A Moving Mesh Finite Volume Solver for Macroscopic Traffic Flow Models

A. van Dam

May 2002

Preface

This report is my Master's Thesis in Computational Science and describes the results of the research and work done during an internship at TNO Inro, Delft, The Netherlands, conducted from September 2001 to May 2002.

Main research interests during this project were the properties of a moving mesh solver and its applicability to general systems of hyperbolic partial differential equations. As a part of this, two macroscopic traffic flow models have been investigated and a flexible solver and traffic flow simulator has been developed.

The project was supervised by Ir. Chris Tampère of TNO Inro, and Dr. Paul Zegeling of the Department of Mathematics of Utrecht University. I am very grateful for their support and expertise, in particular the degree of freedom they gave me in doing my own research.

Special thanks also to Dr. Ir. Serge Hoogendoorn of the Delft University of Technology for his support and cooperation in traffic simulations, and to Dr. Huazhong Tang of Peking University and Prof. Dr. Tao Tang of the Hong Kong Baptist University who kindly provided me their articles on moving meshes and program source code of their moving mesh solver.

Arthur van Dam
May 2002

Abstract

This report presents the research done as graduation project for Computational Science at the Department of Mathematics of Utrecht University. The research was conducted within the project "Modelling of traffic flow and driver behaviour in congestion for the analysis of Automated Driver Assistance Systems", which is part of a joint research program of the Netherlands TRAIL Research School and the Dutch Organization of Applied Science TNO. One of the main research interests in this project is to develop insight into the mechanisms of individual and collective driving behaviour in congested conditions.

The research presented in this report consists of three parts. The report presents an introduction to traffic flow models, especially macroscopic formulations as nonlinear systems of Partial Differential Equations. A moving mesh finite volume solver for PDE systems of this kind is studied, implemented and investigated. For one specific model, traffic flow simulations are performed in various scenarios.

Traffic Flow Models

Modelling traffic flow can be done at various levels of detail: submicroscopic, microscopic, mesoscopic and macroscopic. (Sub-)microscopic models distinguish the individual drivers and describe their behaviour in great detail. Macroscopic models describe the traffic flow by continuous aggregate functions like average density, velocity and flow in the space-time domain. The dynamics of traffic flow is modelled by a nonlinear system of two or three PDEs. Mesoscopic models form a huge 'transition' class between micro- an macroscopic models.

Kerner [22, 23, 24] has presented a model of two PDEs, describing the dynamics of traffic density r and average velocity V. Hoogendoorn [15, 16, 17, 18, 19, 20, 21] has done much research in the late nineties on 2- and 3-PDE formulations, including more realistic description of macroscopic flow dynamics by introducing multiple userclasses, like person cars and trucks.

Moving Mesh Solver

In many fields, the technique of adaptive meshing has become increasingly important in the last decade. Adaptive meshing is a technique to concentrate meshpoints in a discretized DE solver at the locations where they are needed the most. Often, this means placing the meshpoints near regions where the solution has a large gradient. To use a time-dependent adaptive mesh, i.e. a moving mesh, monitor functions are used to automatically 'monitor' the importance of the various parts of the domain, by assigning a 'weight'-value to each location.

Using moving meshes in combination with an advanced finite volume PDE solver can yield serious gain in accuracy at relatively low extra costs. The choice of a suitable monitor function is very important.

Implementing a Traffic Flow Simulator

Using the formulations from macroscopic traffic flow models and the solving techniques like moving meshes and a finite volume PDE solver, a powerful traffic flow simulator can be implemented in a MATLAB program. By using a model- and problem-based description of the traffic flow PDEs to be used, a flexible simulator is the result.

By entering the desired traffic flow PDEs and domain specifications as separate modules, the simulator is a powerful tool in investigating all kinds of traffic flow models and during the development of new macroscopic formulations.

Traffic Flow Characteristics

Although it has been criticised for not always being so realistic, Kerners model shows various interesting characteristics in the formation of traffic jams. These can often be intuitively explained using fundamental diagrams.

The effectiveness of velocity management systems can also be investigated by running macroscopic traffic flow simulations, albeit in a qualitative way.

