Table 8.1: Sod problem, errors in the solution on uniform grids, with two limiter functions, $t = 0.1$. The number of time steps is set from $\Delta x/\Delta t = 8$. The 'level' indicated is the level of the grid cells, if they are created by splitting a 50 cell basic grid.

The plots with the modified Van Albada limiter (figure 8.1) and the Minmod limiter (figure 8.2) show no difference. However, there is a difference in accuracy. A table of the global errors from equation (5.1), table 8.1, shows that the Van Albada limiter is more accurate in all cases. Note that the errors decrease almost exactly inversely proportional with the grid size: the method is first-order accurate in practice. This is usual for problems with shocks, all second-order methods studied by Woodward and Colella [24] show the same behaviour.

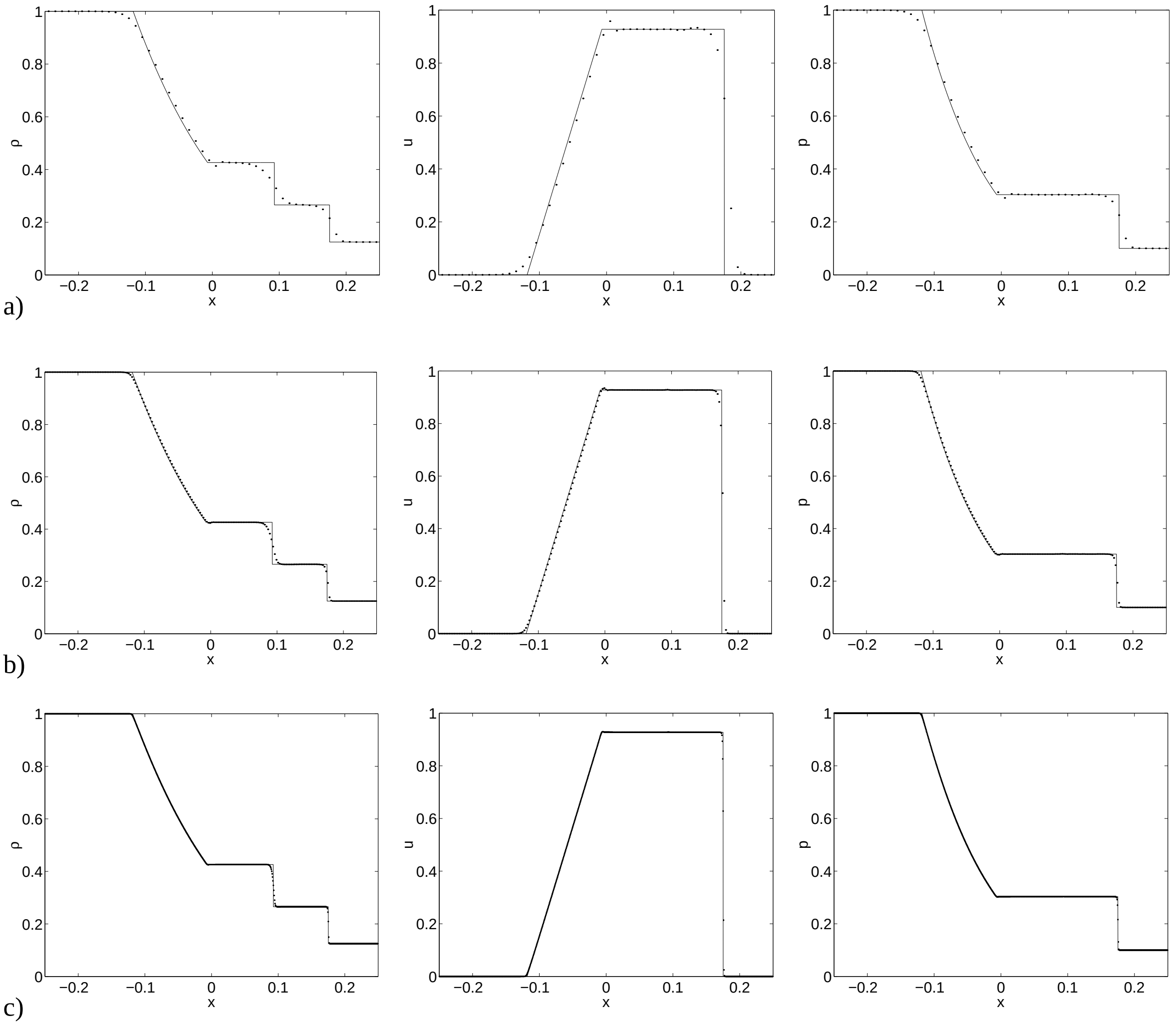


Figure 8.1: Sod problem, solution on uniform grids with 50 (a), 200 (b) and 800 (c) cells. Modified Van Albada limiter. Domain $x = [-0.25, 0.25]$, $t = 0.1$, $\Delta x/\Delta t = 8.0$.

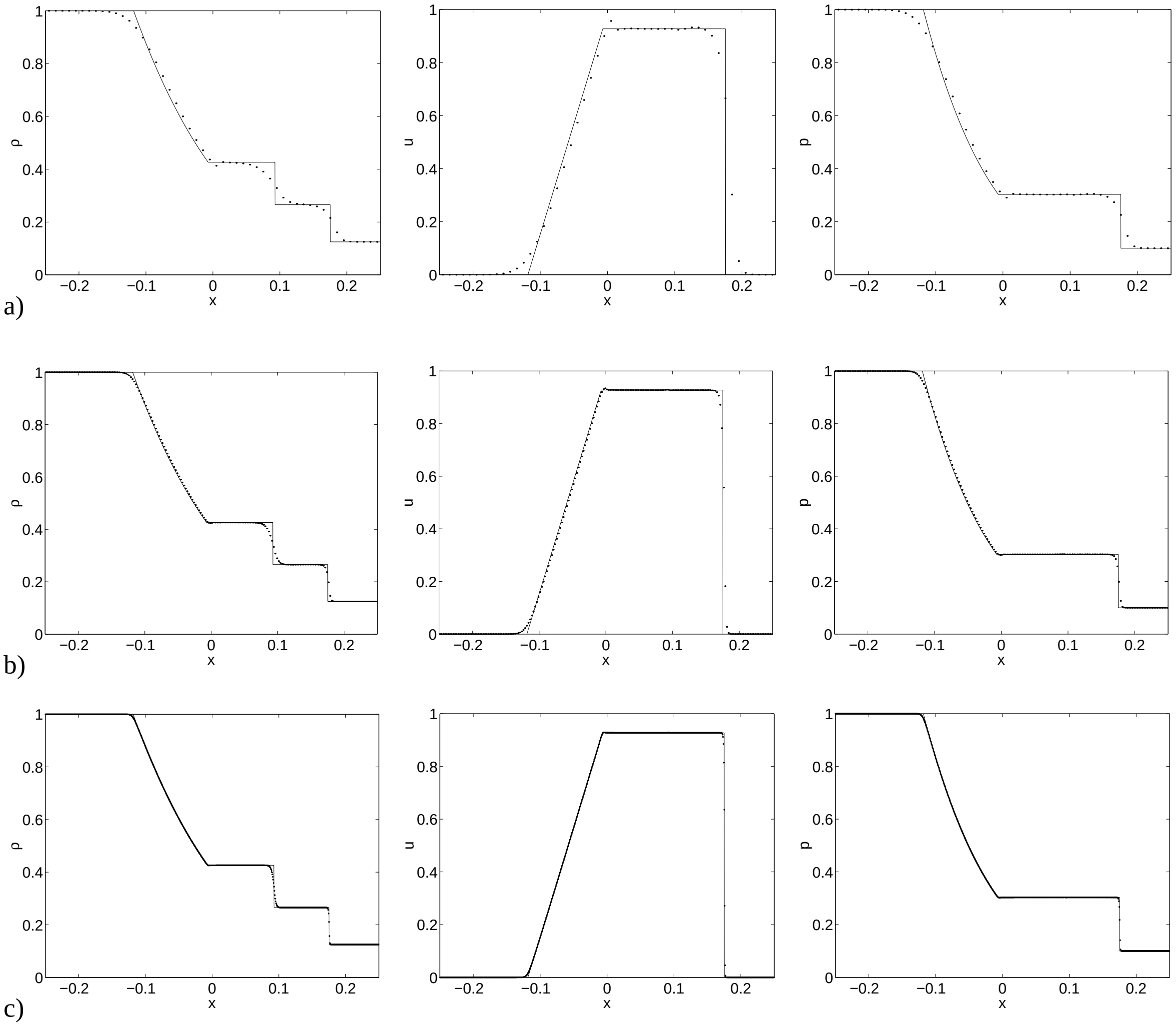


Figure 8.2: Sod problem, solution on uniform grids with 50 (a), 200 (b) and 800 (c) cells. Minmod limiter. Domain $x = [-0.25, 0.25]$, $t = 0.1$, $\Delta x/\Delta t = 8.0$.

### *8.1.2 Forward-facing step*

This problem was calculated with second-order accuracy almost without changes. Only the entropy / enthalpy correction above the step, as described in section 5.1 causes problems: the solution becomes unstable there. This problem is caused by the extrapolations used at the boundary: the gradients at the corner of the step are very high, so extrapolation may cause unphysical states at the boundary. Without the correction, the numerical viscosity in the scheme keeps the solution smooth, so no problems appear. But when the states in some cells at the corner are reset every step, then the solution becomes less smooth and the extrapolation causes unphysical states. This is prevented by moving the pattern of cells to be reset (figure 5.2) two cells in the positive $x$-direction. Thus, the singularity at the step gets two cells to develop and then the entropy is reset in an area with much lower gradients. With this change, all tests were calculated without problems. $\Delta x/\Delta t$ was doubled, from 6.25 to 12.5.

All cases were calculated with the Minmod limiter, because the Van Albada limiter causes instability. This confirms the prediction that the numerical viscosity from the limiter stabilises the scheme. The estimated CFL number is above 0.25, therefore the scheme is not stable without added viscosity. Apparently, the higher numerical viscosity of the Minmod limiter is enough to stabilise the scheme.

Results (figure 8.3 and 8.4) are far superior to the first-order results from section 5.1. The Mach stem is resolved always and is generally in a better position than in the first-order cases, as shown by a comparison with the 'golden' solution from figure 5.3. The oblique shock behind the expansion is resolved on all but the first grids and the finest two grids even show the weak contact discontinuity behind this shock. The second Mach stem, on the step, has almost disappeared. However, there still is a large entropy gradient above the step and a Mach stem with a contact discontinuity is visible above the step on the finest grids, although it is very small. This stem is probably caused by the same mechanisms that formed the first Mach stem and is thus not physically wrong, but it has developed too early.

An interesting phenomenon are the little waves, visible in the entropy graph 8.4. This is a physical feature, known as Kelvin-Helmholtz instability. This type of instability appears when two flows with different tangential velocities are separated by a contact discontinuity. Here, the instability is triggered by the small disturbances coming from the Mach stem. It was also noted by Woodward and Colella [24] in the solutions with their best solver. In the present first-order results (figure 5.5) the instability is damped out.

The shock-like error above the step has not completely disappeared, although it has become smaller. Theoretically, it should disappear, because the states at the cell boundaries are interpolated, so the sonic points of the solutions do not necessarily lie on the cell boundaries. However, the sonic line above the step is nearly vertical (figure 8.4a), so maybe the sonic points still lie on the boundaries, causing the same errors in the Osher scheme as for the first-order cases.

Wiggles behind the shock are also visible here, both behind the normal shock of the Mach stem and behind the lowest part of the bow shock. Their effect is not large, but they cause disturbances in the density and entropy distributions.

The CPU times for these solutions, given in table 8.2, are generally about 5.5 times higher than those for the first-order solutions on the same grid (table 5.3). Also, changing from first- to second-order on the same grid gives an increase in computational costs that is comparable with doing the first-order calculation on a two times finer grid (which requires 8 times more computation). But, as figures 5.4 and 8.3 show, the second-order results are still more accurate than these first-order solutions on finer grids. Therefore, the use of the second-order accurate discretisation on uniform grids is entirely justified, it gives better results for the same computational effort.

| $\Delta x$ | 'Level' | Cells | CPU time (s) |
|---|---|---|---|
| 1/20 | 0 | 1008 | 74 |
| 1/40 | 1 | 4032 | 576 |
| 1/80 | 2 | 16128 | 4663 |
| 1/160 | 3 | 64512 | 43184 |

Table 8.2: Forward-facing step, CPU times for the solution on uniform grids (figure 8.3). The 'level' indicated is the level of the grid cells, if they are created by splitting a $\Delta x = 1/20$ basic grid.

## 8.2. Gradient $\rho$ refinement criterion

The simple $C_{\partial\rho}$ refinement criterion was introduced in section 4.6. It can be used for second-order problems without changes. Results of the two test problems from the previous section are given here. To determine the influence of the limiter function for refined problems, the Sod problem is calculated with two different limiters.

*Sod problem* Solutions of the Sod problem with the modified Van Albada limiter are given in figures 8.5 and 8.6. Figure 8.5 shows the cell level distribution for three values of $d_s$ and maximum levels 1 – 4. The three waves show different behaviour: $C_{\partial\rho}$ in the expansion depends mostly on the physical solution, so the level distributions are almost identical to those from the first-order accurate solutions (figure 5.8). The highest level in the expansion is 1 for $d_s$ = 8, 2 for $d_s$ = 4 and probably 4 for $d_s$ = 1. There is only one exception, the head

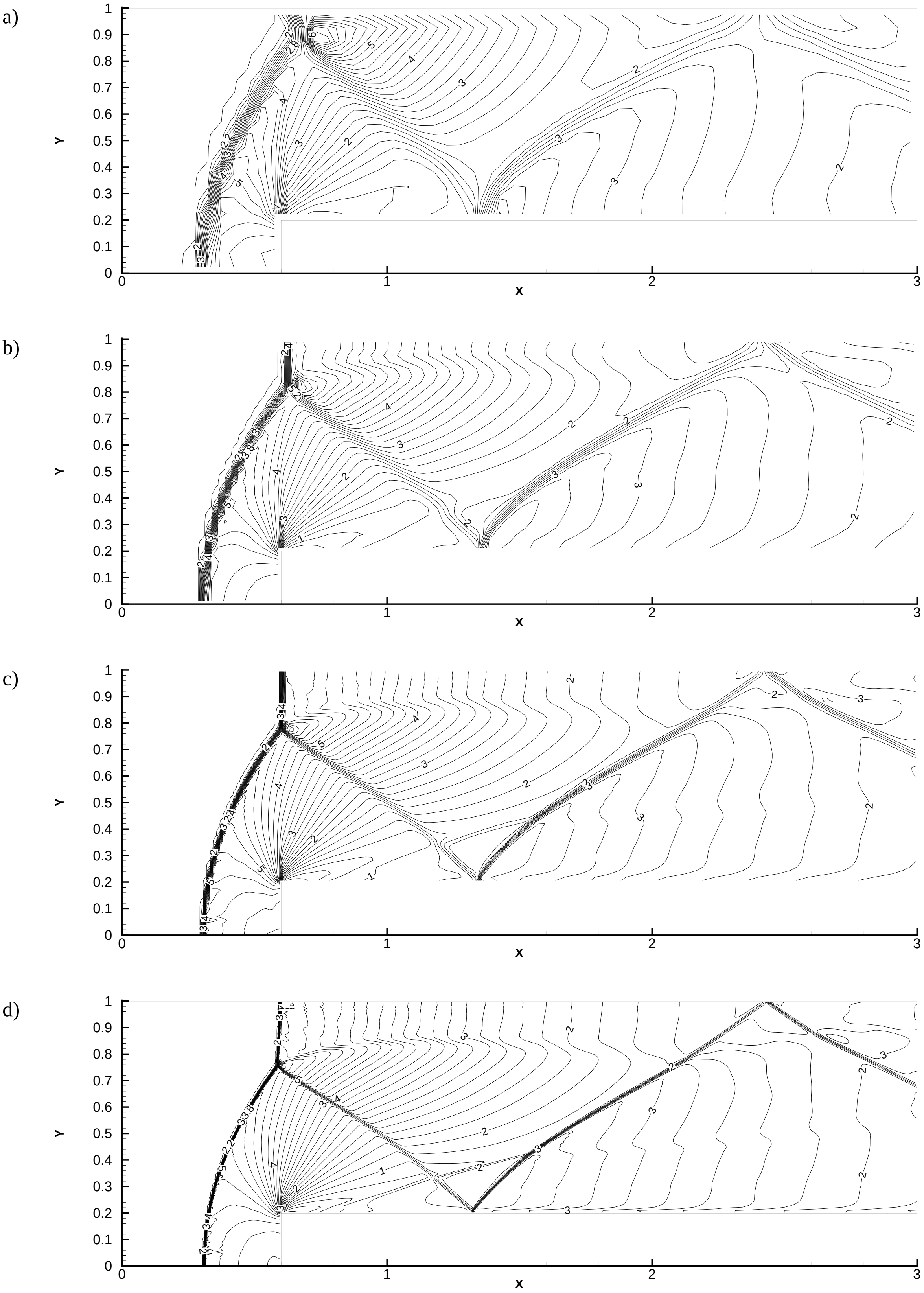


Figure 8.3: Forward-facing step, solution on uniform grids, Minmod limiter. Density $\rho$ at $t = 4.0$. Iso-lines, step 0.2 between the levels. $\Delta x = 1/20$ (a), $1/40$ (b), $1/80$ (c) and $1/160$ (d), $\Delta x/\Delta t = 12.5$.

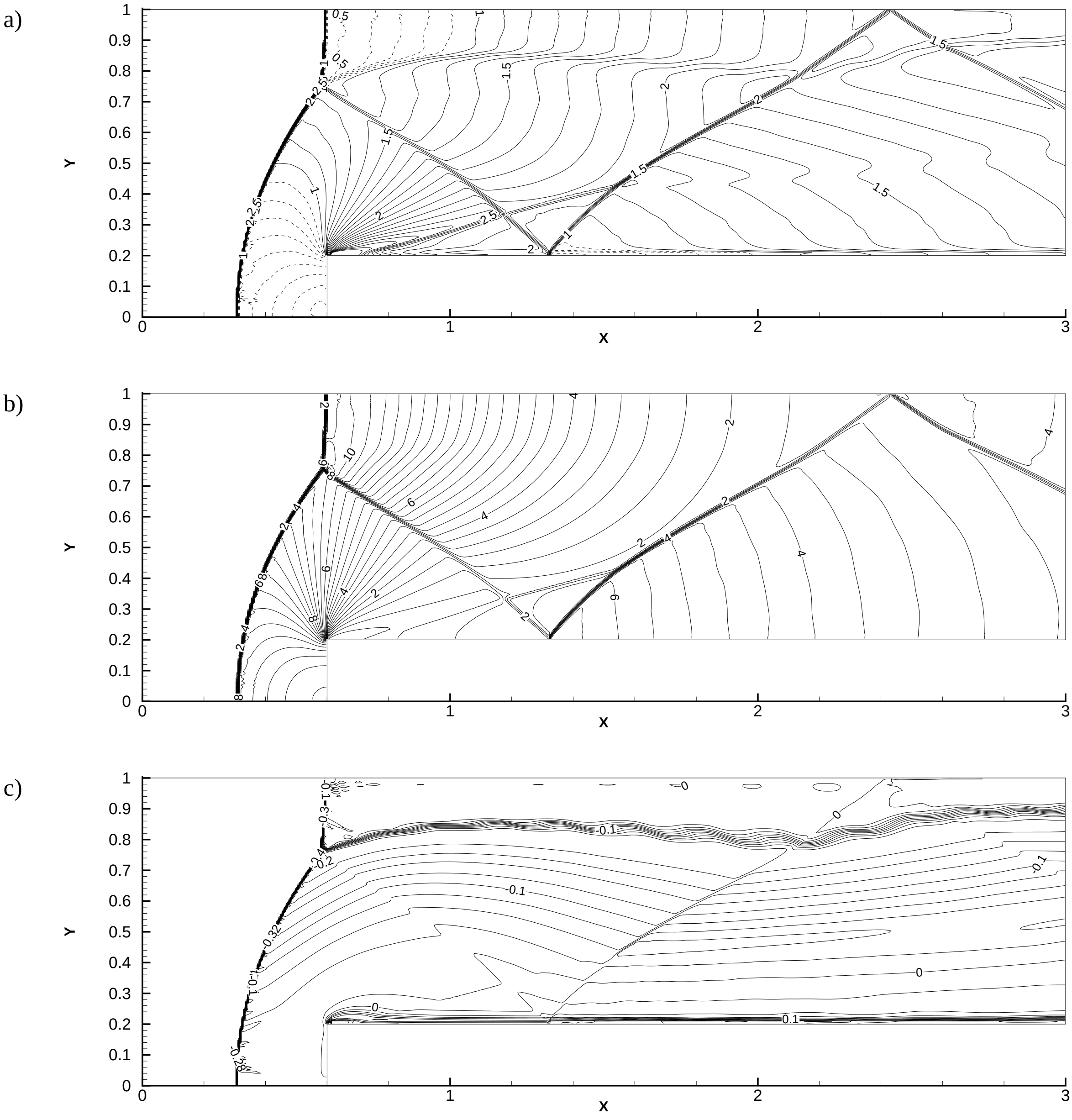


Figure 8.4: Forward-facing step, solution on uniform grid $\Delta x = 1/160$, Minmod limiter. Iso-lines for Mach number $\frac{\sqrt{u^2+v^2}}{c}$, 0.1 between the levels (a), pressure $p$, 0.4 between the levels (b) and unscaled entropy $z$, 0.02 between the levels (c).

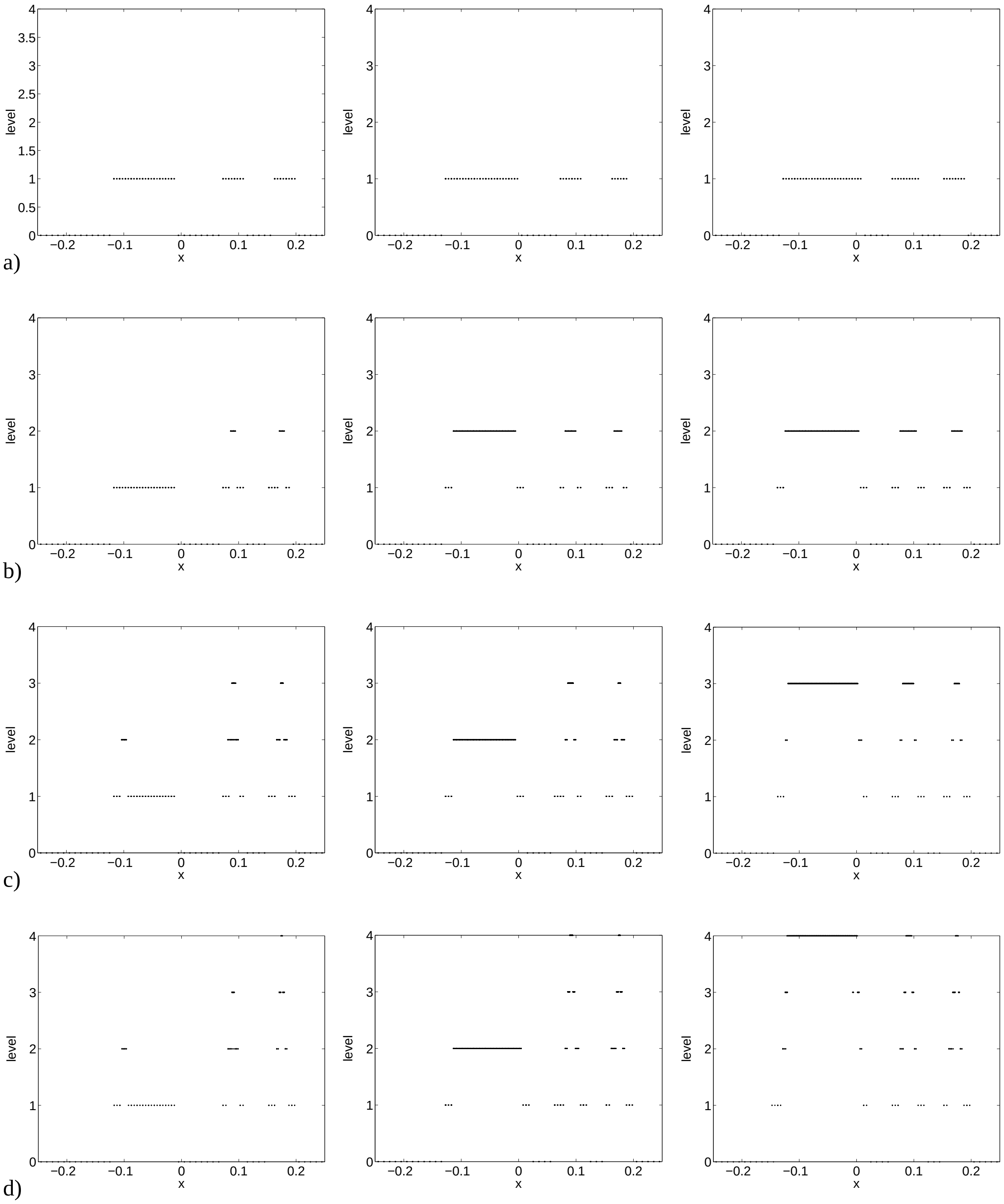


Figure 8.5: Sod problem, $C_{\partial\rho}$ refinement criterion. Cell level distributions with different settings of $d_s$: 8.0, 4.0 and 1.0. $d_m = 0.4d_s$. Modified Van Albada limiter. Maximum level 1 (a), 2 (b), 3 (c) and 4 (d).