Outline for further Research

Based upon this research and its conclusions, some directions for future research can be given:

- Since the solver is aimed at a rather general class of PDEs, no problem specific tricks can be implemented, whereas these tricks can generally yield faster runs and sometimes more accurate results. Possibly, the PDE-solving part of the moving mesh solver could be extended with some additional PDE solvers that are slightly more aimed at certain more specific PDEs from the general class of nonlinear hyperbolic PDEs. For each problem to be solved, the most suitable solver is chosen.
- The monitor function that is finally used in this research is a quite powerful one, but maybe even better ones exist. This could make the mesh-moving even more efficient and more accurate.
- For the investigated models, more advanced simulations could be performed, like multi-lane formulations and a more detailed study of multiple userclass characteristics.
- Models that could not directly be solved with the eventual solver should be further investigated. If it is possible to identify the term(s) that cause the instability with the solver, measures could be provided to overcome the problems.
- In case the developed software in TRAFLOWPACK is used often, a graphical shell could be built around it, to make it more userfriendly.

Korte Samenvatting

Dit rapport beschrijft de resultaten van onderzoek, gedaan als afstudeerproject voor Computational Science bij de vakgroep Wiskunde van de Universiteit Utrecht. Het onderzoek is uitgevoerd binnen het project "Modellering van verkeersstroom en bestuurdersgedrag bij congestie (voor de analyse van automatische bestuurder ondersteunende systemen", wat onderdeel is van een gezamenlijk onderzoeksprogramma van de TRAIL Onderzoeksschool Nederland en de Nederlands Organisatie voor toegepast-natuurwetenschappelijk onderzoek TNO. Een van de hoofddoelen tijdens dit project is inzicht te krijgen in de mechanismes van individueel en collectief verkeersgedrag in congestie.

Het onderzoek in dit rapport bestaat uit drie delen. Het rapport geeft een korte introductie in verkeersstroommodellen in het algemeen, en gaat vooral in op de macroscopische formulering van verkeersstroom als een niet-lineair stelsel van partiële differentiaalvergelijkingen. Een eindige volume techniek in combinatie met een bewegend rooster, geschikt voor PDV's van dit type wordt bestudeerd, geïmplementeerd en onderzocht. Voor één specifiek verkeersstroom model worden simulaties van diverse verkeersscenario's uitgevoerd.

Verkeersstroom Modellen

Het modelleren van verkeersstroom kan op verschillende niveaus geschieden, gebaseerd op de mate van detail: submicroscopisch, microscopisch, mesoscopisch en macroscopisch. (Sub-)microscopische modellen maken onderscheid tussen de individuele bestuurders en beschrijven hun gedrag in grote mate van detail. Macroscopische modellen beschrijven de verkeersstroom aan de hand van continue, geaggregeerde functies, zoals gemiddelde dichtheid, snelheid en stroomintensiteit in het plaats-tijd domein. De dynamiek van de verkeersstroom wordt gemodelleerd door een niet-lineair stelsel van twee of drie PDV's. Mesoscopische modellen vormen een grote overgangs-klasse tussen micro- en macroscopische modellen.

Kerner [22, 23, 24] stelt een model van twee PDV's voor, waarin de verandering van verkeersdichtheid r en gemiddelde snelheid V wordt beschreven. Hoogendoorn [15, 16, 17, 18, 19, 20, 21] heeft eind jaren '90 veel onderzoek gedaan naar 2- en 3-PDV formuleringen, waaronder realistischere beschrijving van macroscopische verkeersstroom door verschillende gebruikersklassen te onderscheiden, zoals personenauto's en vrachtwagens.

Bewegende Roosters

In diverse gebieden zijn adaptieve-rooster-technieken de laatste jaren steeds belangrijker geworden. Het gebruik van adaptieve roosters is een techniek, waarbij de roosterpunten in een gediscretiseerd domein geconcentreerd worden rond locaties waar ze het meest nodig zijn. Vaak komt dit neer op het plaatsen van veel roosterpunten op locaties waar de oplossingsgradiënt groot is. Bij het gebruik van een tijdsafhankelijk rooster, oftewel een bewegend rooster, worden monitor-functies gebruikt om de belangrijke lokaties in het domein te lokaliseren. Hierbij kennen monitor-functies een soort gewicht toe aan de locaties in het domein.

Het gebruik van bewegende roosters in combinatie met een geavanceerde eindige volume PDV oplostechniek kan flinke winst in nauwkeurigheid opleveren, tegen relatief lage extra kosten. De keus van een geschikte monitor-functie is hierbij wel erg belangrijk.

Implementatie van een Verkeersstroom Simulator

Gebruik makend van de formuleringen uit macroscopische verkeersstroommodellen, een eindige volume techniek en bewegende roosters, kan een goede verkeersstroom-simulator geïmplementeerd worden als MATLAB programma. Door een model- en probleem-gebaseerde beschrijving van de op te lossen PDV's ontstaat een flexibele simulator.

Door de gewenste verkeersstroom-PDV's en domein-specificaties als losse modules te programmeren, is de simulator een krachtig hulpmiddel bij het onderzoek naar allerlei verschillende verkeersstroommodellen en bij het ontwikkelen van nieuwe macroscopische formuleringen.

Verkeersstroom Eigenschappen

Alhoewel Kerner's model vaak bekritiseerd is om de beperkte realiteits-waarde, bevat het toch een aantal interessante eigenschappen en verschijnselen bij het ontstaan van files. Vaak kunnen deze goed verklaard worden met behulp van een fundamenteel diagram.

De effectiviteit van snelheidsregulerende systemen kan ook onderzocht worden door macroscopische verkeersstroom-simulaties, al is dat in dit geval vooral kwalitatief.

Aanwijzingen voor verder Onderzoek

Uit dit onderzoek en de bijbehorende conclusies kunnen een aantal aanwijzingen voor verder onderzoek gegeven worden:

- Aangezien de oplostechniek gericht is op een behoorlijk algemene klasse van PDV's, kunnen geen probleemspecifieke trucjes gebruikt worden in het oplosproces. Dergelijke trucjes kunnen echter wel vaak tijdswinst of zelfs hogere nauwkeurigheid opleveren. Eventueel zou het PDV-oplos-gedeelte uitgebreid kunnen worden met een aantal alternatieve oplostechnieken. Per probleem wordt dan de meest geschikte oplostechniek gekozen.

- De monitor-functie waar uiteindelijk mee gewerkt wordt levert goede resultaten, maar misschien kunnen er nog betere gevonden worden. Het bewegende rooster zou dan nog efficiënter en nauwkeuriger kunnen worden.

- Voor de onderzochte verkeersmodellen zouden geavanceerdere simulaties uitgevoerd kunnen worden, waarbij bijvoorbeeld meerdere rijstroken en gebruikersklassen worden meegenomen.

- Modellen die nu niet direct opgelost kunnen worden met de huidige oplostechniek, zouden verder onderzocht kunnen worden. Zodra de term(en) die de instabiliteit veroorzaken, aangewezen kunnen worden, kunnen maatregelen genomen worden om de problemen op te lossen.

- Indien de ontwikkelde software in TRAFLOW$_{\mathsf{PACK}}$ vaak gebruikt wordt, zou er een grafische schil omheen gebouwd kunnen worden, voor groter gebruikersgemak.

Contents

Preface iii

Abstract v

Korte Samenvatting vii

Contents xi

List of Figures xiv

1 Introduction 1
- 1.1 Research goals . . . 1
- 1.2 Thesis structure . . . 2
  - 1.2.1 Research approach . . . 2
  - 1.2.2 Conventions in this document . . . 2

2 Traffic flow theory 3
- 2.1 Introduction: traffic flow models . . . 3
  - 2.1.1 Submicroscopic traffic models . . . 3
  - 2.1.2 microscopic traffic models . . . 4
  - 2.1.3 Mesocopic traffic models . . . 4
  - 2.1.4 Macroscopic traffic models . . . 5
- 2.2 the Kerner and Konhäuser model . . . 5
  - 2.2.1 Reactions to Kerners model . . . 6
- 2.3 Hoogendoorns MultiClass-model . . . 8
  - 2.3.1 Userclasses . . . 8
  - 2.3.2 Phase-Space-Density . . . 8
  - 2.3.3 Method of Moments . . . 8
  - 2.3.4 Conservative 2-PDE formulation of Multiclass Traffic Flow . . . 9
- 2.4 The fundamental diagram . . . 10
- 2.5 More model improvements . . . 11