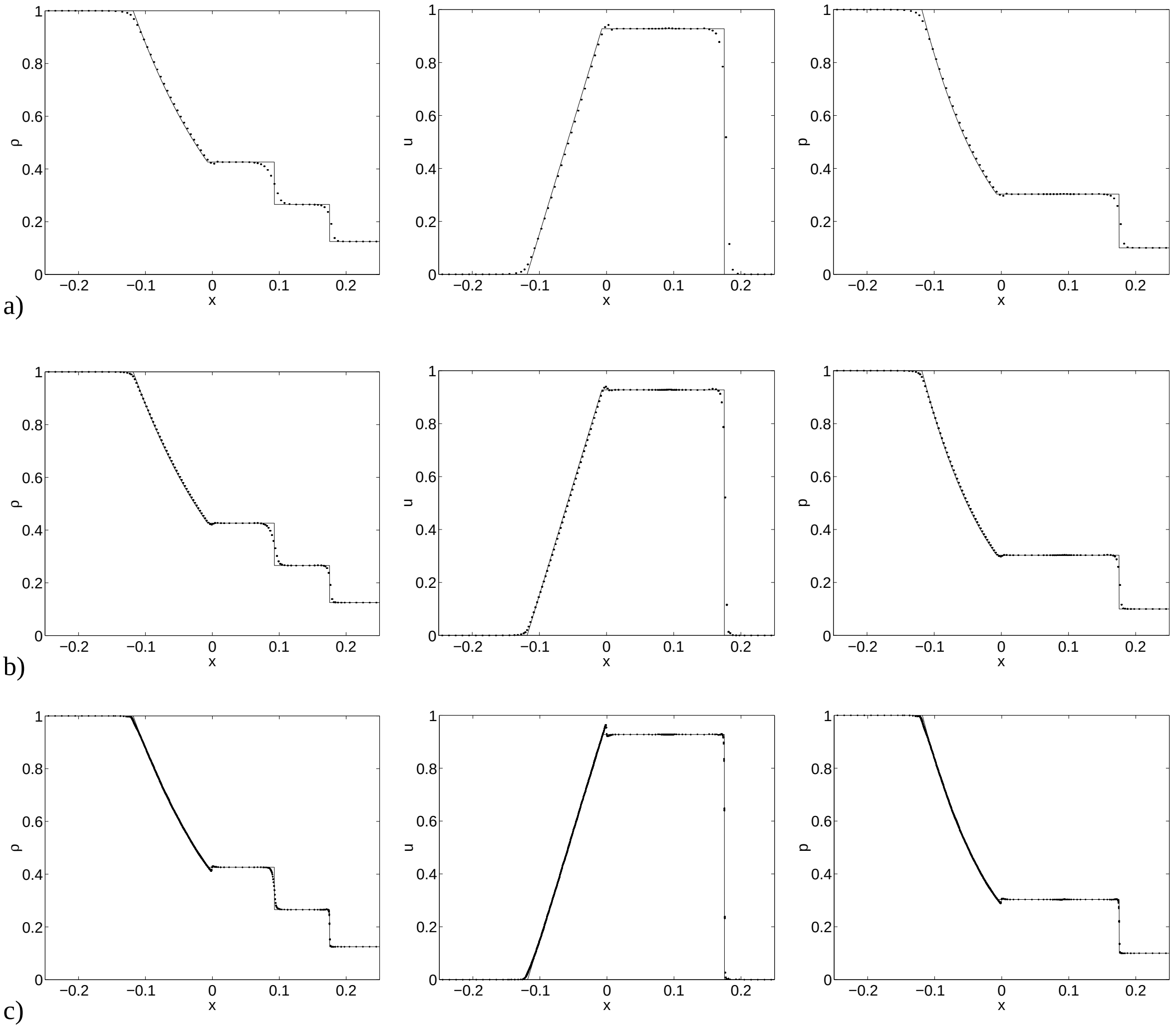


Figure 8.6: Sod problem, $C_{\partial\rho}$ refinement criterion. Solution for maximum levels 1 (a), 2 (b) and 4 (c). $d_s$ = 1.0, $d_m$ = 0.4. Modified Van Albada limiter. Domain $x = [-0.25, 0.25]$, $t = 0.1$, $\Delta x/\Delta t = 8.0$.

of the expansion has $l$ = 2 for $d_s$ = 8. The density curve is rather steep at the head of the expansion, but it is smeared in the first-order accurate solutions, so these have $l$ = 1 in the entire expansion for $d_s$ = 8.

The shock waves are always resolved on the highest level. This is expected, because the shock is self-steepening: when the grid around the shock is refined, the shock becomes sharper, so the grid is refined again, etc. This behaviour is, again, the same as for the first-order solutions.

The contact discontinuity is usually solved on a higher grid level than the contact discontinuities in the corresponding first-order problems. The contact discontinuity is not self-steepening, but the second-order discretisation does not smear the contact discontinuity much, so it stays sharp and it is resolved on high levels. In fact, it always has the highest level allowed, except for $M$ = 4 and $d_s$ = 8, there it is resolved on level 3. It is therefore expected that the contact discontinuity will be resolved on lower levels too for $M$ = 1 to 3 if $d_s$ is chosen high enough.

Plots of the solutions (figure 8.6) show excellent agreement with the uniform solutions from figure 8.1. Only one extra error is noted, the relatively high overshoot behind the expansion. This overshoot develops in the first time step, when the shock is still extremely sharp, and it does not dissipate afterwards.

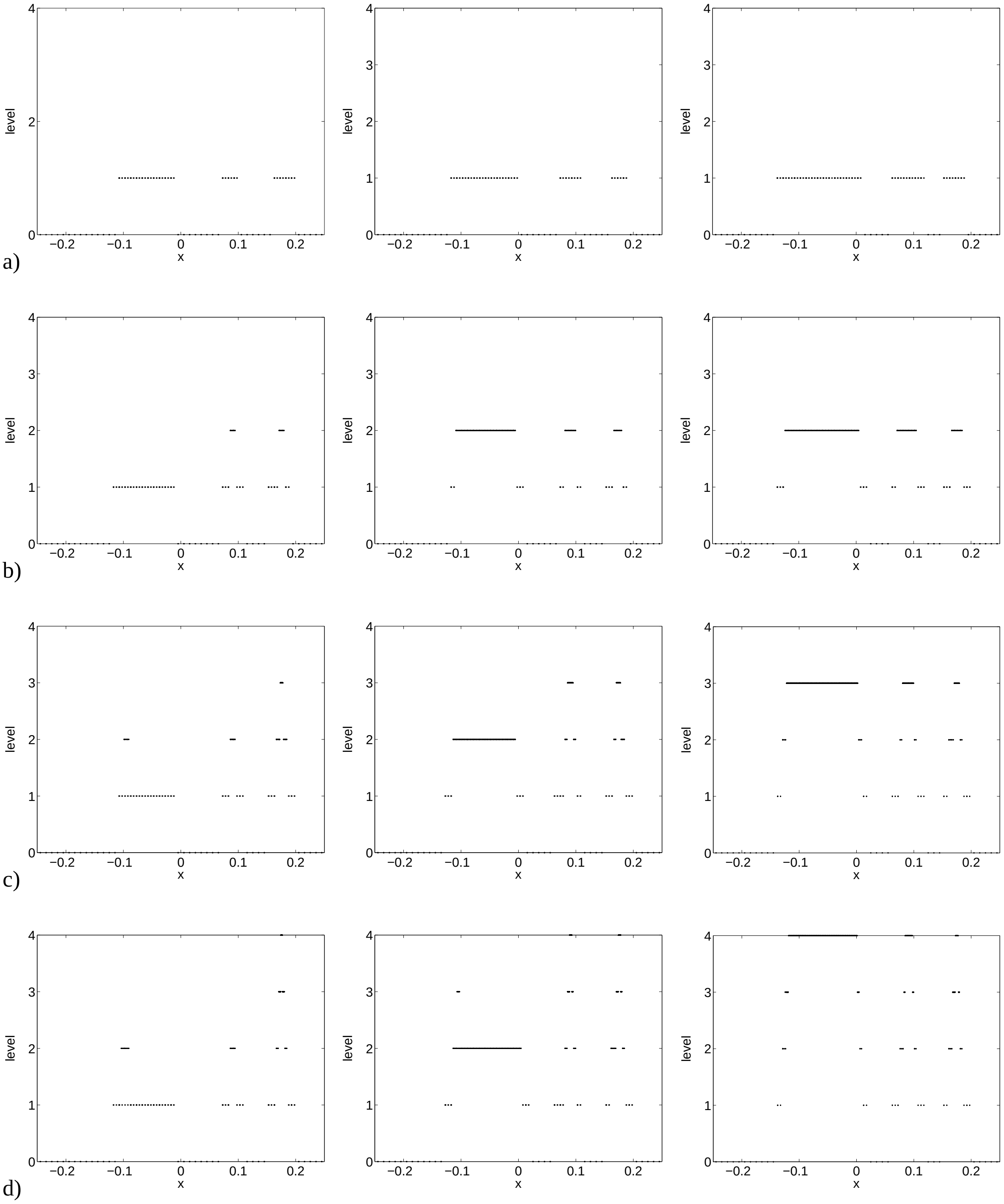


Figure 8.7: Sod problem, $C_{\partial\rho}$ refinement criterion. Cell level distributions with different settings of $d_s$: 8.0, 4.0 and 1.0. $d_m = 0.4d_s$. Minmod limiter. Maximum level 1 (a), 2 (b), 3 (c) and 4 (d).

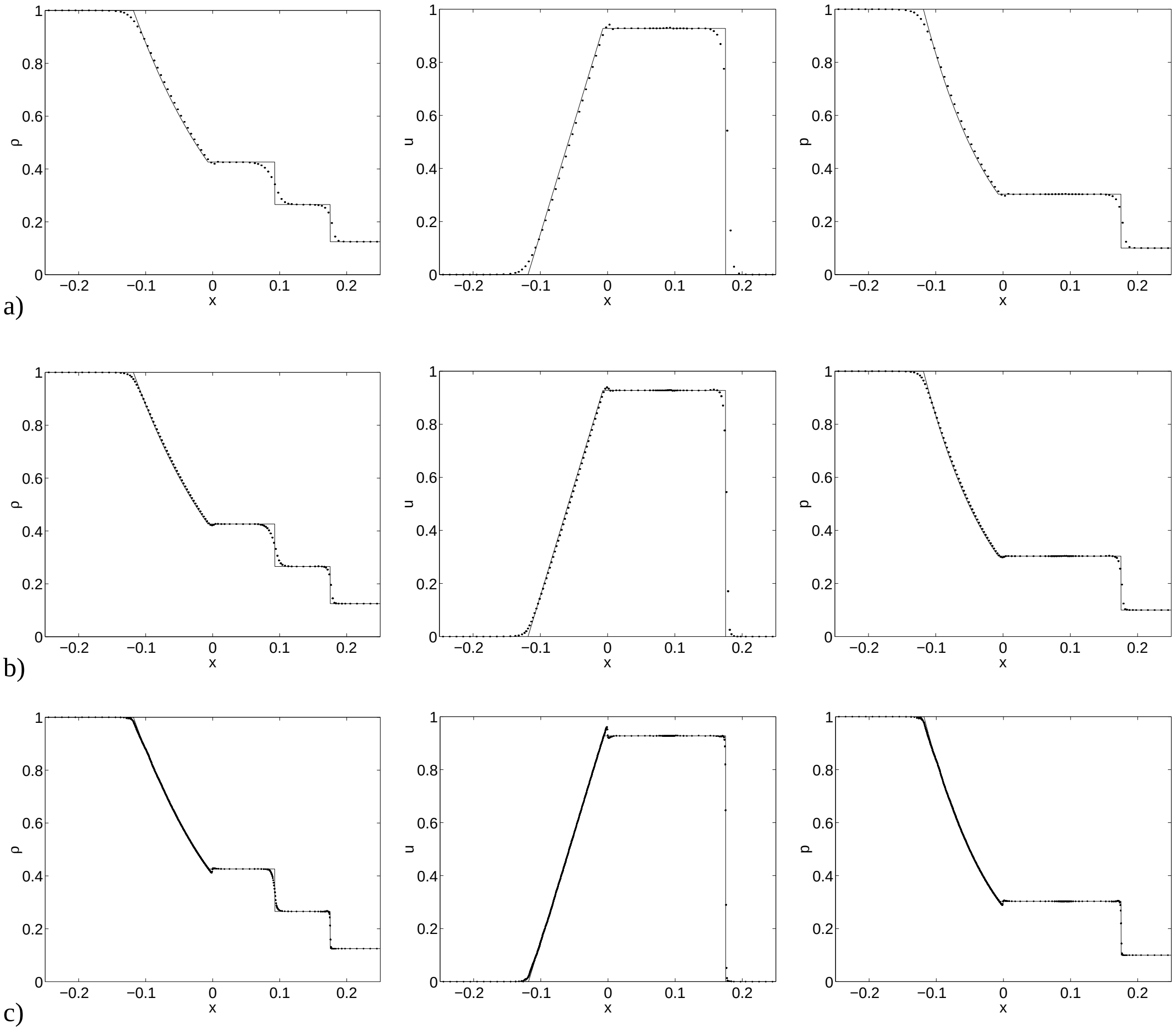


Figure 8.8: Sod problem, $C_{\partial\rho}$ refinement criterion. Solution for maximum levels 1 (a), 2 (b) and 4 (c). $d_s$ = 1.0, $d_m$ = 0.4. Minmod limiter. Domain $x = [-0.25, 0.25]$, $t = 0.1$, $\Delta x/\Delta t = 8.0$.

The solutions with the Minmod limiter (figure 8.7 and 8.8) are almost identical with the modified Van Albada solutions. One difference is noteworthy: for $M = 3$, $d_s = 8$, the contact discontinuity is only resolved on $l = 2$ cells. The higher numerical viscosity of the Minmod limiter has apparently smeared the contact discontinuity more, so now it is resolved on a lower level than with the $\phi_{MVA}$ limiter.

Errors for these solutions are plotted in figures 8.9 and 8.10. We see that, for all cases, the errors are much smaller than the first-order errors from figure 5.10. Also, the results with the modified Van Albada limiter are always more accurate than the results with the Minmod limiter. The errors here do not converge to the uniform values and the difference in the error between different maximum levels of refinement reduces when $M$ becomes larger. Therefore, a part of the error is probably independent of $M$: the overshoot behind the expansion, which does not disappear for large $M$. No attempts have been made to remove this overshoot, but when it is removed, the errors will probably converge to the uniform values.

The errors are almost independent of the settings for the refinement criterion, if $d_s$ is chosen small enough (4 or lower). Only one situation where the error really depends on $d_s$ is visible in the figures: $e_\rho$ decreases with $d_s$ for high values of $d_s$. This is probably caused by the variation in the level of refinement in the contact discontinuity for $d_s$ = 4 to 8. This change is larger for the Minmod limiter, therefore the error with this limiter changes more.

*Forward-facing step problem* Results for the forward-facing step problem are given in figures 8.11, 8.12 and 8.13. The solutions produced for this 2D test problem are excellent: the positions of the shocks (figure 8.11) are the same as the shock positions on the uniform grids (figure 8.3) and the shocks are perfectly sharp. The entropy plot 8.13c shows that even the contact discontinuity is sharp. Plots of the grids (figure 8.12) show that the $C_{\partial\rho}$ criterion detects all flow features: the shocks, the contact discontinuity behind the Mach stem, even the weak oblique shock behind the expansion. The Kelvin-Helmholtz instability in the contact discontinuity is visible in figure 8.13c, although it is damped slightly. The bands of refined cells are narrow and the grid level changes smoothly from 0 to $M$.

Only one flow feature is resolved less accurately: the contact discontinuity behind the weak oblique shock. This flow feature is extremely weak, so it is not detected by the $C_{\partial\rho}$ criterion. Therefore, the contact discontinuity can be seen in the density plots for $M$ = 2 and 3, but it is less pronounced than on the uniform grids.

Adaptive gridding makes the solutions slightly less stable: considerable wiggles can be seen behind the normal shock and the lower part of the bow shock, as well as near the second reflected oblique shock. However, these wiggles do not affect the overall quality of the solution.

A last point of interest is the ‘boundary layer’ above the step. The entropy graph shows that adaptive gridding with the $C_{\partial\rho}$ criterion *reduces* the strength of this error: on the coarser grid between the step and the shock reflection, the strength of the entropy gradient is reduced. But behind the shock reflection on the lower wall, a strange error in the entropy appears. This feature, though less pronounced, is also visible in the density plot 8.13c.

The CPU times for these solutions, table 8.3, are, of course, higher than those for the first-order solutions. But their relation to the uniform CPU times is almost the same. Even here, the computational costs for $M$ = 3 are five times lower than on the corresponding uniform grid.

| Max. level | CPU time (s) | % of unif. |
|---|---|---|
| 1 | 394 | 68% |
| 2 | 2054 | 44% |
| 3 | 8554 | 20% |

Table 8.3: Forward-facing step, CPU times for the solution on refined grids, $C_{\partial\rho}$ refinement criterion (figure 8.11). The last column gives the CPU time, divided by the time for the solution on a uniform grid on the maximum level.

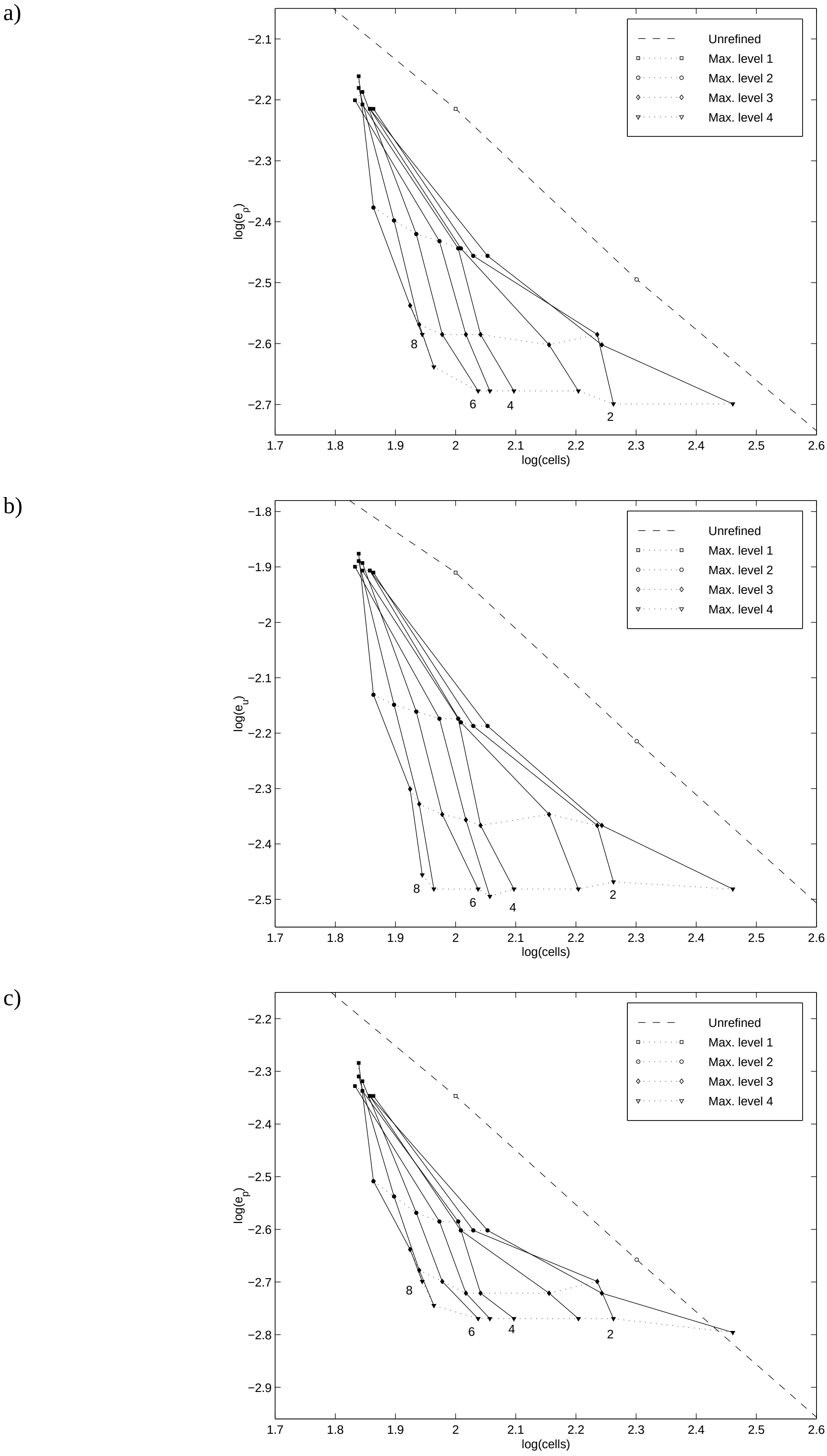


Figure 8.9: Sod problem, $C_{\partial\rho}$ refinement criterion, modified Van Albada limiter. Errors in $\rho$ (a), $u$ (b) and $p$ (c). Dashed line gives the errors for uniform grids, solid lines the errors for $d_s = 1 \ldots 8$ (indicated on the plots).

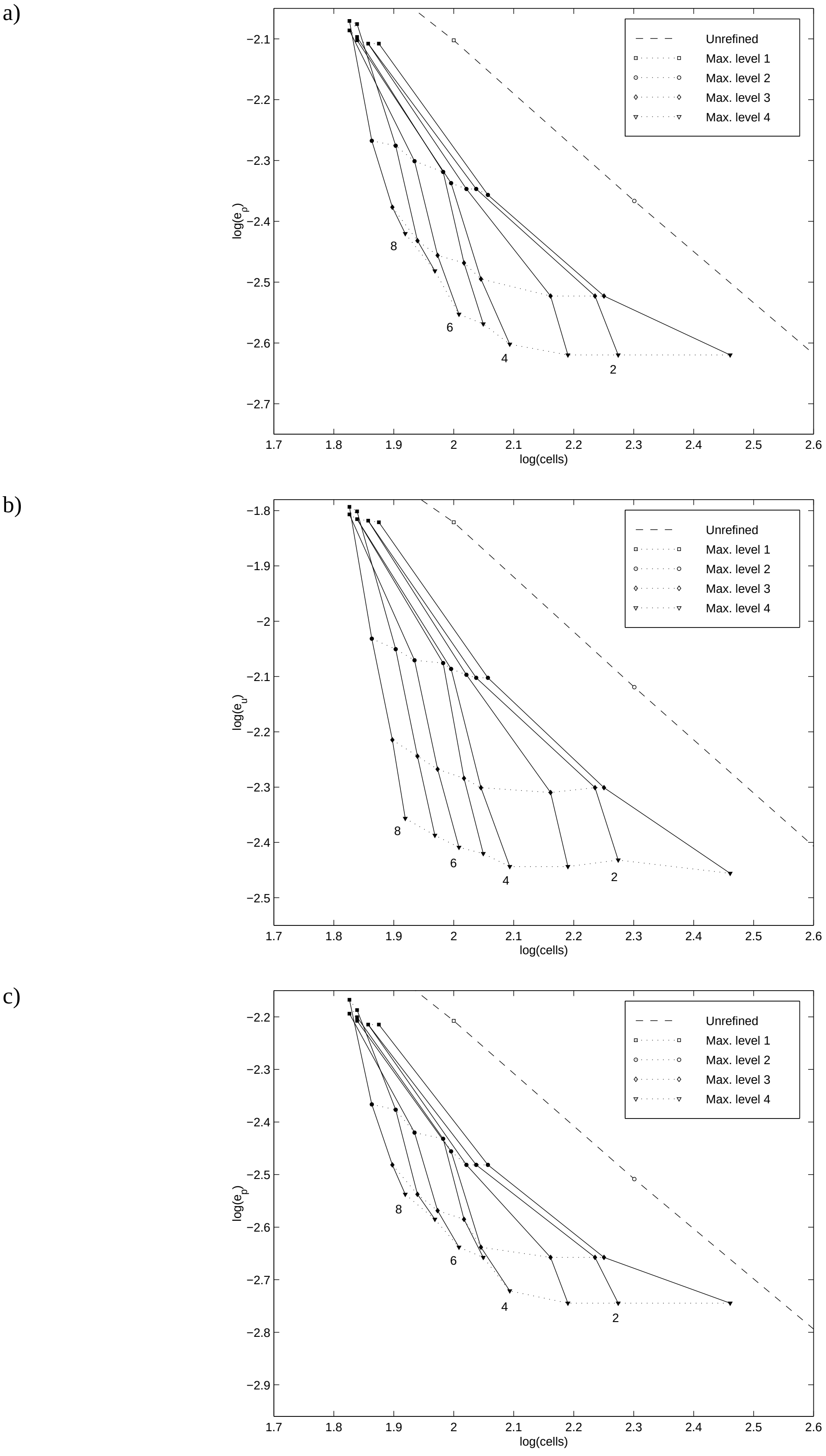


Figure 8.10: Sod problem, $C_{\partial\rho}$ refinement criterion, Minmod limiter. Errors in $\rho$ (a), $u$ (b) and $p$ (c). Dashed line gives the errors for uniform grids, solid lines the errors for $d_s = 1 \ldots 8$ (indicated on the plots).

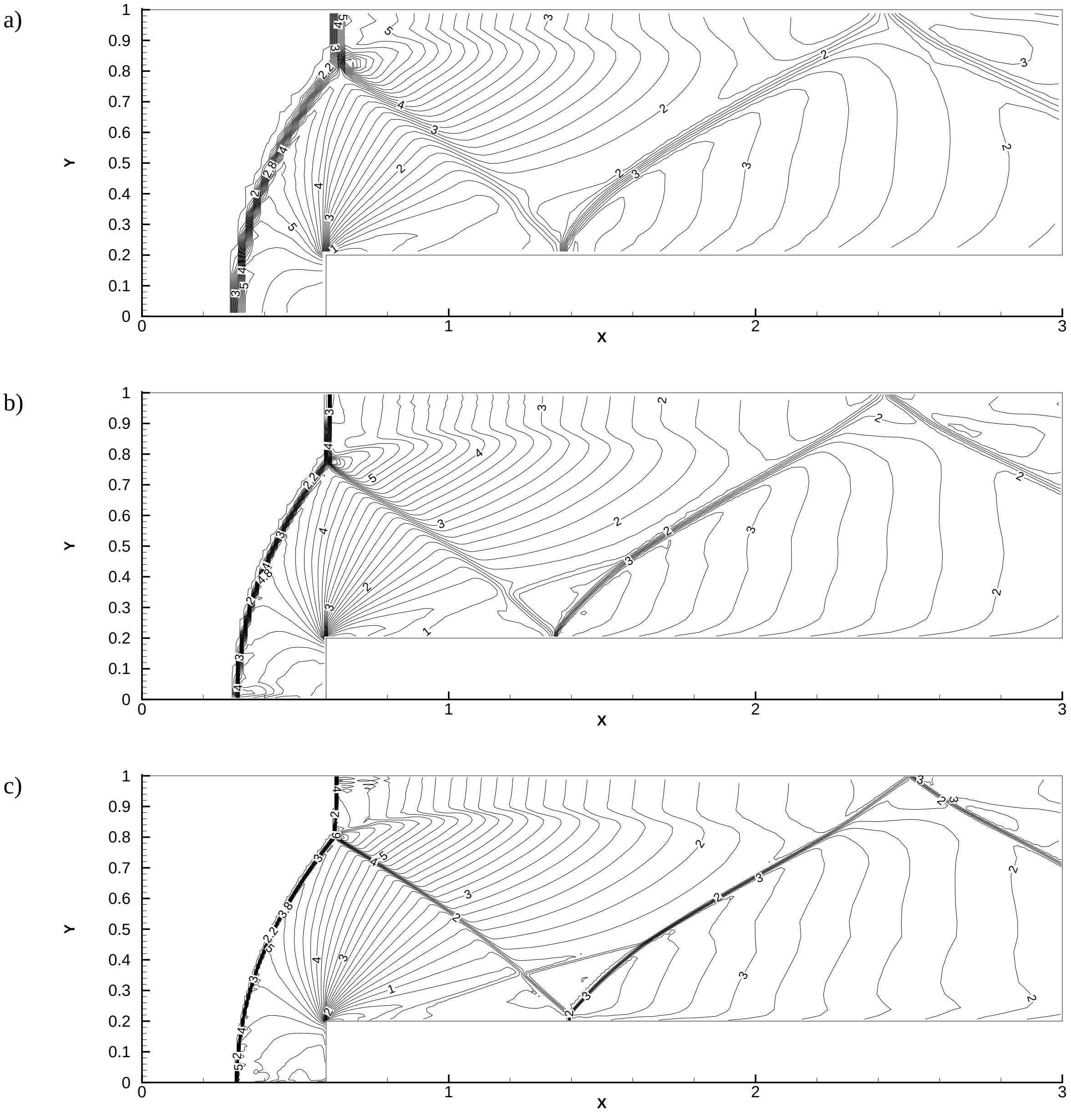


Figure 8.11: Forward-facing step, solution on adapted grids with $C_{\partial\rho}$ refinement criterion, $d_s = 4.0$, $d_m = 1.6$, with Minmod limiter. Density $\rho$ at $t = 4.0$. Iso-lines, step 0.2 between the levels. $\Delta x = 1/20$ for the basic grid, maximum level $M = 1$ (a), 2 (b) and 3 (c).

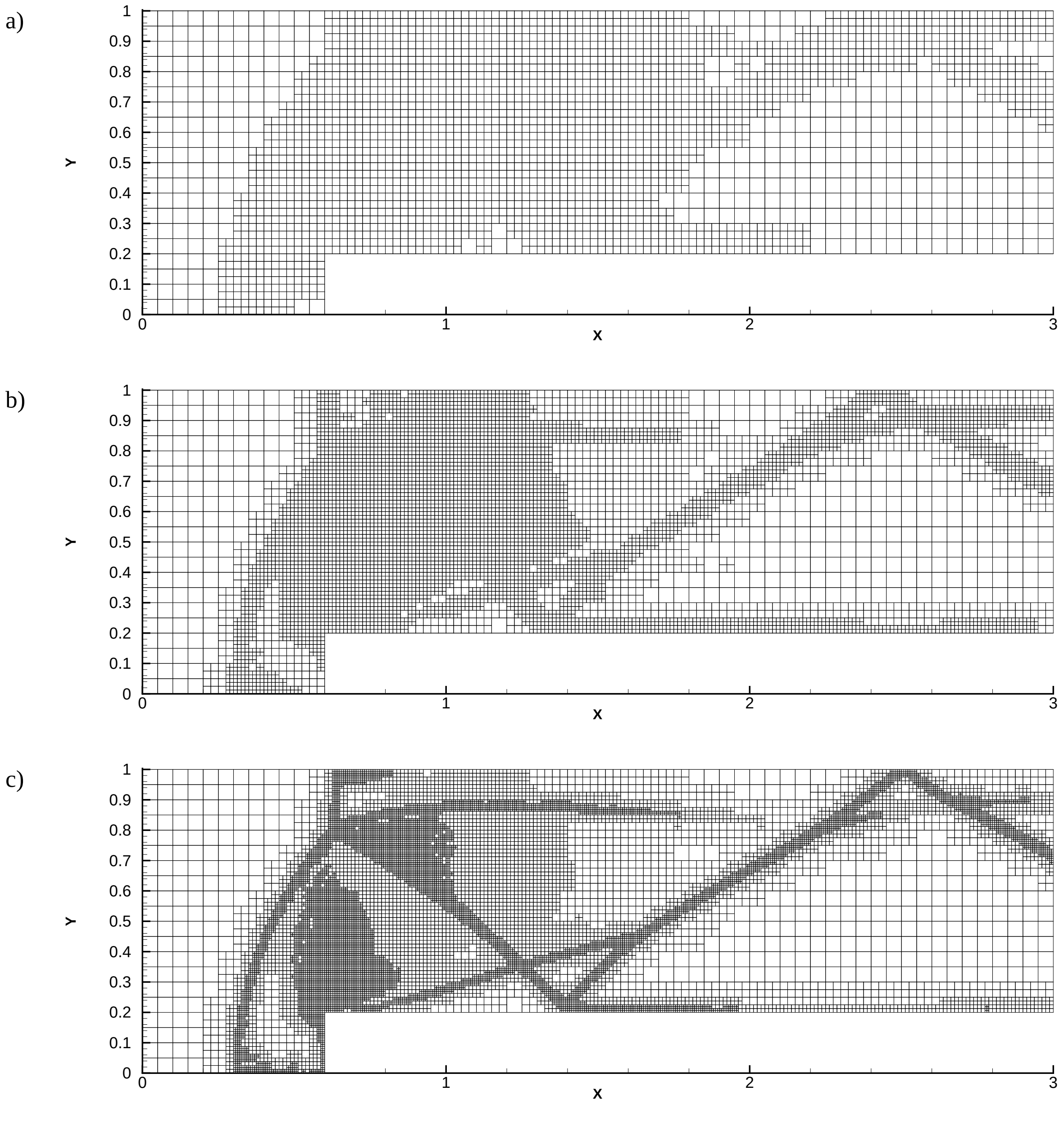


Figure 8.12: Forward-facing step, adapted grids. $C_{\partial\rho}$ refinement criterion, $d_s = 4.0$, $d_m = 1.6$, with Minmod limiter. $\Delta x = 1/20$ for the basic grid, maximum level $M = 1$ (a), 2 (b) and 3 (c).

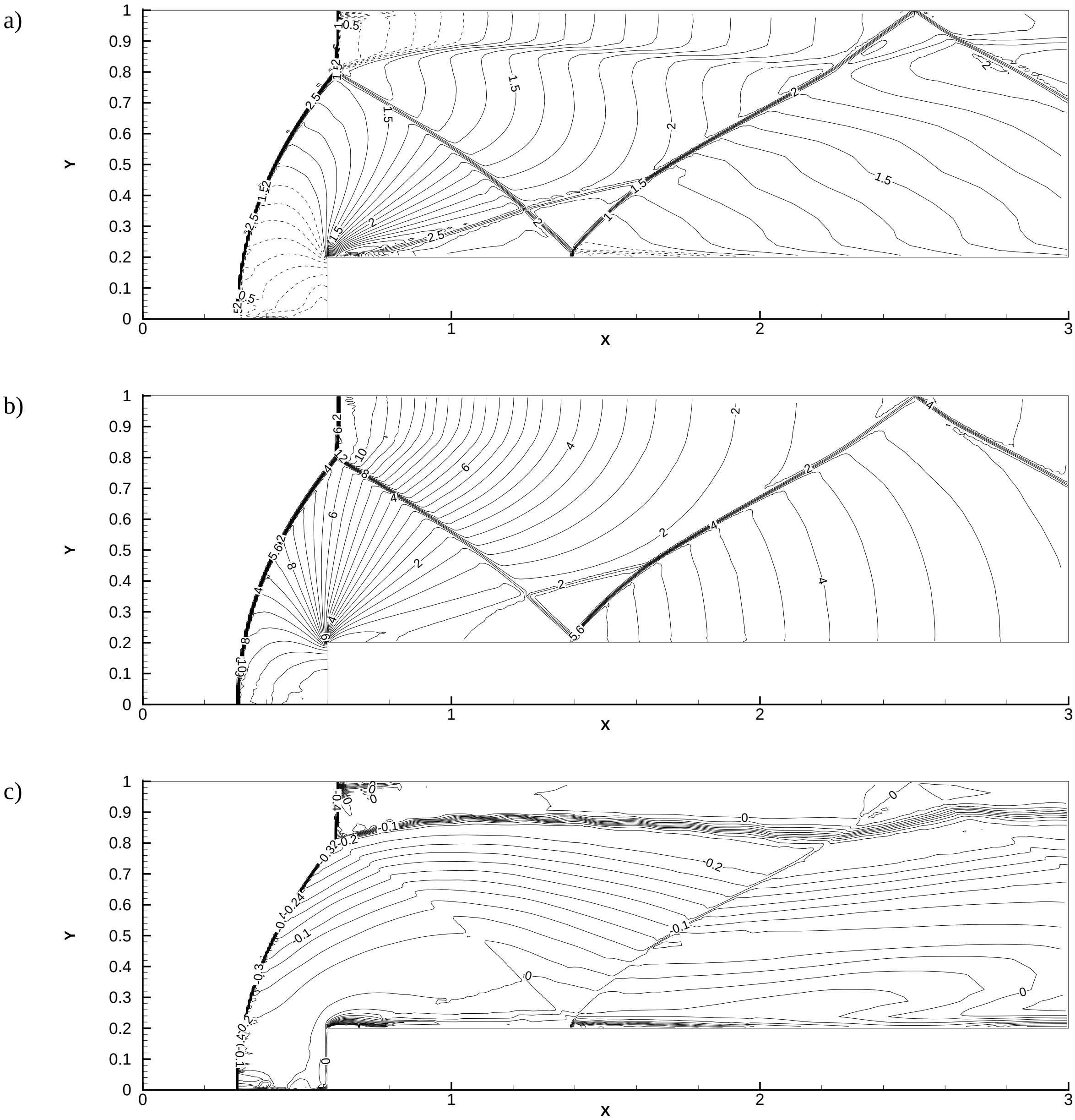


Figure 8.13: Forward-facing step, solution on adapted grids with $C_{\partial\rho}$ refinement criterion, $M = 3$, Minmod limiter. Iso-lines for Mach number $\frac{\sqrt{u^2+v^2}}{c}$, 0.1 between the levels (a), pressure $p$, 0.4 between the levels (b) and unscaled entropy $z$, 0.02 between the levels (c).

## 8.3. Second derivative $\rho$ refinement criterion

Just like the $C_{\partial\rho}$ criterion, the implementation of the $C_{\partial^2\rho}$ criterion does not depend on the order of the solution procedure, so the form (4.10), introduced in section 4.2 can be used for second-order problems without changes. But the reason why it was introduced in section 4.2 is no longer valid, as the error for the second-order scheme is not proportional to the second derivative of the solution. However, the limited solution is still largely determined by numerical viscosity, so the $C_{\partial^2\rho}$ criterion may still give good results.

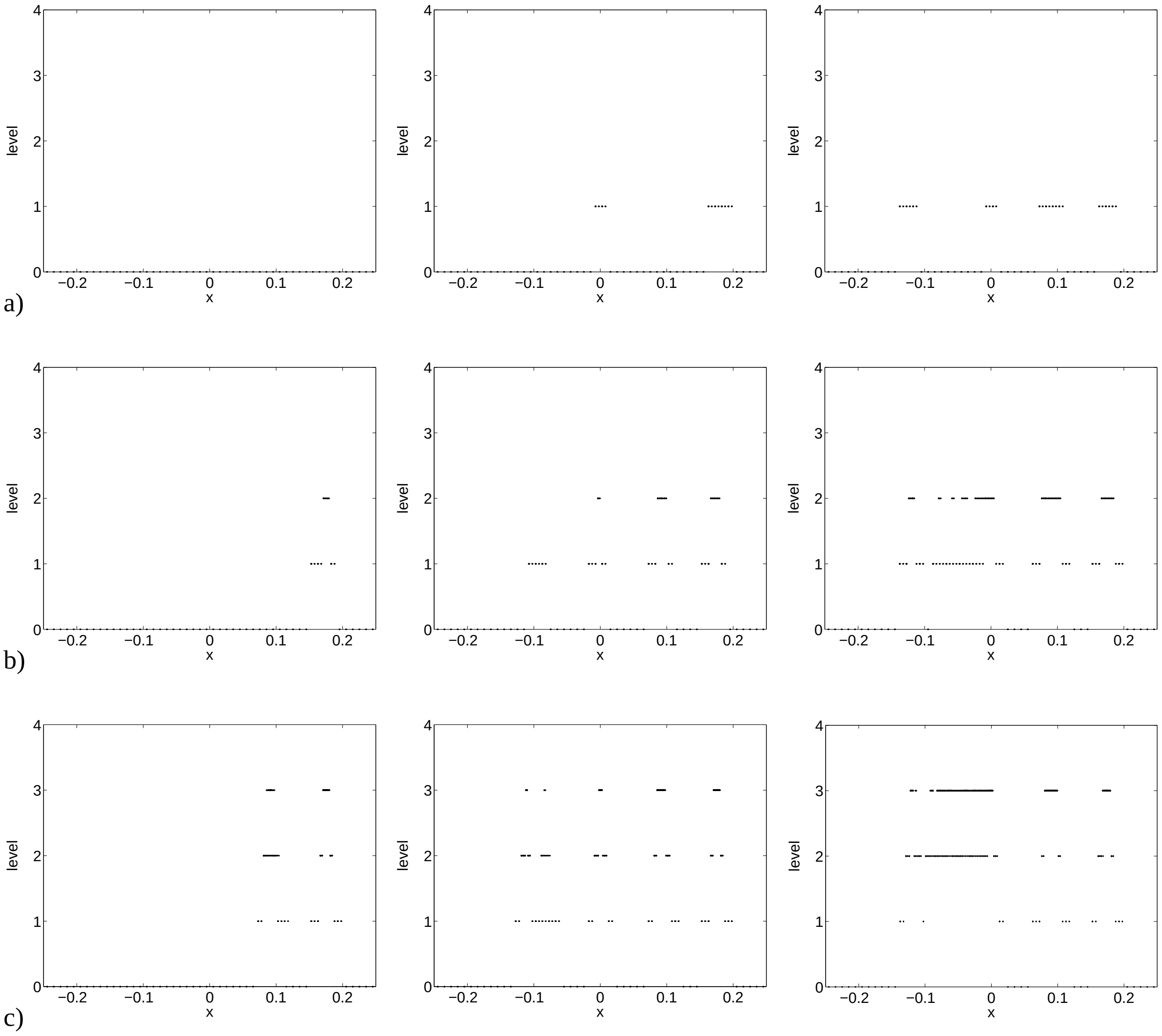


Figure 8.14: Sod problem, $C_{\partial^2\rho}$ refinement criterion. Cell level distributions with different settings of $d_s$: 800, 400 and 100. $d_m = 0.4 d_s$. Modified Van Albada limiter. Maximum level 1 (a), 2 (b) and 3 (c).

*Sod problem* In section 5.3, it was concluded that the $C_{\partial^2\rho}$ criterion does not work for the first-order problems. The criterion proved to be more sensitive to local disturbances in the solution than to the global solution itself. But for second-order problems, these local disturbances are much smaller, so the $C_{\partial^2\rho}$ criterion gives sensible results (see the grids in figure 8.14). We see that the $C_{\partial^2\rho}$ criterion refines on different parts of the solution. For the expansion, usually only the head and and the tail are refined, as these parts have the largest curvature. The ‘body’ of the expansion has a large gradient, but it is nearly linear so it is usually not refined.

The refined regions for the shock and the contact discontinuity are usually broad.

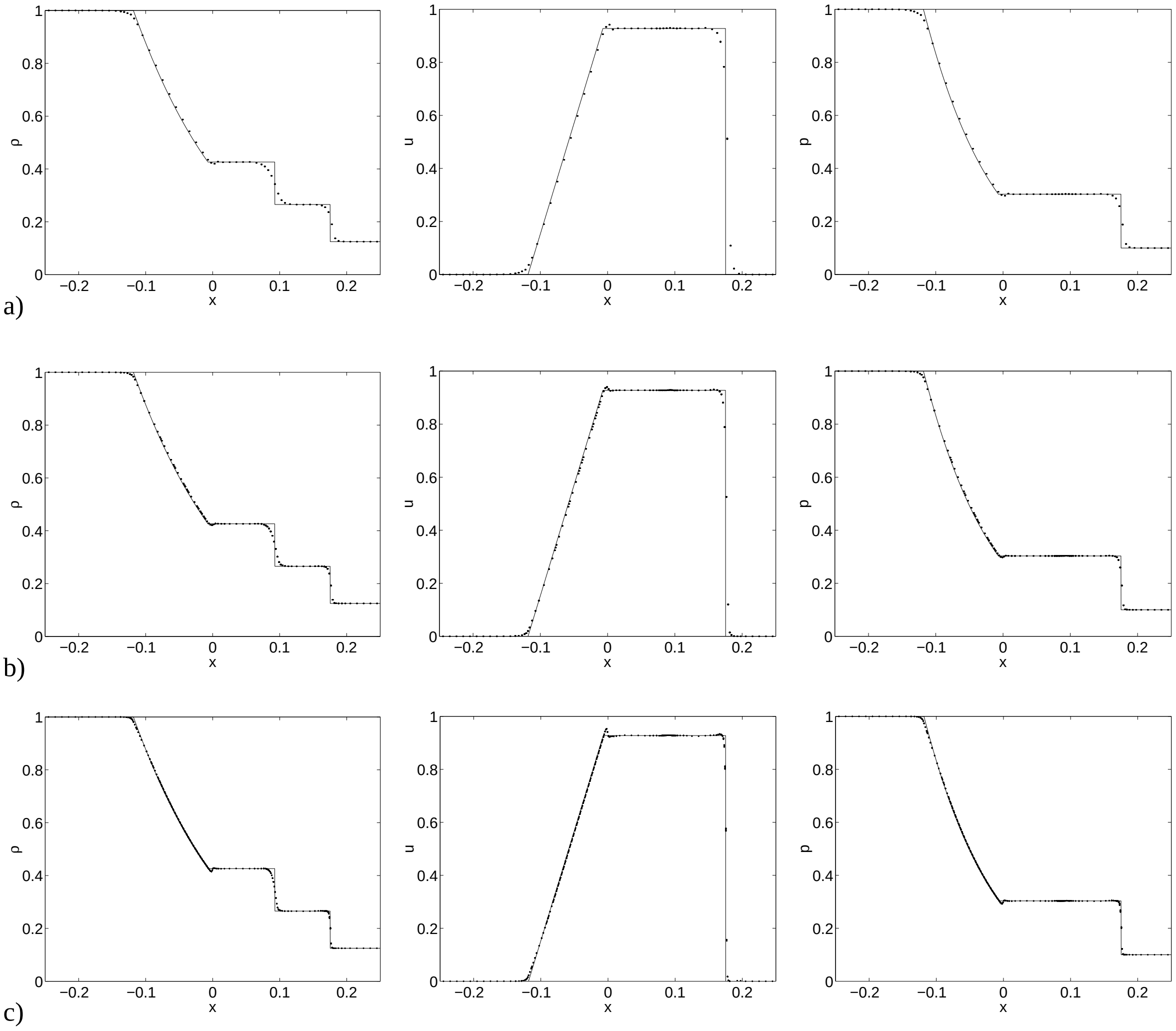


Figure 8.15: Sod problem, $C_{\partial^2\rho}$ refinement criterion. Solution for maximum levels 1 (a), 2 (b) and 3 (c). $d_s$ = 100, $d_m$ = 40. Modified Van Albada limiter. Domain $x = [-0.25, 0.25]$, $t = 0.1$, $\Delta x/\Delta t = 8.0$.

It is obvious that the level distributions are very sensitive to the value of $d_s$. The shock and the contact discontinuity are either refined at the maximum level or not at all. One explanation is that the $C_{\partial^2\rho}$ criterion varies quadratically with the width of a shock or contact discontinuity: when a discontinuity with a constant strength becomes twice narrower, then the gradient in the middle is two times higher and this gradient is reached in half the previous distance, so $\frac{\partial^2\rho}{\partial x^2}$ is four times higher. Therefore a small smearing of a shock causes a large change in $C_{\partial^2\rho}$, causing the level of refinement to drop rapidly.

An unpleasant aspect of the second-derivative $\rho$ criterion is its high sensitivity to small wiggles. In the grids from figure 8.14 it is visible that the criterion reacts to the disturbances caused by the wall boundary conditions on the sides of the domain. For low $d_s$ and high $M$, the cells next to the boundary in the expansion have a different level than the other cells. When this happens, two levels of cells are shown at one $x$-location: the grid is non-uniform in $y$-direction.

Due to this instability of the criterion, it was impossible to obtain solutions for $M$ = 4.

Figure 8.15 shows that these instabilities do not influence the solutions. These graphs show solutions for a low value of $d_s$, so the grids are non-uniform in $y$-direction in the expansion. But this is not visible in the solution. Again, an overshoot is visible behind the expansion, this overshoot disappears when $d_s$ is chosen higher (not shown here). Another error is the small wave behind the shock.

Plots of the errors, figure 8.16, show how well the $C_{\partial^2\rho}$ criterion works for high settings of $d_s$. The errors in this figure are almost the same as the errors with the $C_{\partial\rho}$ criterion, but the number of cells is lower. It is not useful to take $d_s$ very small, this does not decrease the error (for $e_u$, a small $d_s$ even causes a *larger* error). But the number of cells grows for low values of $d_s$, as the amount of 'wiggles' in the solution increases. Therefore, the $C_{\partial^2\rho}$ criterion works best for large values of $d_s$.

*Forward-facing step problem* The results for this problem, figure 8.17, calculated with the $C_{\partial^2\rho}$ criterion look good. The overall quality is comparable with the $C_{\partial\rho}$ solutions. In some places, the solution looks a bit wavy (especially around the weak oblique shock), but in other places the wiggles are reduced (e.g. behind the normal shock). The weak contact discontinuity is resolved more accurately here than with the $C_{\partial\rho}$ criterion. The strong contact discontinuity is smeared a bit when it interacts with the reflected shocks. This is visible in the entropy plot 8.19, the damping of the Kelvin-Helmholtz instability is also visible here.

The flow above the step is resolved here almost exactly like the uniform solution. The numerical boundary layer is not damped out, like in the $C_{\partial\rho}$ solutions and the spurious entropy pattern behind the reflection has also disappeared.

When looking at the grids, figure 8.18, we see the same features that were also noted for the Sod problem. Especially on the finest grid, the bands of cells around the shocks are slightly broader, compared to the $C_{\partial\rho}$ grids. On the other hand, fewer cells are refined in the expansion, causing a reduction of the total number of refined cells, especially on the $M$ = 1 grid. Certain flow features, like the weak oblique shock and the contact discontinuity behind the reflected shocks, are fully refined on the $M$ = 3 grid and not refined at all on the other grids, so again the criterion proves to be very sensitive to changes in the maximum level.

The sensitivity of the criterion to small wiggles is shown in certain areas of the flow, like the 'christmas tree' grid pattern in the expansion. The grid changes from the maximum level to one level lower very often. The same feature is seen in the expansion behind the first reflected shock.

The CPU times from table 8.4 show that the $C_{\partial^2\rho}$ criterion is faster than the $C_{\partial\rho}$ criterion for maximum levels of 1 and 2, because it uses fewer cells there. But when the maximum level is increased to 3, then so many more flow features are resolved on refined grids that the calculation becomes slower than for the $C_{\partial\rho}$ criterion.

| Max. level | CPU time (s) | % of unif. |
|---|---:|---:|
| 1 | 260 | 45% |
| 2 | 1348 | 29% |
| 3 | 8981 | 21% |

Table 8.4: Forward-facing step, CPU times for the solution on refined grids, $C_{\partial^2\rho}$ refinement criterion (figure 8.17). The last column gives the CPU time, divided by the time for the solution on a uniform grid on the maximum level.

## 8.4. Error estimate refinement criterion

The theoretical basis for this refinement criterion is lost when the truncation error of a discretisation is of $\mathcal{O}(h^2)$. In that case, the error made in one step is $h^3 a_i^k$ and the equivalence of (4.21) reads

$$\boldsymbol{q}_{i,j}^{k+2} - R_{l+1}^{l}\, \boldsymbol{q}_{i\pm1/2,j\pm1/2}^{k+2} = 2ch^3 a_{i,j}^k + \mathcal{O}\left(h^3\right). \tag{8.1}$$

The leading-order term depends no longer only on the truncation error.