3 Moving mesh methods 13
- 3.1 Mesh generation . . . 14
  - 3.1.1 Monitor functions . . . 14
  - 3.1.2 Mesh map . . . 15
  - 3.1.3 1D case . . . 15
  - 3.1.4 Example mapping . . . 16
- 3.2 1D algorithm . . . 17
  - 3.2.1 Mesh-redistribution . . . 18
  - 3.2.2 Solution updating on new mesh . . . 18
  - 3.2.3 Solving the PDE on new mesh . . . 18
- 3.3 Program enhancements and MATLAB implementation . . . 19

3.3.1 Spatial boundaries . . . 19
3.3.2 Effects of enhancements . . . 20
3.3.3 Runge-Kutta time-integration . . . 21
3.3.4 Stepsize control and the CFL-condition . . . 21
3.3.5 Moving the mesh . . . 22
3.3.6 Computational vs. physical gradients . . . 23
3.3.7 Smoothing of the monitor function . . . 24
3.3.8 A system of PDEs . . . 26
3.3.9 Non-zero source term . . . 27
3.4 Error estimates and parameter finetuning . . . 27
3.4.1 Error estimates . . . 27
3.4.2 Errors at a shock . . . 28
3.4.3 Choosing the adaptivity parameter . . . 29
3.4.4 Choosing a sensible monitor-function . . . 33
3.5 Example problems . . . 33
3.5.1 Burgers' equation . . . 33
3.5.2 The shock tube or Riemann problem . . . 34

4 Traffic flow simulation 43
4.1 Problem settings . . . 43
4.1.1 Ringroad . . . 43
4.2 Simulations with Kerners model . . . 44
4.2.1 Metastable behaviour . . . 44
Variable Speed Control . . . 52
4.3 Simulation with Hoogendoorn . . . 53
4.3.1 Instability . . . 53
4.3.2 McCormack solver . . . 55
4.3.3 Conclusion . . . 56

5 TraFlow$_{\text{PACK}}$ software documentation 57
5.1 MMFVPDES : the numerical PDE solver . . . 57
5.1.1 Commandline syntax . . . 57
5.1.2 Settingsfile syntax . . . 59
5.1.3 Model- and problem-functions syntax . . . 59
5.1.4 Command window output . . . 62
5.1.5 visualization output . . . 63
5.1.6 Input-/Output routines and error checking . . . 66
5.2 TraFlow : traffic flow simulation tool . . . 67
5.2.1 Commandline syntax . . . 67
5.2.2 Variables, userclasses and roadway lanes . . . 68
5.2.3 Model-settingsfile syntax . . . 69
5.2.4 Problem-settingsfile syntax . . . 69
5.2.5 Input-/Output routines . . . 70
5.2.6 Visualization routines . . . 70

6 Conclusions and Recommendations 73
6.1 Conclusion . . . 73
6.2 Improvements and further research . . . 75

References 77

Glossary 81

A Program Code **85**
A.1 TraFlow$_{\mathsf{PACK}}$: MMFVPDES and TraFlow 85

List of Figures

2.1 Speed-density relation by Helbing and Kerner compared to empirical traffic data obtained from the dutch highway A9. 7
2.2 Fundamental diagram for Kerners model 11
2.3 A 10km road with a congested and uncongested region (left). The accompanying fundamental diagram is shown right. 11