Still, some calculations were made on the Sod problem. $C_{EE}$ is first determined for solutions on uniform grids,

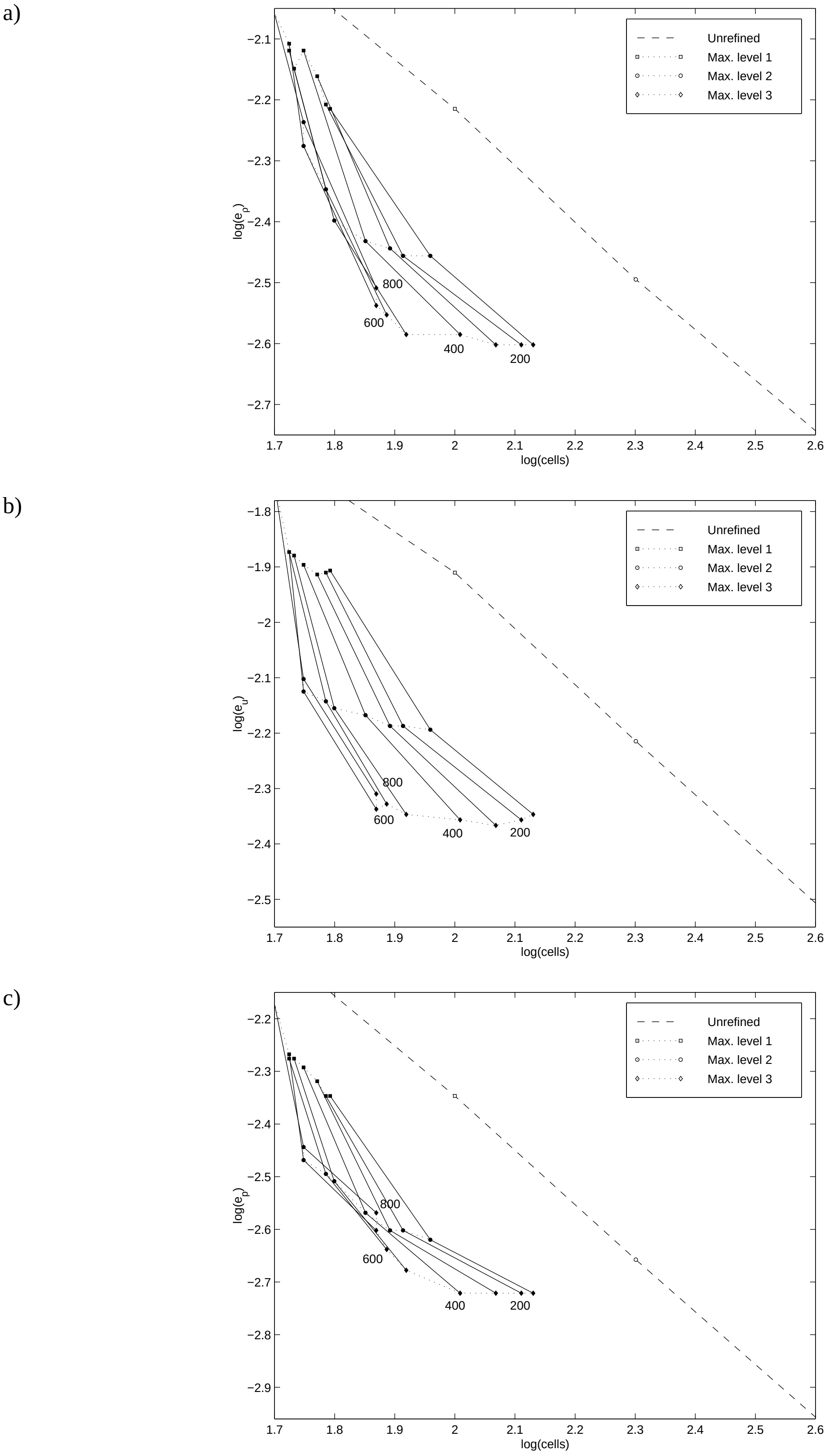


Figure 8.16: Sod problem, $C_{\partial^2 \rho}$ refinement criterion, modified Van Albada limiter. Errors in $\rho$ (a), $u$ (b) and $p$ (c). Dashed line gives the errors for uniform grids, solid lines the errors for $d_s = 100 \ldots 800$ (indicated on the plots).

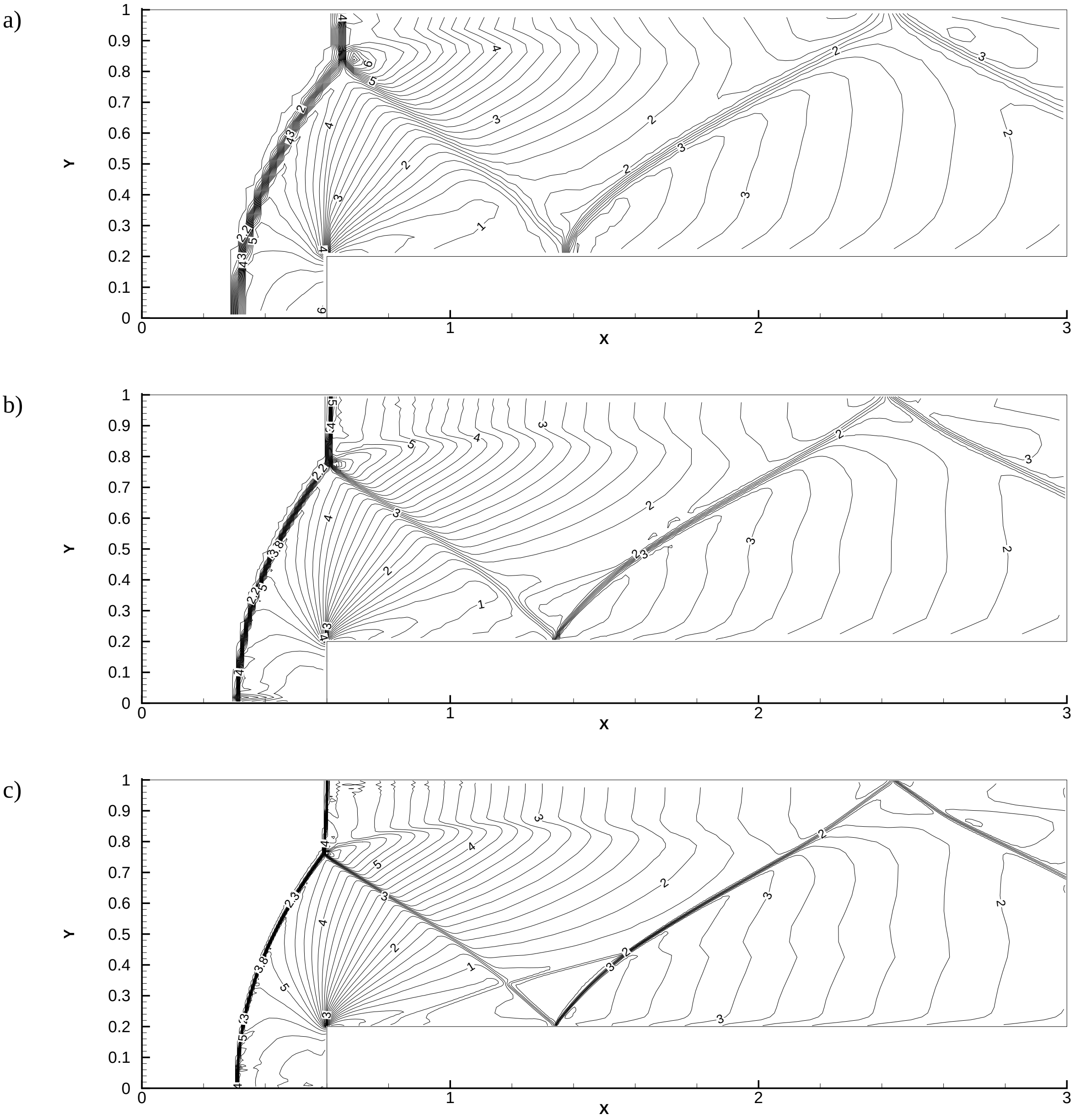


Figure 8.17: Forward-facing step, solution on adapted grids with $C_{\partial^2\rho}$ refinement criterion, $d_s = 100$, $d_m = 40$, with Minmod limiter. Density $\rho$ at $t = 4.0$. Iso-lines, step 0.2 between the levels. $\Delta x = 1/20$ for the basic grid, maximum level $M = 1$ (a), 2 (b) and 3 (c).

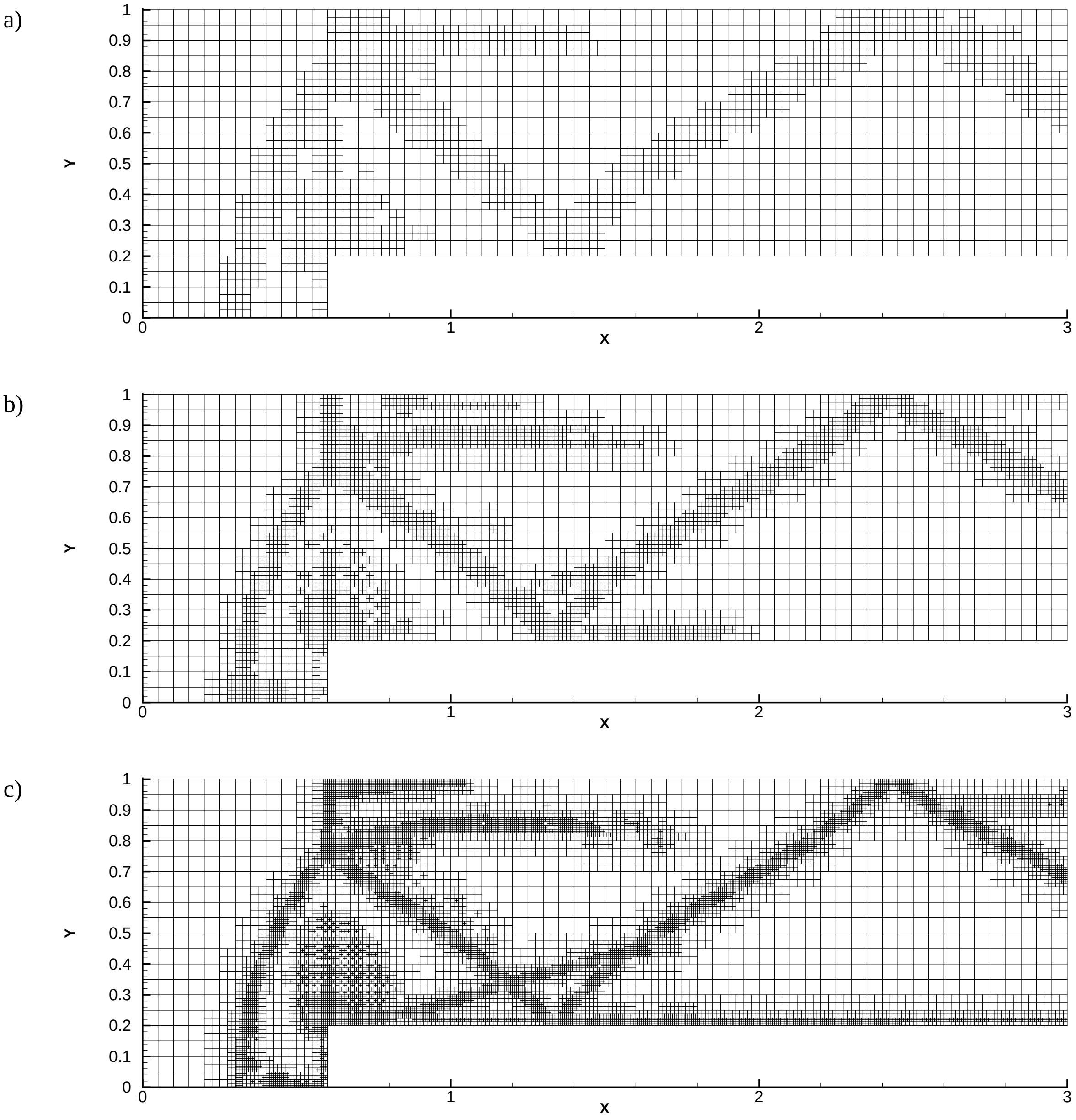


Figure 8.18: Forward-facing step, adapted grids. $C_{\partial^2\rho}$ refinement criterion, $d_s = 100$, $d_m = 40$, with Minmod limiter. $\Delta x = 1/20$ for the basic grid, maximum level $M = 1$ (a), 2 (b) and 3 (c).

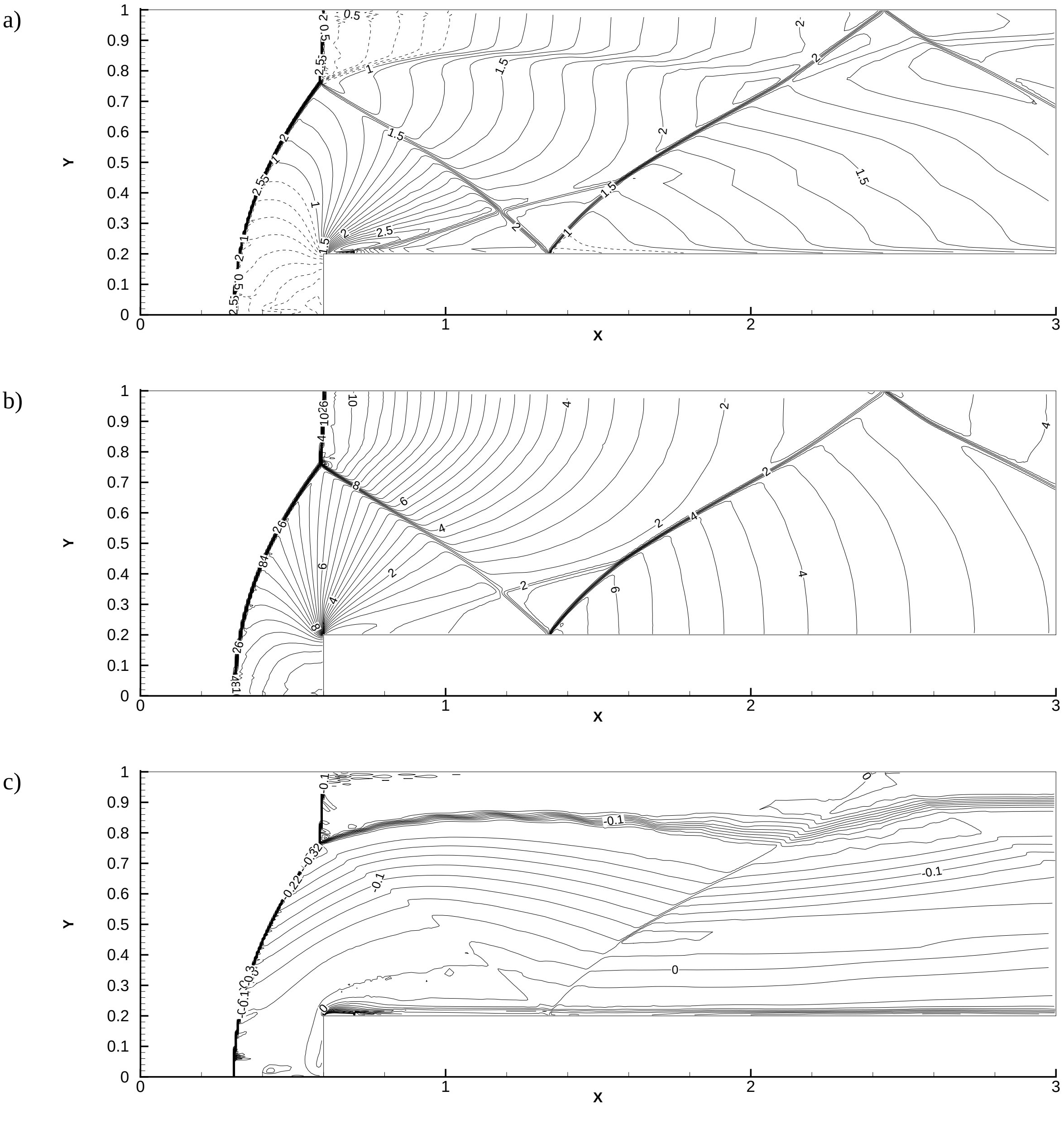


Figure 8.19: Forward-facing step, solution on adapted grids with $C_{\partial^2\rho}$ refinement criterion, $M = 3$, Minmod limiter. Iso-lines for Mach number $\frac{\sqrt{u^2+v^2}}{c}$, 0.1 between the levels (a), pressure $p$, 0.4 between the levels (b) and unscaled entropy $z$, 0.02 between the levels (c).

figure 8.20. We see that, in some areas, $C_{EE}$ decreases rapidly when the number of cells is increased. But, in general, $C_{EE}$ is much more nervous than in the first-order case (figure 5.17).

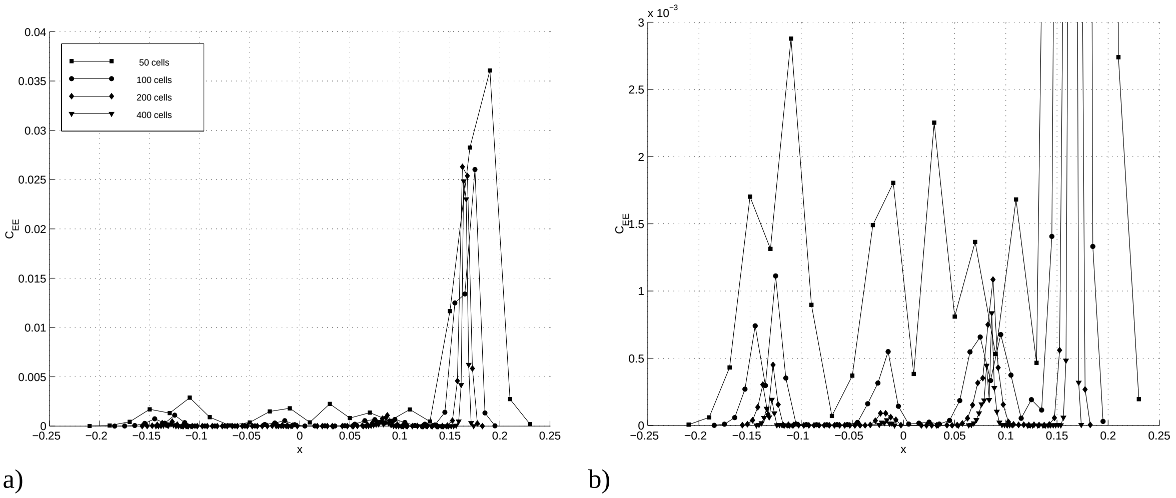


Figure 8.20: Sod problem, the value of $C_{EE}$ on uniform grids.

Plots of the grids (figure 8.21) show how nervous the refinement criterion is: large areas are refined and many ‘wiggles’ in the level distribution occur. Again, the solutions for $M = 4$ were unstable. Graphs of the solutions (figure 8.22) look well, except for the high overshoot behind the expansion, but the error plots 8.23 show how bad the criterion is. There is no clear relationship between $d_s$ and either the error or the number of cells and the errors are very high. It is obvious that the $C_{EE}$ criterion does not work for second-order problems.

## 8.5. Conclusion

In the previous sections, numerical results are obtained with the second-order accurate scheme, with two different limiters and three different refinement criteria. The second-order results are better than the first-order results from chapter 5, a better accuracy is achieved with the same computational costs in all cases.

The two limiters add different amounts of numerical viscosity to the scheme. The modified Van Albada limiter, with its low viscosity, gives more accurate results. But when a high numerical viscosity is needed to stabilise the scheme, the Minmod limiter is a better choice.

Of the three refinement criteria, the $C_{\partial\rho}$ criterion is the safest and the most robust. It is not very sensitive to changes in its settings and it does not cause repeated changes in the cell refinement level for high maximum levels. In all cases studied, the $C_{\partial\rho}$ criterion gives good results. The $C_{\partial^2\rho}$ criterion sometimes achieves the same accuracy with less computational costs, but it is very sensitive to small wiggles and it sometimes causes spurious grid refinement. However, for small maximum levels of refinement, it is an interesting alternative. The $C_{EE}$ criterion is based on a first-order truncation error and it is therefore not suitable for second-order problems. Results confirm this.

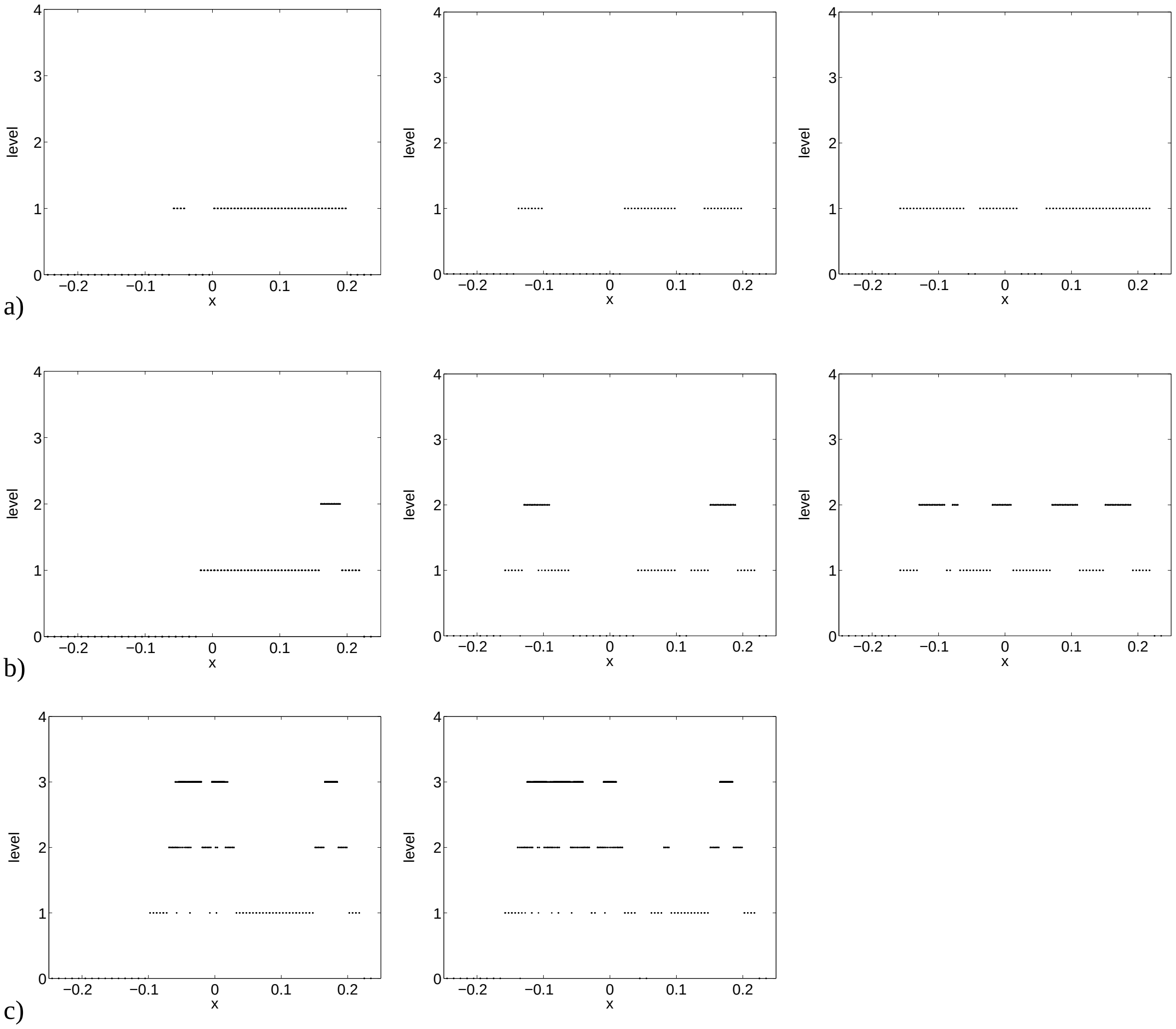


Figure 8.21: Sod problem, $C_{EE}$ refinement criterion. Cell level distributions with different settings of $d_s$: 0.032, 0.016 and 0.004. $d_m = 0.125d_s$, $d_{s\Delta} = 0.25d_s$. Modified Van Albada limiter. Maximum level 1 (a), 2 (b) and 3 (c).