3.1 Solution and coordinate-mapping in case of a shock and discontinuity. 16
3.2 The *discretized* spatial domain with 'beyond-boundary'-points. 20
3.3 Fixed uniform mesh and the solution at $t = 0.9$ for the simplest program (upwind scheme, fixed mesh). 21
3.4 Fixed uniform mesh and the solution at $t = 0.9$. A finite volume approach and Runge-Kutta time integrator is used, in combination with stepsize control. 22
3.5 Mesh-movement through time and the solution at $t = 0.9$. In addition to the previous enhancements, the mesh is now non-uniform ($\alpha = 0.3$). 23
3.6 Effect of repeated monitorsmoothing. Left: Basic smoother and resulting 4-times smoother; right: Effect of this smoother on monitor-values. 25
3.7 Burgers' problem, top row: mesh-movement, bottom row: detail of solution near shock. Left: physical gradients, middle: computational gradients, right: computational gradients using 4 smoothings. 26
3.8 Finding the location of a shock by the x-coordinate of its half-height. 29
3.9 The optimal values for $\overline{\omega}$ for various number of shocks n. 32
3.10 Schematic representation of Sod's shock tube, two top-views of the tube, and the time-trajectories of the various waves. 35
3.11 Variation in flow quantities at $t = 0.2$. 37
3.12 Density-solution of the shock-tube problem at time $t = 0$, $t = 0.05$ and $t = 0.2$ respectively. 38
3.13 Results for the shocktube problem with various monitors. Left column shows the solution at $t = 0.2$, right column shows the mesh movement. All experiments use $N = 103$, and the four monitor functions are, from top to bottom: uniform mesh, BM monitor, ξ_{norm} monitor and Stockie monitor. 39
3.14 Values of BM monitor through time for a traffic flow problem. 41

4.1 Evolution in Kerners model with an initial uniform distribution of 38 veh./km and a small perturbation at 3.7km. Left column shows density at 4 distinct times, right column shows the accompanying velocity. 45
4.2 The evolution of a perturbation $\Delta r_0 = 1$veh./km at three density base levels r_0. Left column shows the traffic density, right column shows the velocity. 47
4.3 The x-t-trajectories and the underlying flow (veh./s) for the simulation with $r_0 =$ 38veh./km. (left) and $r_0 = 60$veh./km. (right) 48
4.4 Evolution in Kerners model with an initial uniform distribution of 60 veh./km (left) and 72 veh./km (right) and a small perturbation at 10km. The 4 diagrams show respectively $r(x,t)$ in 3D, $r(x)$ at $t = 1000$s, the $V(r)$ fundamental diagram and a detail of this diagram. 51

4.5 Dipole effect in Kerners model for the perturbation of $r_0 = 60$. Left: fundamental $m(r)$ diagram, right: x-t trajectories and flow. 52
4.6 Schematic representation of a ringroad with a blockade in the middle. 52
4.7 Evolution of a blockade. Left: without any traffic management, right: with VMS enabled. 53
4.8 Instability of MMFVPDES with Hoogendoorns 2-PDE model. An initial smooth solution blows up for increasing time ($t = 0, 30, 60, 120$s). 54
4.9 Effect of emerging shock on McCormack solution at $t = 0, 120, 192$ and 250s. . . . 55

5.1 2D visualization of all three model variables in the Euler-shocktube problem at sampled time t_d. 64
5.2 Mesh map $x(\xi)$ in the Euler-shocktube problem at sampled time t_d. 64
5.3 Mesh movement in the Euler-shocktube problem. 65
5.4 3D visualization of ρ in the Euler-shocktube problem. 65
5.5 Space-time trajectories of 18 imaginary cars, and the average flow. 71
5.6 Real-world plot of the density at two roadway-lanes. 71

Chapter 1

Introduction

The study of macroscopic traffic flow is a science that can very well make use of computer simulations in investigating the properties of a certain model-based description of traffic flow.
On the other hand, traffic flow is an interesting application of nonlinear hyperbolic partial differential equations (PDEs). Various numerical, mathematical techniques could be implemented, and their applicability tested on these traffic flow models.
The research described in this thesis can be placed on the crossing of these two expertises, and the research goals can be formulated likewise.

1.1 Research goals

Since the field of research is twofold, the research goals are placed into two classes.

Traffic flow simulation

First, the field of traffic flow modelling, especially the macroscopic type, should be investigated to obtain some basic knowledge of the ideas and formulations that currently exist among traffic scientists.

Second, some simulation tool should be implemented in order to actually perform macroscopic traffic flow simulations. The software should not be aimed at one specific model, but should be easy configurable, both in model specification and simulations settings. The resulting simulation data should be easily storeable and retrievable, and should be visualized in an intuitive way as well. In this form the software could be employed in experimental research during the development of new traffic flow models.