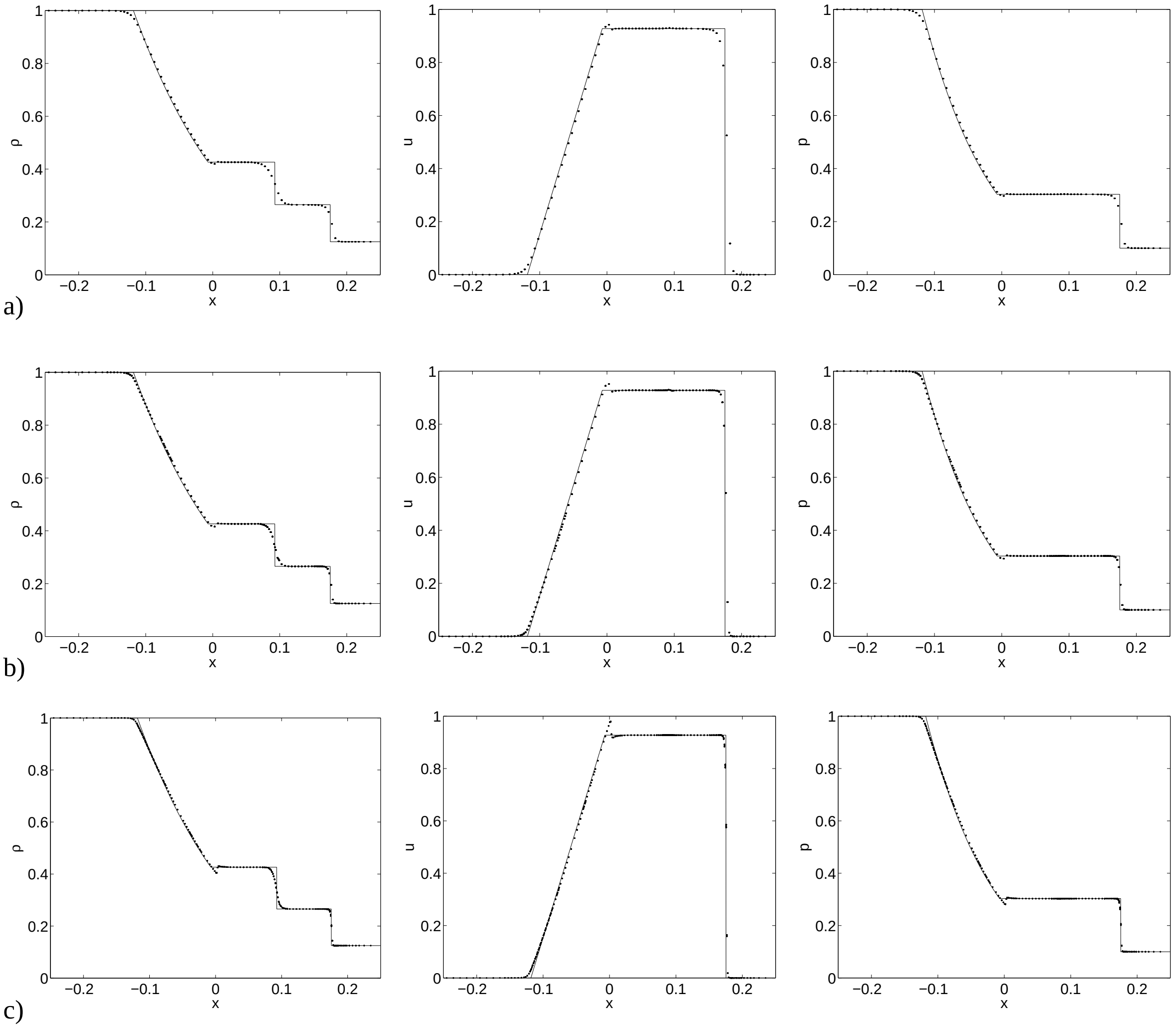


Figure 8.22: Sod problem, $C_{EE}$ refinement criterion. Solution for maximum levels 1 (a), 2 (b) and 3 (c). $d_s$ = 0.008, $d_m$ = 0.001, $d_{s\Delta}$ = 0.002. Modified Van Albada limiter. Domain $x = [-0.25, 0.25]$, $t = 0.1$, $\Delta x/\Delta t = 8.0$.

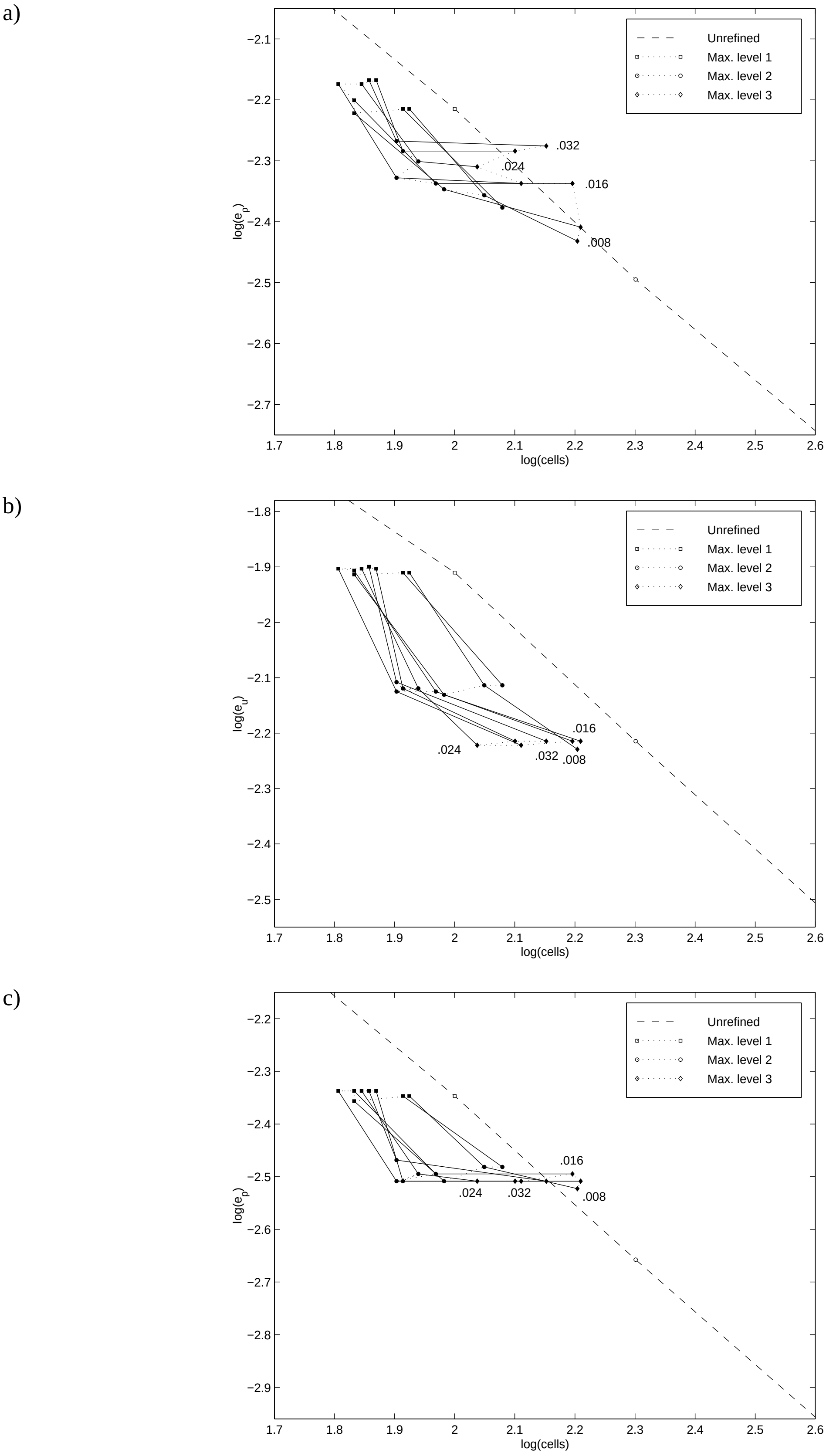


Figure 8.23: Sod problem, $C_{EE}$ refinement criterion, modified Van Albada limiter. Errors in $\rho$ (a), $u$ (b) and $p$ (c). Dashed line gives the errors for uniform grids, solid lines the errors for $d_s = 0.004 \ldots 0.032$ (indicated on the plots).

# Chapter 9
# Vortex shedding from a flat plate

The second-order adaptive-gridding method from chapter 7 is tested in chapter 8 in a way that is common in CFD: the test problems concentrate purely on the newly developed feature, the discretisation. The geometry of the problems is simple and the effect of boundary conditions is kept as low as possible (the entropy error behind the step is corrected in the forward-facing step problem). Furthermore, the solution is known in advance to enable estimation of the obtained accuracy. But when a method is used in practice, the problems to be solved are different: geometries are usually complex and have a strong influence on the solution. And of course the solution is not known in advance, otherwise it would not have to be calculated...

As a final test for the current method, a test problem is desired that resembles these practical conditions as closely as possible, to find out if the method is suitable for practice. Curved boundaries are not yet possible, but a problem for which no solution is known in advance can be selected. To make the test problem even more interesting, two more requirements are suggested:

- One possible field of application for the code are aerospace calculations. Since the current work is a master's thesis in Aerospace Engineering, a test problem is desired that is aerospace related.
- In the previous two test problems, the emphasis lies on features from gasdynamics: shocks, expansions and contact discontinuities in supersonic flow. Therefore, it is desired to have a test case with other flow features.

There is one more reason why a third test is needed. The previous two tests were used in developing the method, therefore the method may only be error-free for these two problems. But if the flow solver can handle a third, new problem without changes, it is probably suitable for most problems.

The test problem chosen here is the shedding of vortices from a flat plate at an angle of attack. The plate is at rest in a fluid at rest and is suddenly set into motion. From then on, the velocity of the plate is kept constant. This is a test problem that fulfils the requirements listed above: with the current rectangular grids and boundary conditions, a flat plate is the closest we can come to a real airfoil. And subsonic vortex flow is an interesting new phenomenon to test the method.

The implementation of the problem and the results obtained are described in the following sections. The problem is unknown, therefore a convergence test is done on uniform grids, to make sure that a unique solution exists. Then a refinement criterion is chosen and adapted to the problem with the help of these uniform solutions. With this calibrated refinement criterion, representative results are calculated on refined grids.

## 9.1. Problem definition

In this section, the test problem is defined. Both the physical parameters and the numerical implementation are given.

*Physical problem* A flat plate with chord $\bar{c} = 1$ is placed in a flow that has uniform initial conditions: $p_\infty = 1$, $\rho_\infty = 1.4$, $u_\infty = 0.5\cos(\alpha)$ and $v_\infty = 0.5\sin(\alpha)$. A perfect gas with $\gamma = 1.4$ is assumed. This is a flow with Mach number $M_\infty = 0.5$ under an angle of attack $\alpha$ (see figure 9.1b). The flow corresponds to a flat plate that is suddenly set into motion in a flow at rest, as in figure 9.1a, but now the plate is fixed in space.

*Computational domain* The problem is solved on a rectangular domain, see figure 9.2. Two domain sizes are used, the small domain has size $7 \times 4$, this domain is used for calculations on uniform grids. A larger domain of size $12 \times 12$ is used later on. This domain is still relatively small, as steady calculations on airfoils are often performed on grids with dimensions of about 100 chord sizes [11, 13].

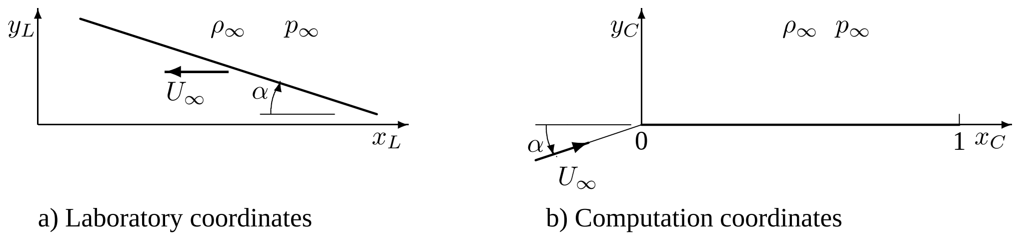


Figure 9.1: Flat plate, geometry and initial conditions.

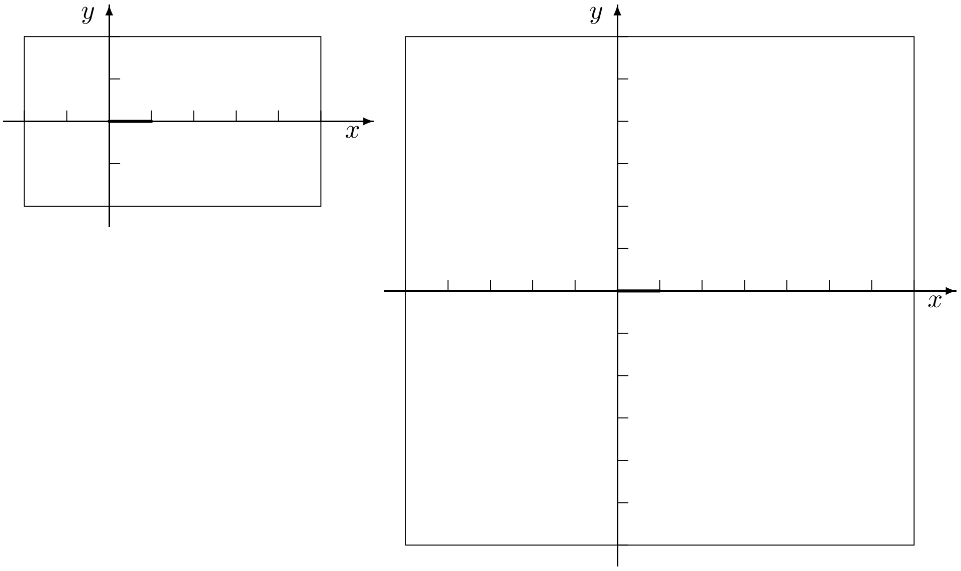


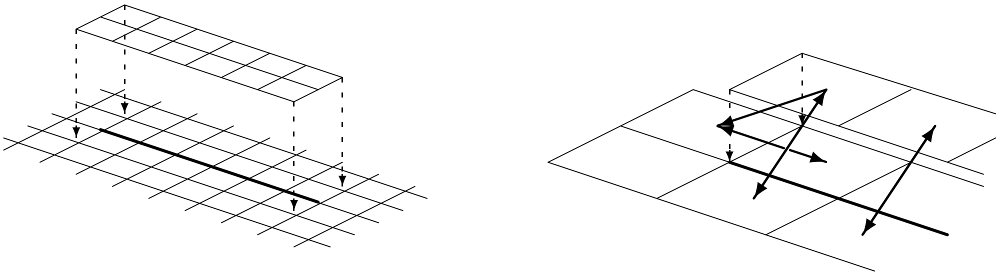
Figure 9.2: Flat plate, two computational domains. The thick line indicates the flat plate.

Boundary conditions are imposed on the farfield edges and on the flat plate. For positive $\alpha$, a subsonic inflow boundary condition (2.21) is specified on the left and the lower farfield boundary; $u$, $v$ and $c$ from the initial conditions are prescribed there. On the right wall, subsonic outflow (2.22) is assumed and $p_\infty$ is specified. On the upper side, the boundary state is assumed to be equal to the state in the flow (like at a supersonic outflow boundary), because specifying the pressure there causes a spurious pressure gradient to move down into the fluid. This is not elegant, but it works. On the flat plate, wall boundary conditions (2.23) are specified on both sides.

*Numerical implementation* The computational domain is divided into square cells. For the computations with adaptive gridding, a coarse $\Delta x$ of 0.5 is chosen, which gives 112 cells in the small domain and 576 cells in the large domain. Uniform calculations are performed on finer grids.

The time step for uniform grids is set at $\Delta t = 1/5\Delta x$. With an estimated $|\lambda|_{\max}$ of 2, this gives a CFL number of 0.4, which proves to be low enough for stability. Refined grids require a CFL number below 0.25 for stability, so here $\Delta t = 1/10\Delta x$ is chosen. The time steps for the uniform grid are not changed to the lower $\Delta t$ here, to make a fair comparison between CPU times for problems on adapted and uniform grids, with realistic time steps for both problems.

The implementation of the boundary conditions on the flat plate is not straightforward, because there is no space available to place boundary cells outside the computational domain. Therefore, the boundary cells overlap the normal cells around the plate: the boundary cells for the lower boundary of the plate are in the same position as the first row of normal cells *above* the plate and vice versa (figure 9.3a).



a) The wall boundary cells

b) Neighbour pointers (detail)

Figure 9.3: Placement of boundary cells for the flat plate boundary and neighbour pointers of these cells. In b), only the upper boundary cells are shown and most pointers are omitted for clarity.

For the calculation of virtual cells at the boundary, all the boundary cells must have a left and right neighbour. This poses problems at both ends of the row (figure 9.3b), as the last boundary cells in the row do not have a normal neighbour. Therefore, the left or right neighbour pointers of these cells are pointed at the normal cells just to the left or the right of the boundary cells, but these pointers are *not* made mutual: the normal cells have a neighbour pointer to their normal neighbour.

With this arrangement of the neighbour pointers, the computer code can handle the plate boundary without being changed. Only the refinement procedure does not work with this arrangement, therefore the four cells around the leading and trailing edge of the flat plate are fully refined before the calculation starts and kept at level $M$ during the entire calculation.

## 9.2. Convergence study

The problem chosen above, the shedding of vortices from a flat plate, does not have a solution that is precisely documented in the literature, like the forward-facing step problem. It is not even certain that the problem has a unique solution at all. Therefore, a grid convergence study is done on uniform grids, before the calculations on adapted grids are started.

Two cases are studied, with an angle of attack $\alpha = 5^o$ and $\alpha = 20^o$, both on the small computational domain. The solutions are calculated from $t = 0$ to $t = 3$, because reflections from the farfield boundary influence the flat plate at later times. We compare three quantities:

- The distribution of the pressure coefficient $C_p$ over the flat plate for $t = 1$ and $t = 3$. $C_p$ is defined as

$$C_p = \frac{p - p_\infty}{\frac{1}{2}\rho U_\infty^2}. \tag{9.1}$$

- The change of the normal force coefficient $C_n$ with time. This coefficient is obtained by integrating $C_p$ over the flat plate,

$$C_n = \frac{1}{\bar{c}} \int_{\text{l.e.}}^{\text{t.e.}} \left(C_{p,\text{below}} - C_{p,\text{above}}\right) \, dx. \tag{9.2}$$

- The magnitude of the velocity vector $|U|$ at $t = 3$. This variable gives an impression of the flow in the entire domain.

Solutions are calculated on grids with $\Delta x = 1/16$, $\Delta x = 1/32$ and $\Delta x = 1/64$. Results are given in figures 9.4, 9.5 and 9.6, CPU times for these solutions in table 9.1.

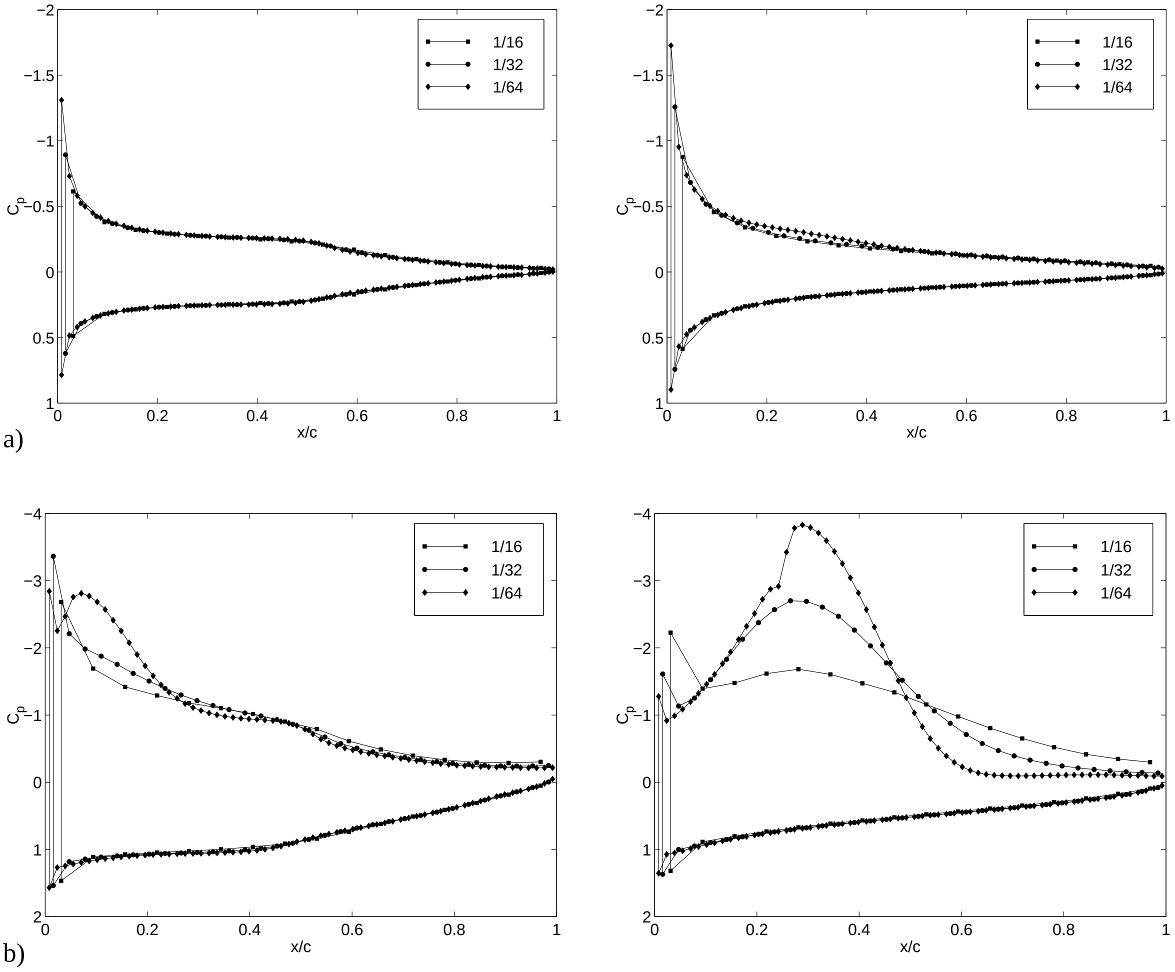


Figure 9.4: Flat plate, $C_p$-distribution over the plate at $t$ = 1 (left graphs) and $t$ = 3 (right graphs) for $\alpha = 5^o$ (a) and $\alpha = 20^o$ (b). Uniform grids, small domain.

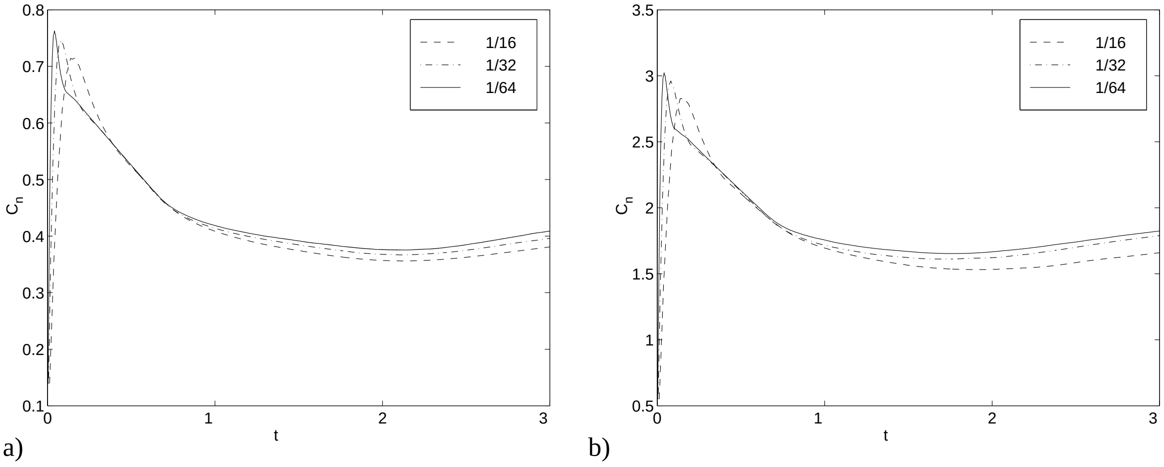


Figure 9.5: Flat plate, normal force coefficient $C_n$ versus time for $\alpha = 5^o$ (a) and $\alpha = 20^o$ (b). Uniform grids, small domain.

The results are briefly discussed here, only to explain the convergence results. A thorough discussion of the flow is postponed to section 9.4.

The $C_p$-distributions for $\alpha = 5^o$, figure 9.4a, show that the flow field is essentially converged on the coarsest grid, $\Delta x = 1/16$. The normal force coefficient (figure 9.5) for the three solutions confirms this, although there is a difference between the lines. Between $t = 1$ and $t = 3$, the $C_p$-distribution approaches the shape of the steady pressure distribution around a flat plate.

For $\alpha = 20^o$, the $C_p$-distributions show a large grid dependency, see figure 9.4b. At $t = 3$, a large area of strong suction is visible on the upper side of the flat plate. Its shape clearly depends on the mesh width. This region is caused by the shedding of a vortex from the leading edge, the so-called dynamic stall vortex. On fine grids, this vortex becomes stronger, but it is spread over a smaller area, therefore the development of $C_n$ for the three grids is almost the same (figure 9.5b). Note that, on the lower side of the plate, $C_p$ is converged. Until $t = 3$, the grid dependency is local: it is limited to the area above the plate.

Plots of the speed $|U|$ in the entire flow field (figure 9.6) confirm the observations made from the $C_p$-distributions. The flow fields are mostly converged for $\Delta x = 1/16$, except for a small area above the plate at $\alpha = 20^o$. The speed in the core of the starting vortex, seen at about one chord length behind the trailing edge, increases for smaller values of $\Delta x$, but the overall pattern of the vortex does not depend on this effect. Therefore, this grid dependency has no influence on the accuracy of the entire solution.

How can this grid dependency be explained? Theoretically, the Euler equations have no way to determine the circulation around an airfoil, since they are inviscid. For steady problems, one extra condition is needed to make the problem well-posed: the Kutta condition. This condition prescribes where the flow leaves the airfoil. Analogous, it is reasonable to assume that prescribing where the flow leaves the airfoil in an unsteady problem

| $\Delta x$ | 'Level' | Cells | CPU time (s) | CPU time (s) |
|---|---|---|---|---|
| | | | $\alpha = 5^o$ | $\alpha = 20^o$ |
| 1/16 | 3 | 7168 | 127 | 125 |
| 1/32 | 4 | 28672 | 1039 | 977 |
| 1/64 | 5 | 114688 | 7874 | 14372 |

Table 9.1: Flat plate, CPU times on uniform grids, small domain, $t = 3$. The last value for $\alpha = 20^o$ is probably erroneous and caused by an unusually high work load on the work station at that time.

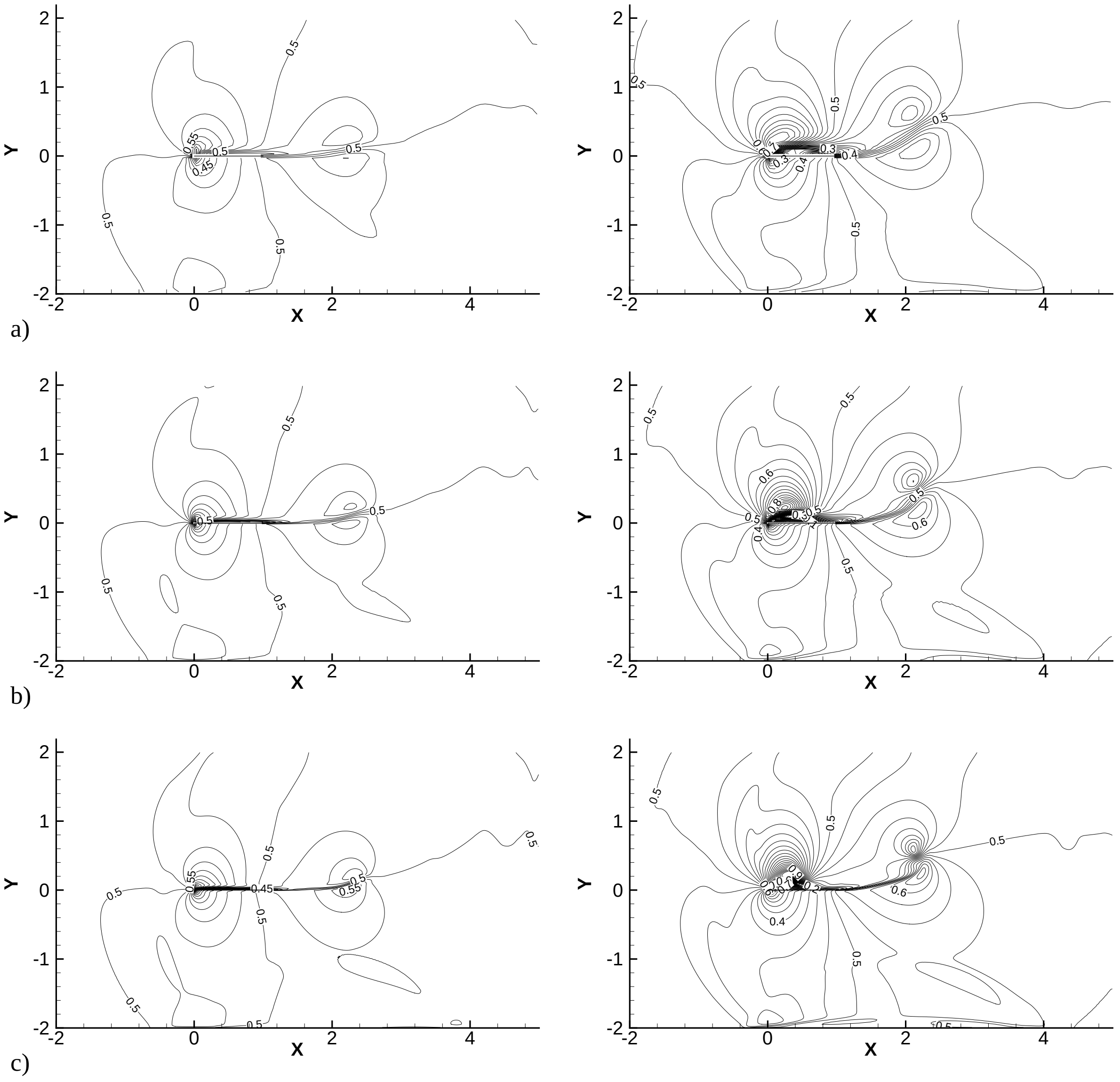


Figure 9.6: Flat plate, distribution of $|U| = \sqrt{u^2 + v^2}$ at $t$ = 3 for $\alpha = 5^o$ (left graphs, 0.0125 between the levels) and $\alpha = 20^o$ (right graphs, 0.025 between the levels). $\Delta x$ = 1/16 (a), $\Delta x$ = 1/32 (b) and $\Delta x$ = 1/64 (c).

is enough to determine the circulation in time. The numerical Euler solver has an implicit Kutta condition: numerical viscosity forces the flow to leave an airfoil at the trailing edge. Therefore, it is expected that the numerical flat plate problem is well-posed, when the flow leaves the plate *at only one location*.

On the stalled flat plate, the flow leaves the plate at two well-defined locations: the leading edge and the trailing edge. However, this flow is not well-posed in the unsteady case because separation at the leading edge does not occur immediately. During a small period after $t = 0$, the flow is attached and the length of this interval depends on the numerical viscosity and hence on $\Delta x$. In figure 9.4b, at $t = 1$, we see that separation has already started for $\Delta x = 1/64$, while the flow for $\Delta x = 1/16$ is still fully attached.

Concluding, the flat plate at $\alpha = 5^o$ does not stall and the problem converges. The flow solution at $\alpha = 20^o$ is grid-dependent. However, this problem is studied anyway, to obtain a qualitative picture of the flow behaviour.

## 9.3. Refinement criterion

Before adaptive-gridding calculations can be performed on the large computational domain, a refinement criterion is chosen and proper values for $d_s$ and $d_m$ are determined. This is done by performing calculations with adaptive gridding on the small domain and comparing these solutions with the corresponding uniform-grid solutions. One test case is chosen, the uniform solution with $\Delta x = 1/64$, which corresponds to a maximum grid level $M = 5$ on the $\Delta x = 1/2$ coarse grid.

The gradient $\rho$ refinement criterion is not used for this problem, because the subsonic flow around the flat plate has a relatively small variation in $\rho$. But a gradient criterion is the safest choice for a well-functioning refinement criterion, therefore the gradient of $\rho u$, the second component of the state vector, is used:

$$C_{\partial \rho u} = \max_{n=b,r,a,l} \left( \frac{|(\rho u)_{n_i} - (\rho u)_i|}{\frac{1}{2} + 2^{l_i - l_{n_i} - 1}} \right), \tag{9.3}$$

This choice works well, because the momentum gradient accurately indicates the regions of interest: the suction above the plate and the starting vortex. The values of $d_s$ and $d_m$ are determined below.

***The case $\boldsymbol{\alpha = 5^o}$*** Three values of $d_s$ are tested: 0.12, 0.08 and 0.04. $d_m$ is always chosen as $0.4d_s$. Results for this test are given in figures 9.7a and 9.8a, the CPU times for the test in table 9.2.

$C_p$-distributions at $t = 1$ and $t = 3$ are close to the uniform distributions, but there are some very small differences, especially at $t = 1$, see figure 9.7a. It is hard to see in these figures, but none of the three distributions is always closer to the uniform distribution than the others. But on the other hand, the distributions for $d_s = 0.04$ are smoother and resemble the shape of the uniform curve better than the other distributions. The normal force development for $d_s = 0.04$ is clearly superior to the others (figure 9.8a), the peak in $C_n$ in the first halve time unit is calculated much more accurately. Therefore the finest value, $d_s$ = 0.04 and $d_m = 0.016$, is chosen for $\alpha = 5^o$.

***The case $\boldsymbol{\alpha = 20^o}$*** For this case, calculations are again performed with $d_s$ = 0.12, 0.08 and 0.04. The gradients are larger than for $\alpha = 5^o$, so more cells are refined and the CPU times are longer (table 9.2). Both the $C_p$-distributions and the $C_n$-curves resemble the uniform curves excellently, see figures 9.7b and 9.8b. Therefore, the highest $d_s$ can be used for the $\alpha = 20^o$ calculations.

Even the $C_p$-distributions above the plate are the same as the uniform distributions. So, for these values of $d_s$, the influence of numerical viscosity on the flow separation is *not* changed by using adaptive gridding

| $d_s$ | $\alpha = 5^o$ | | $\alpha = 20^o$ | |
|---|---|---|---|---|
| | CPU time (s) | % of unif. | CPU time (s) | % of unif. |
| 0.04 | 197 | 2.5 | 1263 | 8.8 |
| 0.08 | 136 | 1.7 | 533 | 3.7 |
| 0.12 | 117 | 1.5 | 293 | 2.0 |

Table 9.2: Flat plate, CPU times on refined grids, small domain, $M = 5$, $t = 3$.

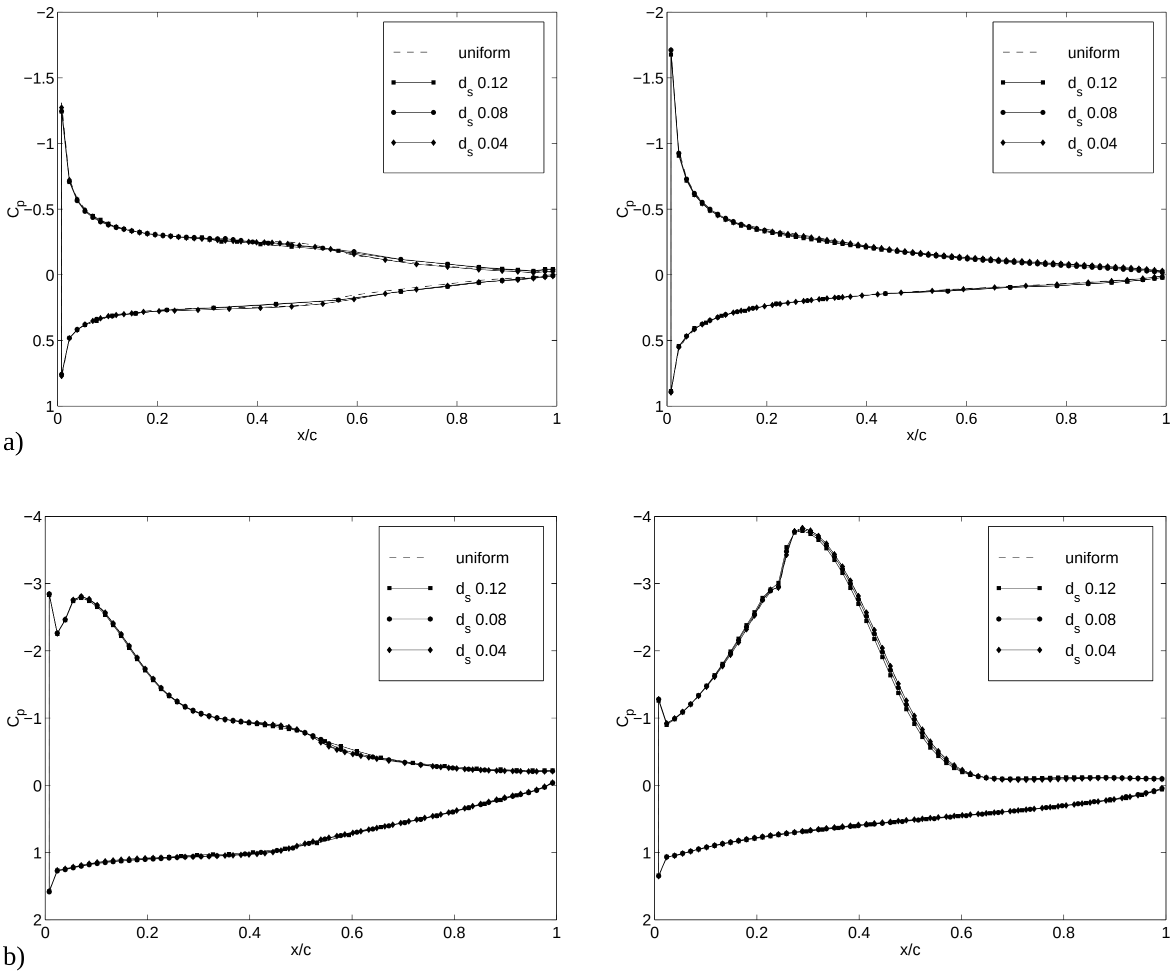


Figure 9.7: Flat plate, $C_p$-distribution over the plate at $t$ = 1 (left graphs) and $t$ = 3 (right graphs) for $\alpha = 5^o$ (a) and $\alpha = 20^o$ (b). Solutions on refined grids with $\Delta x = 1/2$, $M = 5$, $d_m = 0.4 d_s$ compared with solution on uniform grid, $\Delta x = 1/64$. Small domain.

instead of uniform grids. It depends only on the size of the smallest cells in the grid.

Figure 9.9 shows the distributions of $|U|$ and the refined grids at $t = 3$ for the chosen values of $d_s$ and $d_m$. The solution near the flat plate and the starting vortex are the same as the uniform solutions 9.6c, but the reflections from the boundaries are damped by the coarse cells away from the flat plate. Before $t = 3$, this does not influence the solution near the plate.

## 9.4. Results

With the refinement criterion chosen in the previous section, adaptive-gridding calculations are performed on the large computational domain. On this domain, computations can be run until $t = 9$ before the farfield boundaries start influencing the solution. In the $\alpha = 5^o$ test case, the development and movement of the starting vortex is studied. For $\alpha = 20^o$, a qualitative study is made of the dynamic stall phenomena on the flat plate. CPU times for these calculations are given in table 9.3.

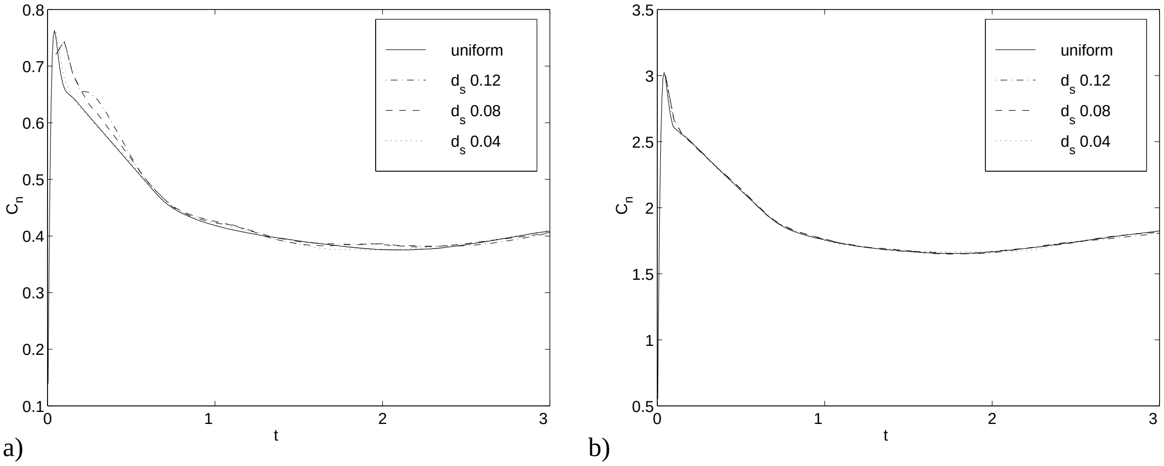


Figure 9.8: Flat plate, normal force coefficient $C_n$ against time for $\alpha = 5^o$ (a) and $\alpha = 20^o$ (b). Solutions on refined grids with $\Delta x = 1/2$, $M = 5$, $d_m = 0.4d_s$ compared with solution on uniform grid, $\Delta x = 1/64$. Small domain.

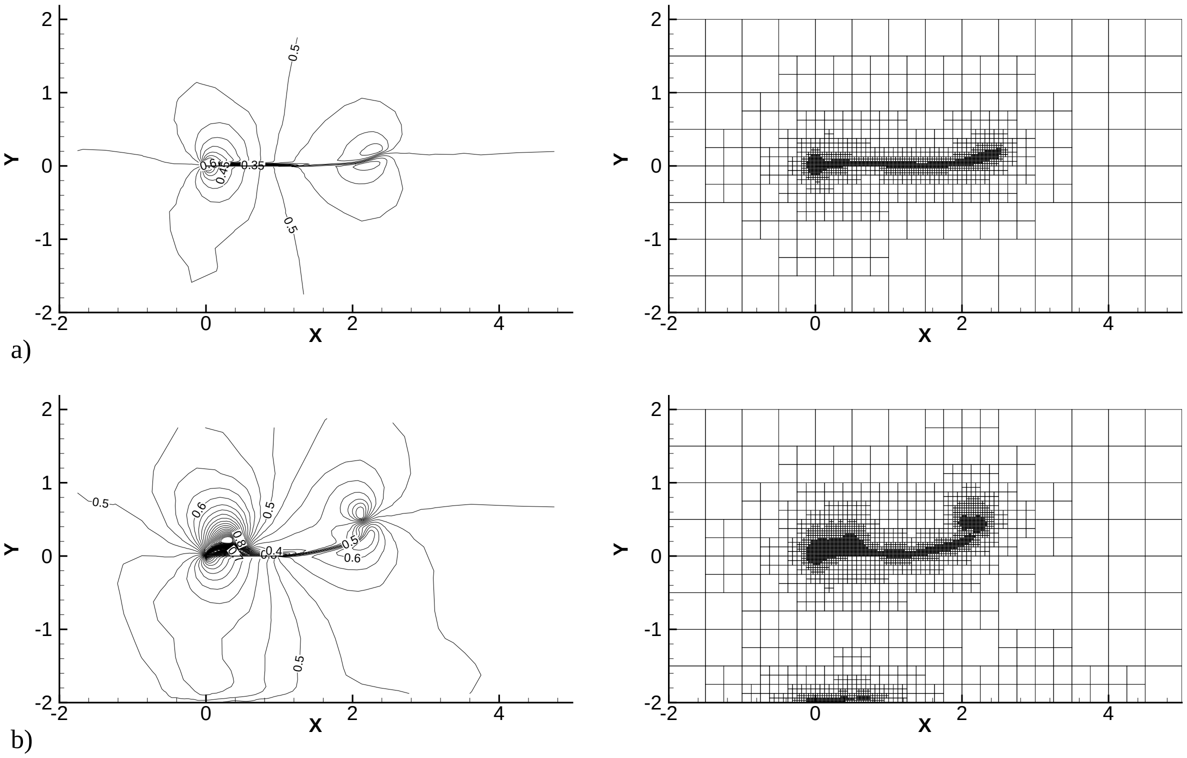


Figure 9.9: Flat plate, distribution of $|U| = \sqrt{u^2 + v^2}$ at $t$ = 3 and refined grid at $t = 3$ for $\alpha = 5^o$, $d_s = 0.04$, $d_m = 0.016$ (a) (0.0125 between the levels) and $\alpha = 20^o$, $d_s = 0.12$, $d_m = 0.048$ (b) (0.025 between the levels). $\Delta x$ = 1/2, $M$ = 5.

### 9.4.1 *Starting vortex, $\alpha = 5^o$*

A lift-generating airfoil in a steady flow contains a ‘bound’ vortex, a vortex fixed to the airfoil, which is one explanation of why the airfoil generates lift. But, according to Kelvin’s theorem [1], vorticity cannot be created out of nothing, the total sum of vorticity in a flow must remain constant. Therefore, another vortex with a strength that is exactly opposite to the bound vortex must exist somewhere in the flow. This is the so-called starting vortex. It is shed from the trailing edge when the airfoil is accelerated from rest and it moves with the flow, away from the airfoil.

In the flow field around the suddenly started flat plate, the appearance and movement of the starting vortex are studied, as well as the development of lift on the flat plate.

Figure 9.10 gives an overview of the speed distribution in the flow field. Just after the plate is started, the flow on the plate surface is not tangent to the plate, but this changes immediately. The velocity vectors in the flow on the plate surface are turned to make the flow tangent to the surface. To do this, a high pressure develops below the plate and a low pressure above the plate. This pressure and the velocity on the plate are *the same* everywhere, because the flow has not yet had time to ‘see’ that the plate is finite. This is still apparent in the grid at $t = 0.25$, figure 9.10a, that has two perfectly horizontal bands of refined cells, above and below the flat plate.

Because of the pressure difference on the plate, fluid starts to flow up around the leading and the trailing edge. This is seen in figure 9.10a. The areas influenced by this cross flow grow (figure 9.10b) and around $t = 0.75$, they meet, both above and below the plate (figure 9.10c). From that moment, the region around the leading edge continues to grow, but its centre remains at the leading edge: a stagnation point below the plate and a suction peak above the plate develop there. However, the region near the trailing edge is convected away from the trailing edge, until it has completely left the flat plate in figure 9.10g. This is the starting vortex. It is connected to the flat plate by a more or less straight wake.

From figure 9.10e–h, it is seen that the starting vortex grows in time, but the velocity in the core diminishes. In the centre of the vortex, we see that the wake is curled around the vortex core.

The starting vortex is most obvious when the flow is studied in laboratory coordinates $x_L$ and $y_L$ or, in other words, when the flow is at rest and the flat plate moves. Vector plots of the velocity in this coordinate system are given in figure 9.11. (In these plots, one vector is drawn per cell. The different vector densities are caused by the different levels of refinement of the cells.)

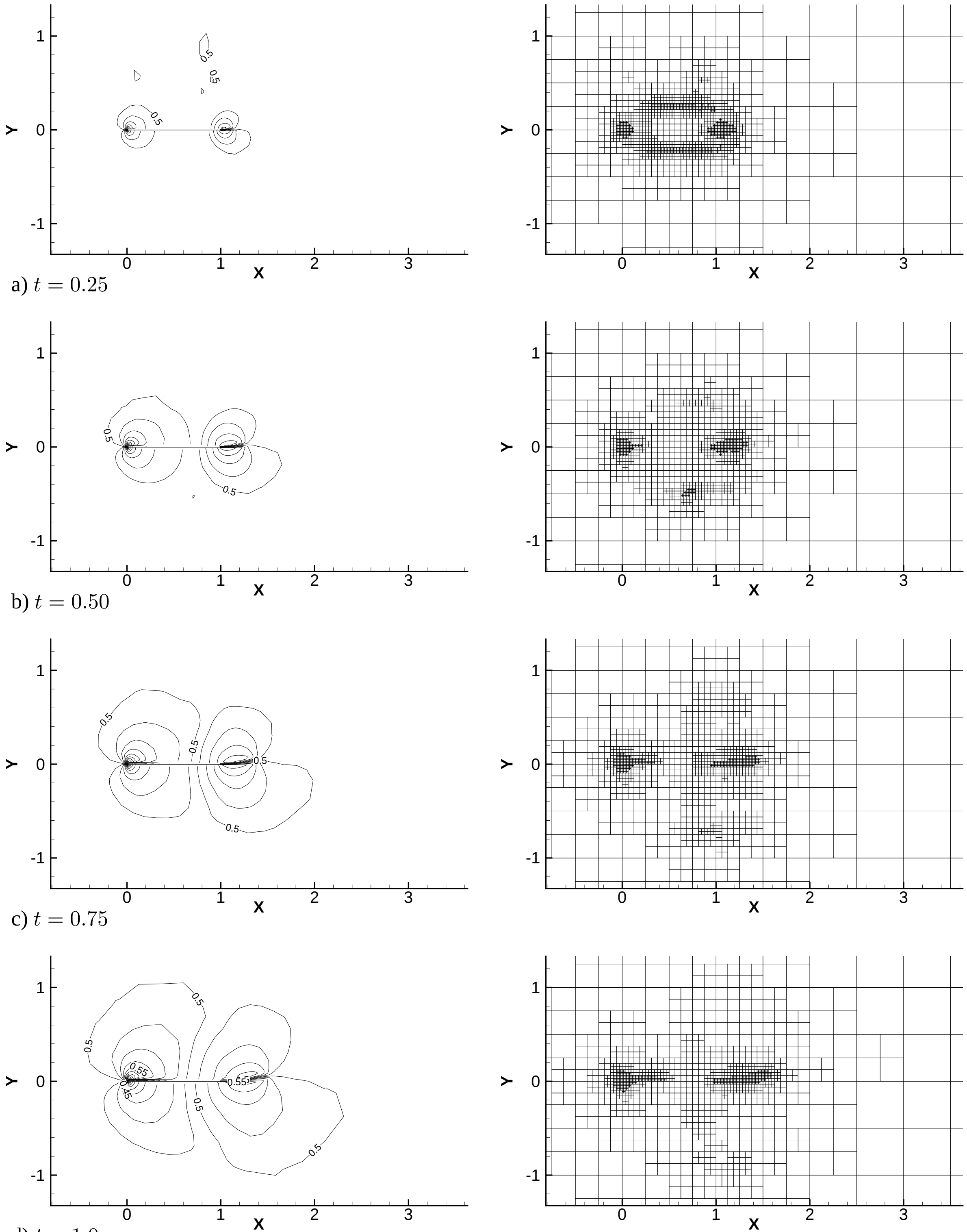
At $t = 0.25$, most of the velocity induced by the flat plate is normal to the plate: the flow is pressed or sucked away by the plate. The flow here is already tangent to the plate, but that cannot be seen from these plots, because the flat plate moves relative to the pictures. The starting vortex is growing here behind the trailing edge. At $t = 0.5$ (figure 9.11b) the vortex is still growing and its centre has moved to the left. At $t = 0.75$, when the velocity regions from the leading edge and the trailing edge have met, the vortex stops growing and the core remains almost fixed in space.

After $t = 0.75$, vorticity is still shed from the plate, but not as a consistent starting vortex. The wake between the plate and the starting vortex becomes a vortex sheet, a region of distributed vorticity. This can be seen from the velocity vectors: when they cross the wake, their components normal to the wake do not change, but their tangential components do. This is typical for a vortex sheet.