Third, some actual traffic flow simulations should be performed. The simulations should be based on a macroscopic traffic flow model that is more thoroughly investigated. The properties of this specific model and traffic evolution in certain special scenarios should be investigated through simulations.

Numerical solving techniques

First of all, some mathematical solver should be implemented, that is able to handle a general class of PDEs among which the macroscopic traffic flow PDEs. The input of PDE systems into the software by the user should be easy, and module-based so that no internal source-code of the solver itself has to be changed each time. The solver itself should in essence have nothing to do with specific traffic flow equations. This way, the solver might even be suitable for problems from other fields that involve PDEs.

Second, prior to any traffic flow simulations, the mathematical solver has to be investigated on its accuracy and calibration. The solver uses a moving mesh technique to obtain a fine spatial discretization in areas of high importance. The effectiveness of some monitor functions that steer

the mesh movement should be investigated. Besides, the gain in accuracy should be related to the loss in performance speed, to find out whether the application of these special techniques is worth the effort.

1.2 Thesis structure

1.2.1 Research approach

The research starts with the investigation of the various levels of traffic flow modelling. Chapter 2 describes the results of this work and thus provides a basic overview of traffic flow modelling, with many references to more detailed resources. In somewhat more detail, two macroscopic models by Kerner and by Hoogendoorn are described.

With the obtained knowledge on the form of macroscopic traffic flow PDEs, a general system of hyperbolic PDEs with nonzero source term is taken as the basis for further numerical research. Chapter 3 describes the techniques used for solving systems of this kind. In particular a moving mesh technique is introduced and thoroughly investigated. Accuracy and calibration of monitor functions is done by solving Burgers' equation and the shocktube problem for the Euler equations.

The described numerical solver has been implemented in a MATLAB program MMFVPDES , and on top of that a traffic simulation shell has been built: TRAFLOW . Both programs have been integrated in a package named TRAFLOW$_{\text{PACK}}$, which is freely available to the public. Chapter 5 describes the possibilities of this software. It shows how the software should be configured and run, and possibly extended. It also shows the various visualization tools and import and export functionality.

With the developed software and the two described macroscopic traffic flow models, a series of simulations is performed. The evolution of traffic from a certain initial distribution is considered, the occurence of traffic jams and the effect of speed limiting systems.

The thesis concludes with an overview of the obtained results, and provides hints for improvements and possible further research.

1.2.2 Conventions in this document

Throughout the entire report, several terms and concepts are in *slanted* font, whenever they are mentioned for the first time. This means that they are further explained in the glossary at page 81. Further occurrences of the same term are displayed in normal font.

Any numbered references between parentheses without any notion on the type of reference can be considered equation-numbers, e.g. (3.8) refers to the moving mesh PDE.

Numbered references between square brackets refer to the bibliographic references at page 77, e.g. [27] refers to Lighthill and Whithams paper.

Chapter 2

Traffic flow theory

Lighthill and Whitham are generally accepted to be the founders of traffic flow theory. In 1955 they presented a *macroscopic* model describing traffic flow based on the analogy of vehicles in a traffic flow with particles in a fluid [27]. Ever since, many improvements and different modelling approaches have been presented.

First, a short overview of the history of traffic flow research is presented. Four basic traffic flow model-types are discussed.

Second, two macroscopic models are discussed in more detail. The first is a model by Kerner and Konhäuser. The second is the more advanced macroscopic *MultiClass*-model for an *aggregate-lane* road by Hoogendoorn. These two models form the basis of further implementation and experiments.

2.1 Introduction: traffic flow models

In the following, model categorization is based on the level of detail of a model. Using this approach, four basic model classes can be distinguished:

- *submicroscopic* models
- *microscopic* models
- *mesoscopic* models
- macroscopic models

2.1.1 Submicroscopic traffic models

Submicroscopic models describe traffic at a very high level of detail. The individual vehicles are distinguished, as well as their interaction with each other. Even the various driving concepts like shifting gears and detailed acceleration- and deceleration-behaviour are considered. Models of this kind are used in applications like *Advanced Driver Assistance (ADA)* and *Fully Automated Vehicles (FAV)* (together: *Automated Vehicles Guidance (AVG)*), where vehicles drive partially or fully autonomous. In these applications high accuracy of the model variables is required, since the safety-margins are relatively small.