The change between the shedding of the starting vortex and of the vortex wake happens exactly when the flow pattern from the trailing edge meets the flow pattern from the leading edge. Therefore, it is assumed that the forward movement of the trailing edge pattern is associated with the starting vortex and the backward

| $\alpha$ | CPU time (s) | % of unif. |
|---|---:|---:|
| $5^o$ | 1448 | 1.2% |
| $20^0$ | 2050 | 1.7% |

Table 9.3: Flat plate, CPU times for calculations on the large computational domain, $M = 5$, $t = 9$. These times are compared with an *estimated* uniform CPU time of 121,500 s, based on the 7875 s time from table 9.1.



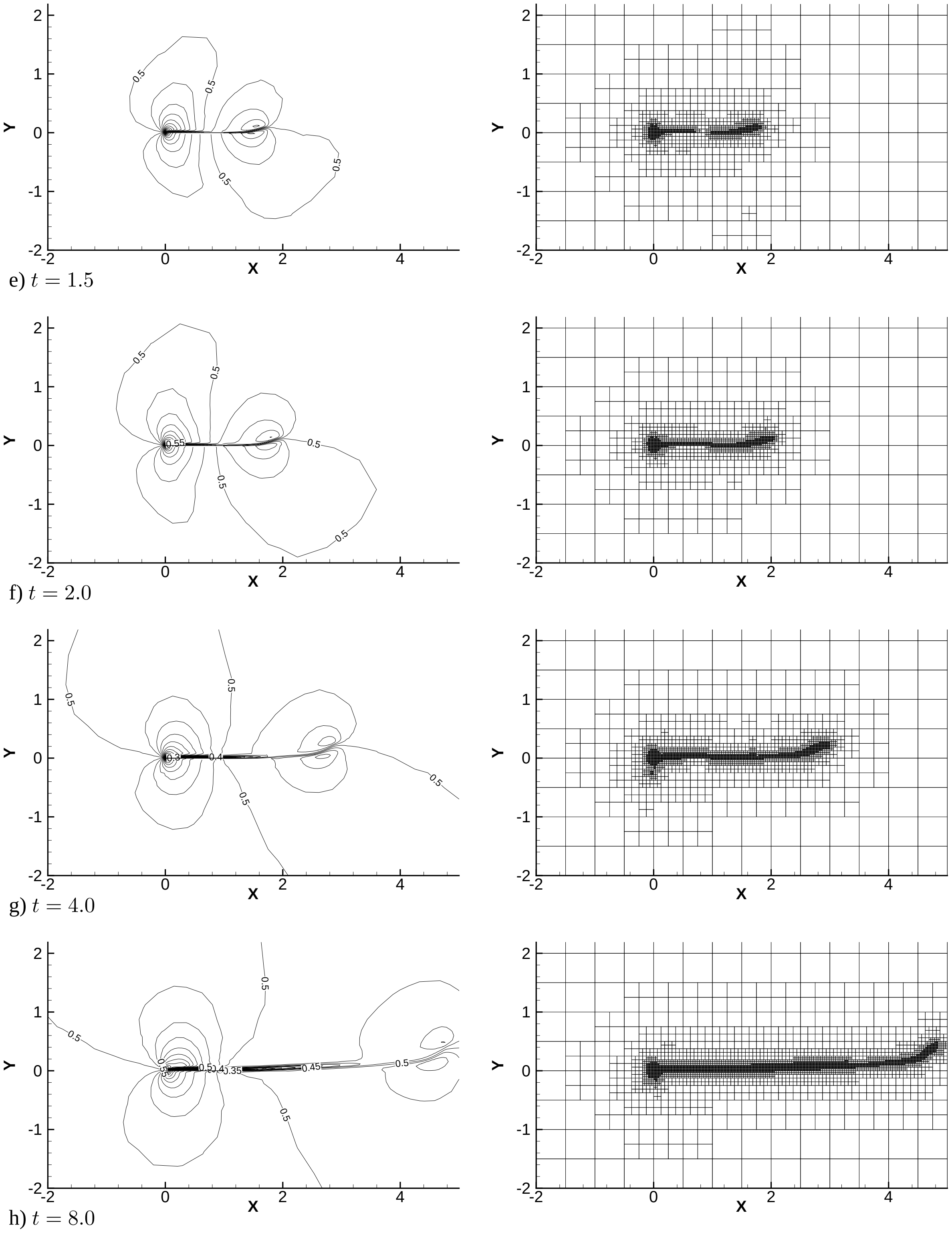
a) $t = 0.25$

b) $t = 0.50$

c) $t = 0.75$

d) $t = 1.0$

Figure 9.10: Flat plate, distribution of $|U| = \sqrt{u^2 + v^2}$ and refined grid for $\alpha = 5^o$ (0.0125 between the levels). $\Delta x$ = 1/2, $M$ = 5, $d_s = 0.04$, $d_m = 0.016$.



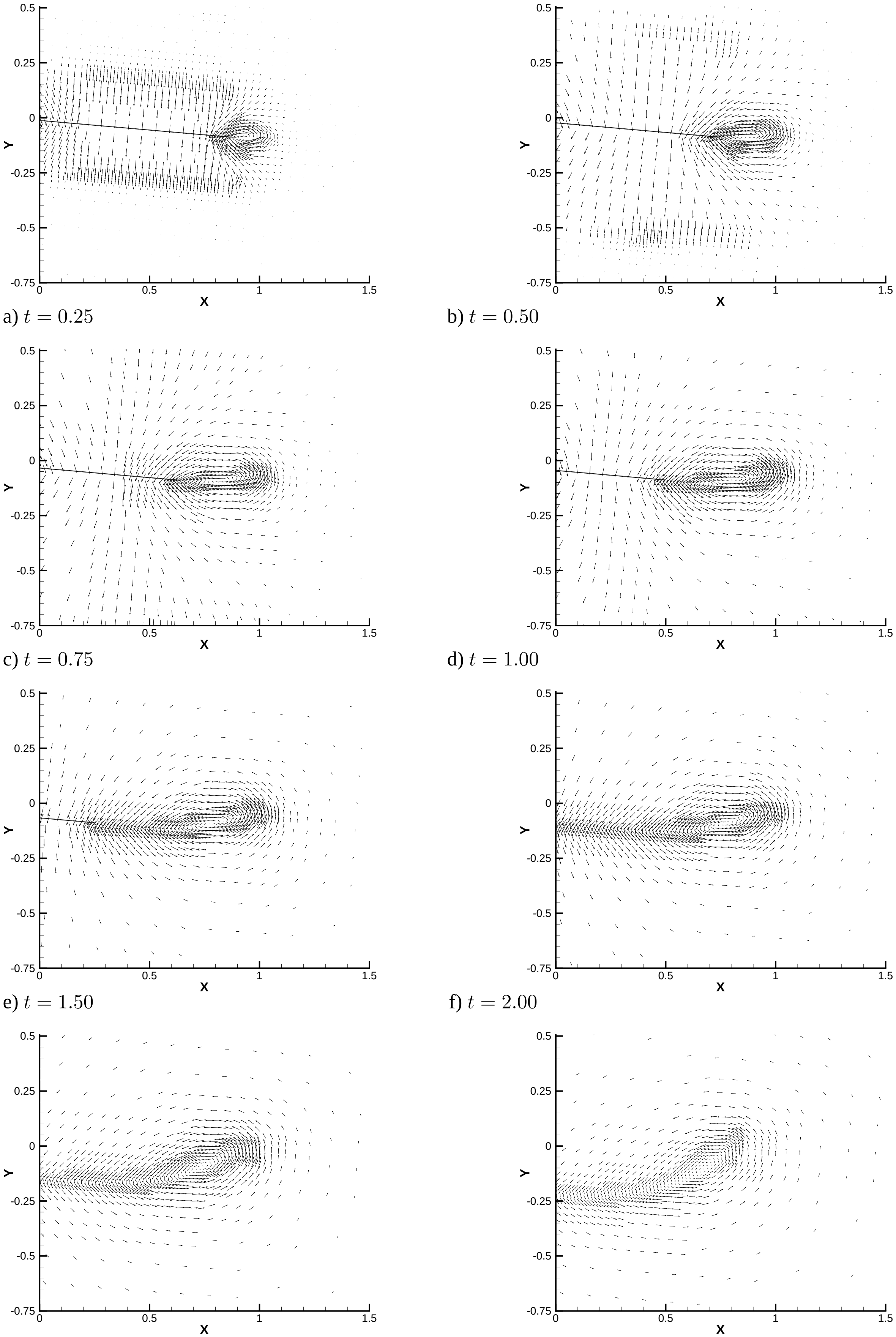
e) $t = 1.5$

f) $t = 2.0$

g) $t = 4.0$

h) $t = 8.0$

Figure 9.10: (cont.) Flat plate, distribution of $|U| = \sqrt{u^2 + v^2}$ and refined grid for $\alpha = 5^o$. The area shown in these figures is larger than in the previous four.



a) $t = 0.25$

b) $t = 0.50$

c) $t = 0.75$

d) $t = 1.00$

e) $t = 1.50$

f) $t = 2.00$

g) $t = 4.00$

h) $t = 8.00$

Figure 9.11: Flat plate, vector plots of the velocity for $\alpha = 5^o$. The velocity is taken relative to the $x_L, y_L$-axes: the fluid is initially at rest and the flat plate moves.

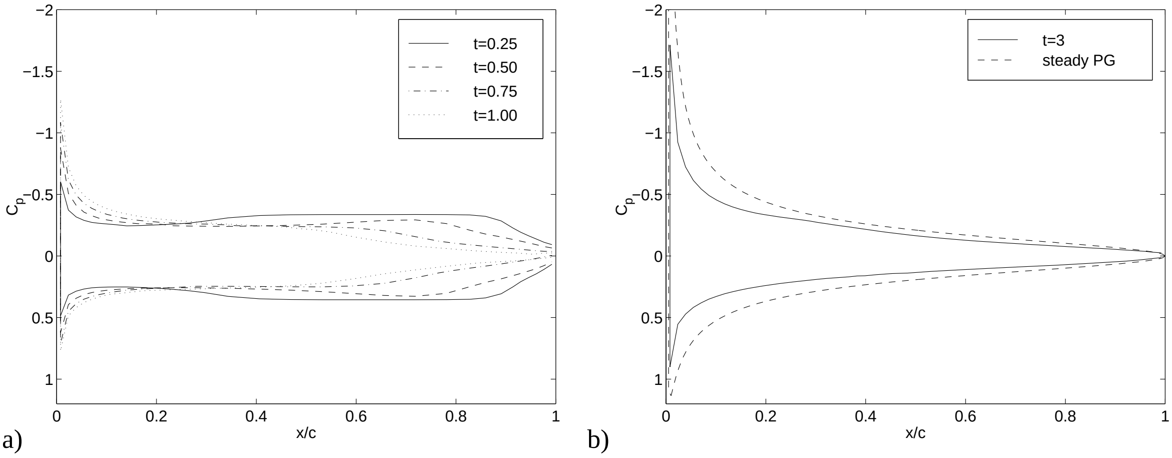


Figure 9.12: Flat plate, $\alpha = 5^o$. $C_p$-development in the first time unit (a) and $C_p$ at $t = 3$, compared with a solution of the steady Prandtl-Glauert equation (b).

movement of the same pattern with the vortex sheet.

A last point of interest is the movement of the wake. Due to the velocity induced by the starting vortex and the flat plate, the wake is moved downward. This can be seen in figures 9.11g–h.

The development of pressure over the flat plate is illustrated by plots of the pressure coefficient $C_p$, figure 9.12. Most changes occur before $t = 1$: at $t = 0.25$ the region of constant pressure is seen, as well as the areas influenced by the leading and the trailing edge. Later on, we see how these regions meet and how the suction peak develops on the leading edge. After $t = 1$, the shape of the $C_p$-distribution remains more or less constant.

The $C_p$-distribution at $t = 3$ is compared with an analytical solution for the steady potential equation around a flat plate in figure 9.12b. For a Mach number $M = 0$, this solution is [1]

$$C_{p_{pot,M=0}} = \begin{cases} 1 - \left(1 + \alpha\sqrt{\frac{\bar{c}}{x} - 1}\right)^2 & \text{upper side,} \\ 1 - \left(1 - \alpha\sqrt{\frac{\bar{c}}{x} - 1}\right)^2 & \text{lower side.} \end{cases} \tag{9.4}$$

From this distribution, the $C_p$ at finite, subsonic Mach numbers is calculated with the Prandtl-Glauert rule,

$$C_{p_{pot}} = \frac{C_{p_{pot,M=0}}}{\sqrt{1 - M^2}}. \tag{9.5}$$

This $C_p$ is plotted in figure 9.12b. The shape of this $C_p$-distribution corresponds well with the numerical solution, but the numerical solution has smaller $C_p$. The lift at $t = 3$ has not yet reached its steady value. After $t = 3$, a small stall vortex appears, so later $C_p$-distributions cannot be compared adequately with the steady potential-flow solution anymore.

Finally, in the normal force development (figure 9.13), a strong peak is seen at low $t$, caused by the pressure needed for the first acceleration of the flow. Then the flow starts leaking around the leading and trailing edges and the normal force drops. After $t = 0.75$, the suction peak starts to grow and the normal force rises again.

As a reference, an analytical estimate of the steady $C_n$ is included. Again, with the Prandtl-Glauert rule,

$$C_{n_{pot}} = C_{l_{pot}} \cos(\alpha) = \frac{2\pi\alpha}{\sqrt{1 - M^2}} \cos(\alpha). \tag{9.6}$$

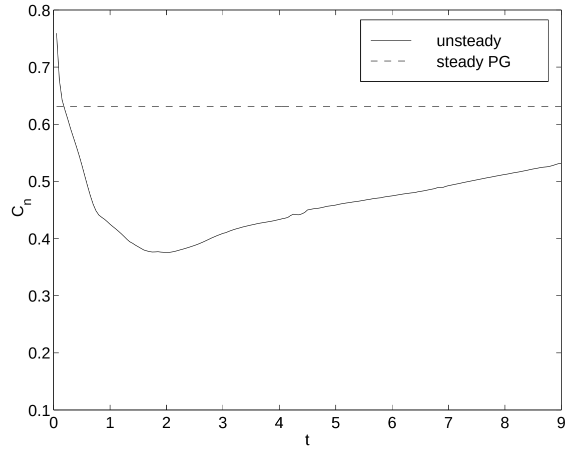


Figure 9.13: Flat plate, $\alpha = 5^o$. The normal force coefficient $C_n$ in time, compared with the steady value from the Prandtl-Glauert equation.

We see that $C_n$ is initially higher than $C_{n_{pot}}$, then it drops below the steady value and starts rising towards it again. The same behaviour is reported by Graham [7] for potential flow calculations with discrete vortex shedding and by Mehta and Lavan [14] for laminar Navier-Stokes solutions.

### *9.4.2 Dynamic stall vortex, $\boldsymbol{\alpha = 20^o}$*

For the large angle of attack, $\alpha = 20^o$, the shedding of the starting vortex and the initial development of lift are comparable with the $\alpha = 5^o$ case, as is seen from the $|U|$-plots in figure 9.14. The starting vortex is stronger and the wake is rotated more (figure 9.14e–h), but the overall pattern is the same. Therefore, the discussion here focuses on the formation of the so-called dynamic stall vortex. We know from section 9.2 that this vortex is strongly grid-dependent, but a qualitative analysis of the flow is worthwhile.

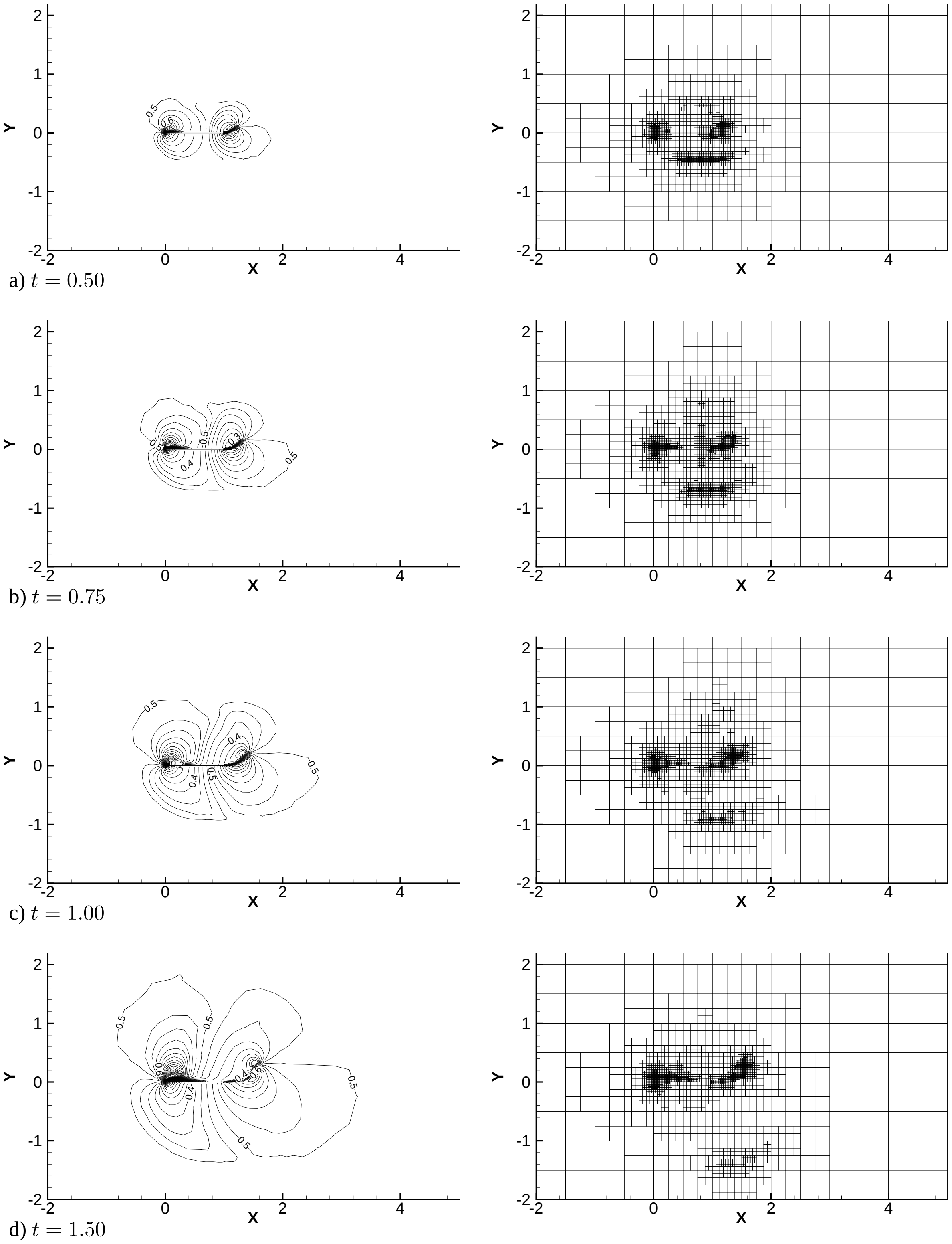
A dynamic stall vortex appears when an airfoil in motion is quickly rotated beyond the $\alpha$ for static stall. Then, the flow separates above the airfoil and a vortex is formed, the dynamic stall vortex. This is again expected from Kelvin's theorem: when the airfoil loses lift, the vorticity must stay *somewhere*. The most interesting feature of dynamic stall is that the stall vortex creates a large suction when it is still above the airfoil. Fighter aircraft use this in fast manoeuvres to reach lift coefficients far above the static maximum.

The dynamic stall vortex is again studied with vector plots, but now the axes move with the flat plate (figure 9.15). The first picture shows a flow that is still attached, but with high speeds and strong suction at the leading edge. (Note, by the way, the flow tangency at the plate surface.) The onset of separation occurs at $t = 0.75$, figure 9.15b, when the velocity becomes zero on the surface right behind the leading edge. At $t = 1$, the flow separates clearly from the leading edge and the first reverse flow is seen on the surface, figure 9.15c. Half a time unit later, the flow has the shape of a small vortex, see figure 9.15d. This vortex grows until it covers the entire upper side of the plate (figure 9.15f).

At $t = 5$, a 'bump' is seen in the vectors at about 40% chord, above the plate. This is a surprising feature: a plot of the local Mach number (figure 9.16) indicates that it is a shock wave! The speeds generated by the stall vortex are so high that two local supersonic areas appear in the flow. The lower lies in the reverse flow area on the plate surface. This supersonic area has a shape that is common for transonic airfoils [11], but here it is reversed.

When the stall vortex covers the entire plate, it continues to grow further downstream. At $t = 7$, it starts to suck air from below the trailing edge (figure 9.15g). This air changes direction and merges with the reverse flow in the vortex. At $t = 8$, a secondary vortex has formed above the trailing edge, with a rotation direction that is the opposite of the first stall vortex. At the same time, a large dead water area develops behind the leading edge, where the vortex has left.

These last figures show the same flow pattern as an experimental result (figure 9.17) from Van Dyke [20], a



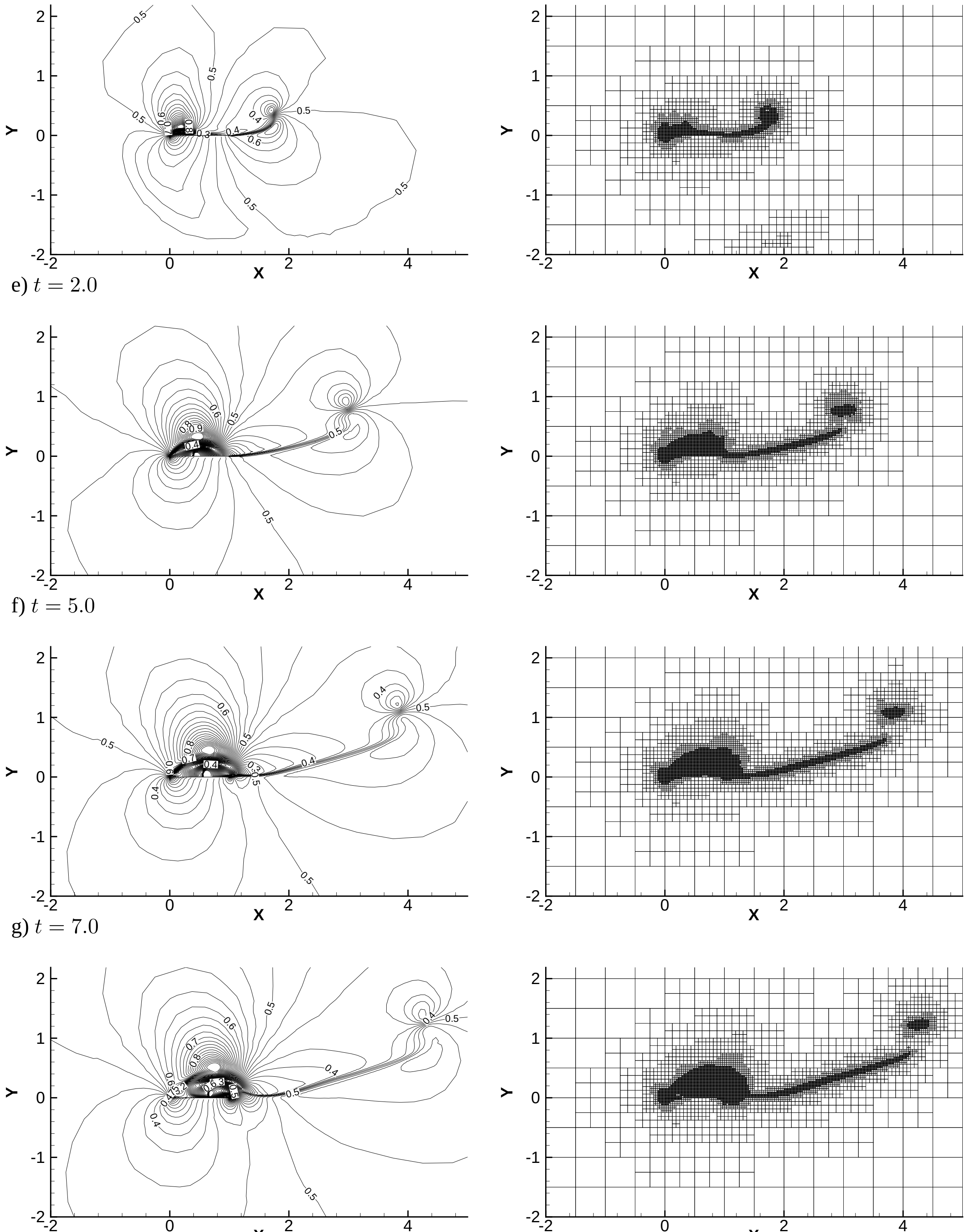
a) $t = 0.50$

b) $t = 0.75$

c) $t = 1.00$

d) $t = 1.50$

Figure 9.14: Flat plate, distribution of $|U| = \sqrt{u^2 + v^2}$ and refined grid for $\alpha = 20^o$ (0.025 between the levels). $\Delta x$ = 1/2, $M$ = 5, $d_s = 0.12$, $d_m = 0.048$.



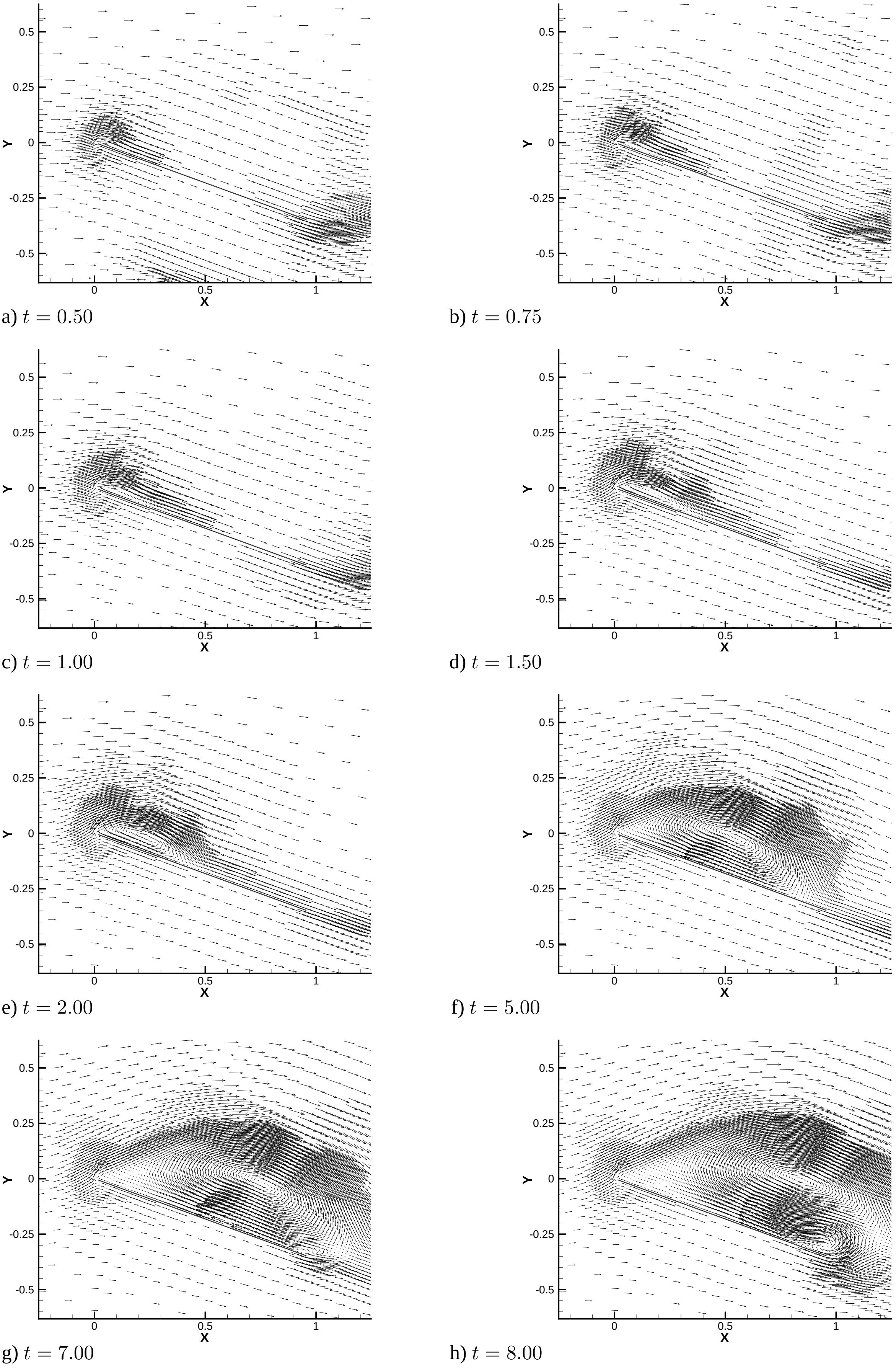
e) $t = 2.0$

f) $t = 5.0$

g) $t = 7.0$

h) $t = 8.0$

Figure 9.14: (cont.) Flat plate, distribution of $|U| = \sqrt{u^2 + v^2}$ and refined grid for $\alpha = 20^o$ .



a) $t = 0.50$

b) $t = 0.75$

c) $t = 1.00$

d) $t = 1.50$

e) $t = 2.00$

f) $t = 5.00$

g) $t = 7.00$

h) $t = 8.00$

Figure 9.15: Flat plate, vector plots of the velocity for $\alpha = 20^o$. The velocity is taken relative to the $x_C, y_C$-axes: the flat plate is kept fixed in space.

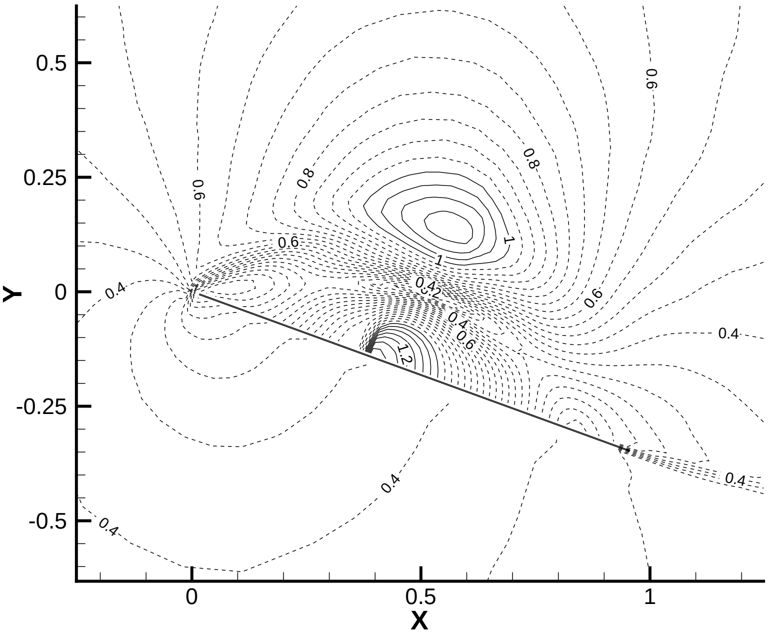


Figure 9.16: Flat plate, Mach number at $t = 5$. 0.05 between the levels, dashed levels have $M < 1$.

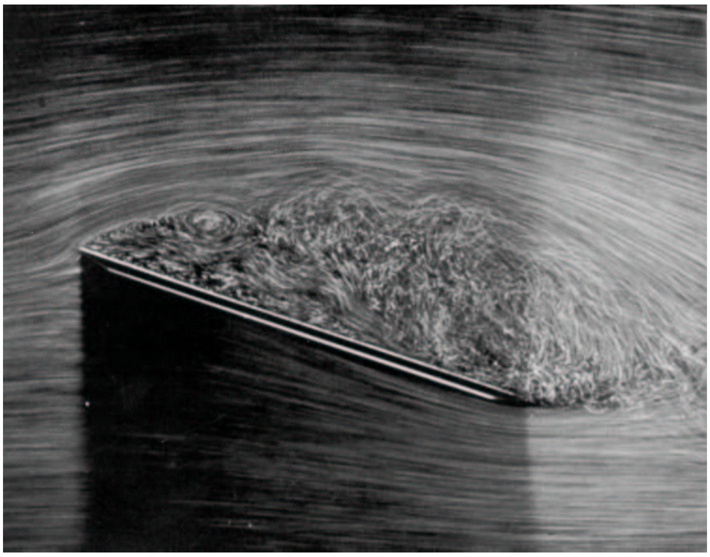


Figure 9.17: Global separation over a flat plate, experiment in water. $\alpha = 20^o$, $Re = 10.000$. Picture from Werlé [23], also in Van Dyke [20].

stalled flat plate at $\alpha = 20^o$ with a Reynolds number $Re = 10,000$. Of course, this is a viscous, incompressible flow and the picture is taken long after the flow has started, therefore the flow inside the separated area is different from the calculated flow. Also, the separated area is a little thinner in the experiment. However, the angle of separation at the leading edge and the flow outside the separated region resemble the calculated flow very well.

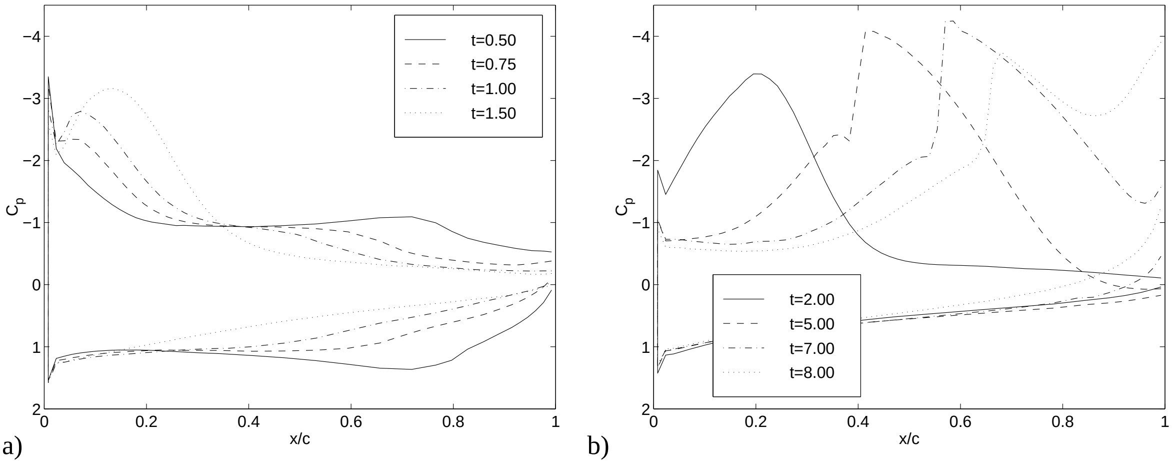


Figure 9.18: Flat plate, $\alpha = 5^o$. $C_p$-development 0.5 – 1.5 s (a) and 2 – 8 s (b).

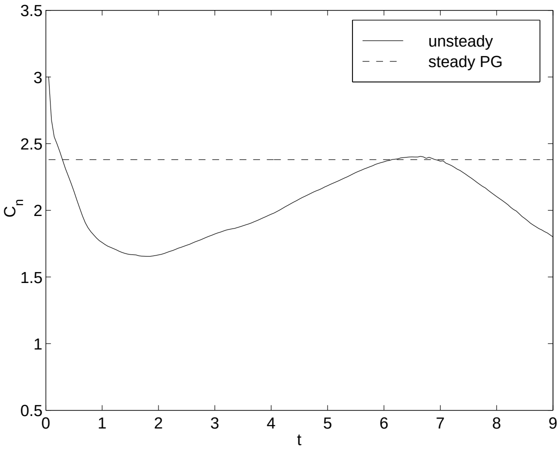


Figure 9.19: Flat plate, $\alpha = 20^o$. The normal force coefficient $C_n$ in time, compared with the steady value from the Prandtl-Glauert equation.

The dynamic stall vortex can be followed in the $C_p$-distributions from figure 9.18. Behind the suction peak, a second, more rounded peak appears at $t = 0.75$: the vortex. This vortex peak grows until it shows the shape of the local supersonic area at $t = 5$. Later on, the dead water area is visible as a horizontal line behind the leading edge. At the trailing edge, the secondary vortex can be seen.

The normal force development is remarkably similar to the $C_n$ for $\alpha = 5^o$, figure 9.19. In fact, the influence of the dynamic stall is not visible until $t = 6$. As long as the stall vortex is completely above the plate, it generates the same normal force that the attached flow would have generated. The normal force drops only when the vortex has partially left the flat plate. This behaviour is found too by Mehta and Lavan [14] in their Navier-Stokes calculations on dynamic stall.

## 9.5. Conclusion

Without denying the value of investigating the physics of vortex flow, it must be stated here that the goal of the flat plate problem is to test the adaptive-gridding solver on a realistic, practical problem. Results have been obtained and analysed, so the performance of the solver can be evaluated.

The previous sections show that using the adaptive-gridding method requires more effort from the researcher than the solution method on uniform grids. A refinement criterion must be chosen, validated and calibrated. But when this is done, the adaptive gridding provides useful and reliable results at low computational costs. In the representative problems, the CPU time is reduced with an estimated factor 80. And although the flow problem was different from the previous test problems, no changes to the solver were needed, apart from the new refinement criterion.

In the test, problems are encountered with grid dependency. However, it is shown that these problems are not caused by adaptive gridding, but by the choice of the boundary conditions and the shape of the flat plate. The grid-dependent flow behaviour is not even influenced by the adaptive-gridding method.

Concluding, the adaptive-gridding solver can handle practical problems and it has proven to be a useful tool for flow analysis.

# Chapter 10
# Conclusions

In chapter 1, the development of a fast and accurate adaptive-gridding solver is proposed, that can be used for the solution of unsteady hyperbolic conservation laws. The solver must be modular, to enable easy adaptation and exchange of all parts. How these goals are reached, is described in the first section of this conclusion.

As the method developed here is quite general, it can be used in, or extended to, a large number of fields and applications. The reduction of computational costs achieved makes the method a useful component for many types of conservation law solvers. Suggestions for the use of the method are given in section 2.

## 10.1. The flow solver

The flow solver that is developed here consists of four parts:

*1. Adaptive grid algorithm* An adaptive grid algorithm is developed that refines a coarse, rectangular grid. The basic cells are split in four when the solution requires this, the smaller cells can be split again etc. The time steps for these smaller cells are smaller too, so the algorithm is adaptive in space and time. Refinement or unrefinement of a cell is allowed after each of its own time steps, therefore the grid can be adapted to the solution several times per coarse time step. This makes it possible to follow travelling waves with very narrow bands of refined cells, ensuring a low number of refined cells and a fast calculation.

*2. Data structure* The adapted grid is stored in a simple and efficient data structure. With only one 'mother' pointer and four 'neighbour' pointers to other cells, the grid geometry and all grid cell relations can be constructed. The neighbour pointers are arranged so, that any cell around a given cell, even the cells on the corners, can be found in a small number of steps, that does not depend on the total number of cells.

*3. Discretisation* A robust and simple second-order accurate discretisation of the flow equations is developed. A well-proven second-order accurate limited scheme for the fluxes, based on Osher's approximate Riemann solver, is combined with the time integration technique from the two-step Richtmyer scheme. The scheme is suitable for the two-step structure used in the adaptive-gridding algorithm and it is self-starting, which is a major advantage when newly refined cells are advanced in time for the first time. Stability for the scheme is proved for CFL numbers of 0.25 or below, but numerical results show that calculations with CFL numbers of up to 0.5 are usually stable, especially calculations on uniform grids. A system of interpolation, the so-called 'virtual states', is developed for flux calculations at cell faces between cells of different sizes.

*4. Refinement criteria* Three different refinement criteria are studied. For the Euler equations, a simple criterion based on the first spatial derivative of the density proved to be very robust and applicable to a broad range of problems. Another criterion, based on the second spatial derivative of the density, is more nervous but gives very fast calculations when a moderate accuracy is desired on grids that are not refined too many times.

Together, these elements form a simple and robust scheme. Application to three test problems shows that the scheme can handle different problems without fundamental changes. The solver is validated with the first two, standard test problems. A third, new problem, the shedding of vortices from a flat plate, confirms that the solution method is suitable for practical problems.

The performance of the second-order accurate discretisation of the flow equations is compared with the performance of a first-order accurate discretisation. The second-order method makes the computational procedure more complex, but it gives more accurate results with the same computational costs. Therefore, it is worthwhile to combine the second-order discretisation with the adaptive-gridding algorithm, instead of using the first-order discretisation and refining the grid further.

The solver calculates solutions with accuracies comparable with second-order methods on uniform grids, but with computational costs that are often at least five times lower. The increase in computing time for refinement and unrefinement is much smaller than the decrease in computing time due to the fast calculation of the flow in the coarse cells.

The method has escaped ‘conservation of difficulty’, it is fundamentally faster than methods on uniform grids.

## 10.2. Possibilities for further research

The present solver is specifically written for research purposes, with its rectangular grids and simple geometries. But the principles used are generally applicable and the method can be extended and modified in several ways. Some suggestions are given here.

*Further development of the 2D solver* In previous work at CWI [13], a method is described to solve the steady Euler equations on arbitrary structured grids. This technique can be combined with the present solver to yield an unsteady Euler solver that can handle arbitrary 2D geometries. Such a solver could be a valuable tool for flow analysis and aerodynamic design.

*Extension to 3D problems* As 3D problems are usually bigger than 2D problems, an efficient calculation of the flow solution is very important. And for 3D, the computational costs for a problem change with the fourth power of the grid cell sizes. Therefore, adaptive gridding can be very useful for these problems. The present routines for flux calculation, time integration and grid refinement can be extended to 3D with very little effort. Only the data structure needs modification, the system of ‘neighbour’ pointers must be changed for 3D cells and the cell search routines must be adapted.

*Implementation of other conservation laws* All other hyperbolic conservation laws can be discretised like the Euler equations are discretised here, with a finite-volume discretisation and a Godunov-type flux function. And in the present solver, the *only* parts that are specific for the Euler equations are the Osher scheme, the Osher-based boundary conditions and the refinement criteria. Therefore, simply changing the Riemann solver and the boundary conditions is enough to make the adaptive-gridding method suitable for the solution of any (system of) hyperbolic conservation law(s).

*Extension to Navier-Stokes* The Navier-Stokes equations, the governing equations for viscous fluid flow, are of mixed hyperbolic-elliptic type and therefore, they cannot be implemented in the adaptive-gridding method without major changes. For example, different types of virtual cells may be needed to calculate the viscous fluxes. And the flow in boundary layers, with its strong gradients in one direction only, must probably be solved on a grid that is refined in only one direction. Yet, a Navier-Stokes solver with adaptive gridding could save much calculation time in flows where a very fine flow structure exists in a small part of the solution only, like the direct numerical simulation of turbulent flow. It is therefore worthwhile to develop such a solver.

*Refinement criteria* The present study of refinement criteria shows that simple gradient criteria give the best results, because they are least sensitive to small disturbances in the solution. But these criteria usually need many cells to obtain an accurate solution and their upper and lower limits must be set properly for each new problem. Therefore, it is useful to search for a well-functioning error-based refinement criterion that needs no tuning. Furthermore, the refinement criteria were all tested here on solutions with discontinuities. It is interesting to test them also with smooth flows, to see if this improves the performance of the truncation-error based refinement criteria.

At this moment, it is only possible to tune the gradient criteria by comparing solutions on adapted grids with solutions on uniform grids for a representative but small problem, like in chapter 9. Therefore, it is very useful to develop guidelines for the proper settings of the refinement criteria, probably based on experience with different flow problems. Such guidelines would enable the researcher to solve flow problems more quickly.

# References

1. J.D. Anderson. *Fundamentals of Aerodynamics*. McGraw-Hill, 1991.

2. M.J. Berger and P. Colella. *Local Adaptive Mesh Refinement for Shock Hydrodynamics.* J. Comp. Phys. **82** (1989), pp. 64–84.

3. M.J. Berger and J. Oliger. *Adaptive Mesh Refinement for Hyperbolic Partial Differential Equations.* J. Comp. Phys. **53** (1984), pp. 484–512.

4. E.H. van Brummelen and B. Koren. *A Pressure-Invariant Conservative Godunov-Type Method for Barotropic Two-Fluid Flows*. Report MAS-R0221, Centre for Mathematics and Computer Science, Amsterdam, 2002.

5. A.F. Emery. *An Evaluation of Several Differencing Methods for Inviscid Fluid Flow Problems.* J. Comp. Phys. **2** (1968), pp. 306–331.

6. S.K. Godunov. Разностный метод численного расчета разрывных решений уравнений гидродинамики. *(Finite Difference Method for Numerical Computation of Discontinuous Solutions of the Equations of Fluid Dynamics).* Mat. Sbornik **47** (1959), pp. 271–306 (in Russian).

7. J.M.R. Graham. *The lift on an aerofoil in starting flow*. J. Fluid Mechanics **133** (1983), pp. 413–425.

8. P.W. Hemker, private communication, 2002.

9. P.W. Hemker and S.P. Spekreijse. *Multiple Grid and Osher's Scheme for the Efficient Solution of the Steady Euler Equations.* Applied Numerical Mathematics **2** (1986), pp. 475–493.

10. W. Hundsdorfer, B. Koren, M. van Loon and J.G. Verwer. *A Positive Finite-Difference Advection Scheme.* J. Comp. Phys. **117** (1995), pp. 35–46.

11. B. Koren. *Multigrid and defect correction for the steady Navier-Stokes equations, Application to aerodynamics*. CWI Tract 74, Centre for Mathematics and Computer Science, Amsterdam, 1991.

12. B. van Leer, private communication, 2002.

13. H.T.M. van der Maarel. *A Local Grid Refinement Method for the Euler Equations*. PhD Thesis, University of Amsterdam, Amsterdam, 1993.

14. U.B. Mehta and Z. Lavan. *Starting vortex, separation bubbles and stall: a numerical study of laminar unsteady flow around an airfoil*. J. Fluid Mechanics **67**, part 2 (1975), pp. 227–256.

15. S. Osher and F. Solomon. *Upwind Difference Schemes for Hyperbolic Conservation laws*. Math. Comput. **38** (1982), pp. 339–374.

16. S.P. Spekreijse. *Multigrid Solution of the Steady Euler Equations*. CWI Tract 46, Centre for Mathematics and Computer Science, Amsterdam, 1988.

17. P.K. Sweby. *High Resolution Schemes Using Flux Limiters for Hyperbolic Conservation Laws.* Siam J. Num. Anal. **21** (1984), pp. 995–1011.

18. E.F. Toro. *Riemann Solvers and Numerical Methods for Fluid Dynamics, A Practical Introduction*. Springer-Verlag, Berlin, 1997.

19. R.A. Trompert. *Local Uniform Grid Refinement for Time-dependent Partial Differential Equations*. PhD Thesis, University of Amsterdam, Amsterdam, 1994.

20. M. Van Dyke. *An Album of Fluid Motion*. The Parabolic Press, Stanford CA, 1982

21. J. Wackers. *A Finite-Volume Solver for the 1D Euler Equations*. 4th year research task, Delft University of Technology, Faculty of Aerospace Engineering, 2001.

22. J. Wackers. *Computer Codes for the Solution of the 2D Unsteady Euler Equations using Adaptive Gridding*. Report, Delft University of Technology, Faculty of Aerospace Engineering, 2002.

23. H. Werlé. *Le Tunnel Hydrodynamique au Service de la Recherche Aérospatiale. (The Water Tunnel in Service for Aerospace Research.)* Publ. no. 156, ONERA, France (in French).

24. P.R. Woodward and P. Colella. *The Numerical Simulation of Two-Dimensional Fluid Flow with Strong Shocks*. J. Comp. Phys. **54** (1984), pp. 115–173.

# Samenvatting

## (in Dutch)

Lokale roosterverfijning is een techniek om het numeriek oplossen van partiële differentiaalvergelijkingen te versnellen, door deze berekeningen te starten op een grof basisrooster en dit rooster alleen daar te verfijnen waar de oplossing dat vereist, bijv. in gebieden met grote gradiënten. Deze techniek is al veelvuldig toegepast, voor zowel stationaire als instationaire problemen. Hier wordt een eenvoudige en efficiënte techniek voor roosteraanpassing gepresenteerd voor het oplossen van stelsels van 2D instationaire hyperbolische behoudswetten. De techniek wordt toegepast op de Eulervergelijkingen uit de aërodynamica. Naar verwachting is uitbreiding naar andere behoudswetten of uitbreiding naar 3D eenvoudig.

Een oplossingsalgoritme wordt gepresenteerd dat een rechthoekig basisrooster verfijnt door grove cellen in vieren te splitsen, zo vaak als dat nodig is, en naderhand weer samen te voegen. De kleinere cellen hebben ook een kortere tijdstap, dus het rooster wordt aangepast in ruimte en tijd. Het rooster wordt meerdere keren per grove tijdstap aangepast aan de oplossing, waardoor het totale aantal cellen laag blijft en de berekening snel verloopt.

Het rooster wordt opgeslagen in een eenvoudige datastructuur. Alle celgegevens worden opgeslagen in 1D arrays en de geometrie van het rooster wordt vastgelegd door vijf verwijzingen naar andere cellen per cel: een ‘moeder’-verwijzing naar de cel waaruit de cel is afgesplitst en vier ‘buur’-verwijzingen. Deze laatste zijn zo gekozen dat alle cellen om de betreffende cel snel gevonden kunnen worden.

Om te bepalen waar het rooster wordt verfijnd, wordt een verfijningscriterium gebruikt. Drie verschillende verfijningscriteria worden bestudeerd: één gebaseerd op de eerste ruimtelijke afgeleide van de dichtheid, één op de tweede ruimtelijke afgeleide van de dichtheid en één op een afschatting van de lokale afbreekfout, vergelijkbaar met Richardson-extrapolatie. Met name het eerste-afgeleide $\rho$ criterium geeft goede resultaten.

Het algoritme wordt gecombineerd met een simpele eerste-orde nauwkeurige discretisatie van de Eulervergelijkingen, gebaseerd op Oshers fluxfunctie en vervolgens getest.

Een tweede-orde nauwkeurige discretisatie van de Eulervergelijkingen wordt gepresenteerd die een tweede-orde gelimiteerde fluxdiscretisatie combineert met de tijdsafgeleiden van het Richtmyer-schema. Dit schema kan eenvoudig gecombineerd worden met het roosteraanpassingsalgoritme. Stabiliteit wordt bewezen voor CFL-getallen lager dan 0,25. Voor cellen met verschillende groottes zijn bij deze discretisatie een aantal interpolatietechnieken ontwikkeld, waaronder het gebruik van virtuele cellen voor de berekening van fluxen.

Het schema wordt getest op twee standaard testproblemen, het 1D Sod-probleem en het voorwaartse-drempelprobleem, bekend uit het werk van Woodward en Colella. Resultaten tonen aan dat het tweede-orde schema efficiënter is dan het eerste-orde schema. Een nauwkeurigheid, vergelijkbaar met oplossingen op uniforme roosters, wordt bereikt, maar met minimaal vijf keer lagere berekeningskosten. Resultaten van een laatste testprobleem, het ontstaan van wervels rond een vlakke plaat die plotseling in beweging wordt gebracht, bevestigen dat de methode bruikbaar is voor verschillende types stromingen en dat hij in de praktijk uitstekend gebruikt kan worden voor analyse van instationaire stromingen.