An example of a submicroscopic model is *SmartAHS* [4, 29]. SmartAHS is a software environment which is able to simulate a large number of traffic processes and events (e.g. speed-regulation, lanekeeping) at a high level of detail. Another example of such a package is PELOPS [28, 40]. Submicroscopic models like these typically consider systems of about 100 vehicles.

At the Netherlands Organisation for Applied Scientific Research (TNO), a package named MIXIC [2] has been developed, which could be classified somewhere between submicroscopic and microscopic. It is mainly used for evaluation of the impact of driver assistance/vehicle control

systems on motorway traffic. The system can deal with a few thousands of vehicles and uses timesteps of order 0.1 second.

2.1.2 microscopic traffic models

Microscopic models do not study the vehicle-operations (like braking) in great detail, but they do consider single vehicles, and the interaction and possible communication between them. The driver's behaviour is generally described by a set of if-then rules. Based on the driver's behaviour and vehicle characteristics, the position, speed and acceleration of each car are computed for each timestep. Therefore these models are also called car-following models.

A well-known commercial microscopic simulation package is PARAMICS [34]. This allows for simulations at a higher level, including complex road-networks, while still producing accurate results at a 'per-vehicle-basis' on *traffic-flow*, congestion, etcetera. Packages like PARAMICS are typically used to simulate road-networks around crossing highways, or near some city, using about 10.000 individual vehicles.

Several other models that consider the individual vehicles are cellular automaton models and particle methods. A more extensive overview of the various microscopic models (as well as meso- and macroscopic models) can be found in [17] and [37].

2.1.3 Mesocopic traffic models

Mesoscopic traffic models actually contain all models that are not micro- or macroscopic. Two main variants can be distinguished:

Micro based on macro Using flow quantities at a macroscopic level (average density/velocity), the dynamics of imaginary individual cars can be determined, for example by integrating velocity over time.

Macro based on micro The traffic dynamics are formulated at a macroscopic level, but the various components of the PDEs, like interaction and lane-changing behavior, are based upon individual driver characteristics.

The first variant is not discussed any further here.

In the second variant, mesoscopic traffic models do not consider the individual vehicles. Instead, more aggregate terms are used, e.g. by using probability density functions. Usually, model variables like traffic density and average velocity are considered. These are affected by various processes like acceleration, interaction between vehicles and lane-changing. The description of these processes *is* based on individual vehicle behaviour. One specific mesoscopic approach is the *gas-kinetic* traffic flow model.

The first variants of the gas-kinetic formulation were proposed in the early sixties by Prigogine and Herman [1]. They basically distinguish three processes:

Convection Vehicles with a speed $v > 0$ flowing into (or out of) a roadsegment $[x, x + \Delta x]$ cause the traffic density to change at that segment.

Acceleration towards desired velocity Whenever possible, vehicles will accelerate towards their desired velocity v^0.

Deceleration due to *interaction* Whenever a vehicles catches up with a slower vehicle, it has to slow down in order to avoid an accident.

The acceleration and deceleration are often placed in one *relaxation term*.
Especially the interaction term has been criticised a lot, for example by Paveri-Fontana [30]. Main points of critique were the vehicular chaos assumption[1], the asymmetric interaction[2] and the

[1] The assumption that "correlation between nearby drivers can be neglected" was argued to be only valid in situations where no vehicles are *platooning*.

[2] Asymmetric interaction means that fast vehicles (fast *vehicle classes*) catch up with slow vehicle classes more often than vice versa.

discrete formulation of deceleration[3]. Lately many work has been done on traffic flow models with multiple roadlanes and multiple userclasses (e.g. trailers, person cars) by Helbing [9, 11, 12] and Hoogendoorn [15, 16, 17, 18, 20, 21].

The model-variables (e.g. ρ) are considered dependent of 4 variables: $\rho(x, t, v, v^0)$, meaning the probability density at location x, time instant t, of vehicles driving with velocity v and having a desired velocity v^0. This special concept of density is generally called *Phase-Space-Density*.

Other mesoscopic models include *headway distribution models* and *cluster models*, but these will not be discussed here.

2.1.4 Macroscopic traffic models

Macroscopic models use one further step of aggregation; they are interested in the overall dynamics of the traffic flow. The independent variables are x and t, this final aggregation step can be summarized as follows:

$$\int\int \text{gaskinetic model } dvdv^0 \quad \longrightarrow \quad \text{macroscopic model}$$

By integrating over the velocity v and the desired velocity v^0, these are no longer independent variables. The aggregate model variables (e.g. density, flow) are considered at a certain location x and time t, for any v and v^0.

Basically, two kinds of macroscopic model formulations can be considered: *primitive* and *conservative*. The macroscopic entities that are usually considered are:

- r, density. Average number of cars per unit length. [m^{-1}] (primitive and conservative).
- V, velocity. Average velocity of cars. [m s^{-1}] (primitive).
- Θ, velocity variance. Average velocity variance of cars. [$\text{m}^2\ \text{s}^{-2}$] (primitive).
- m, flow. Average flow per unit time, $m \stackrel{\text{def}}{=} r \cdot V$. [$\text{s}^{-1}$] (conservative).
- e, energy. Average energy of traffic flow, $e \stackrel{\text{def}}{=} \frac{1}{2} r \left(V^2 + \Theta\right)$. [$\text{m s}^{-2}$] (conservative).

A more detailed list of traffic entities can be found in Chapter 3 of Hoogendoorns thesis [17].

Macro simulations can be performed on domains of dozens of kilometers, depending on the discretization used. The following two sections study two macroscopic models in more detail.

2.2 the Kerner and Konhäuser model

The very first macroscopic traffic flow models were based upon a single PDE, the *continuity equation*:

$$\frac{\partial}{\partial t}[r] + \frac{\partial}{\partial x}[rV] = 0, \tag{2.1}$$

which is also known from general gas or liquid flow systems.

Payne [31] was the first to introduce a second, coupled, PDE for more realistic description of macroscopic traffic flow. Besides the continuity equation an *equation of motion* is used, which in general form reads:

$$\frac{\partial}{\partial t}[V] + V\frac{\partial}{\partial x}[V] = \frac{V^e(r) - V}{\tau} - \frac{1}{r}\frac{\partial}{\partial x}[P(r)] + \frac{\mu}{r}\frac{\partial^2}{\partial x^2}[V]. \tag{2.2}$$

Here, $P(r)$ denotes the so-called *traffic pressure*, often defined as $P \stackrel{\text{def}}{=} r\Theta$, where Θ is the *velocity variance*. The term $\partial_x P \stackrel{\text{def}}{=} \partial P/\partial x$ is often considered as drivers' anticipation to spatially changing

[3] In most models, the deceleration takes place instantly when a vehicle catches up with a slower vehicle.

traffic conditions. Hoogendoorn [17] has shown that $\partial_x P$ is in fact an additional refinement to the convection term $V\partial_x V$. An easy explanation is the following: consider one small roadsegment or cell, which has the same average velocity V at its *upstream* and *downstream* boundary . At the upstream boundary the velocity variance is $\Theta = 0$, meaning that all vehicles have velocity $v = V$. At the downstream boundary, half of the vehicles has velocity $v = 0$, the other half has velocity $v = 2V$, i.e. $\Theta = V^2$. (Note that the average velocity is indeed V.)
Now, at the upstream boundary vehicles keep flowing into the cell with the same velocity $v = V$, so the average velocity remains the same. At the downstream boundary the vehicles with $v = 2V$ flow away out of the cell and only the vehicles with $v = 0$ are left, so the average velocity decreases. This means that the average velocity in the whole cell decreases as well.
If only $V\partial_x V$ had been taken into account, no decrease in velocity would have shown up, since $V\partial_x V = 0$. Including $\partial_x P$, and thus $\partial_x \Theta$, yields a more realistic description of the flow.

The second order derivative $\partial_x^2 V$ is often considered as some *traffic viscosity*, and included in the PDEs as a self-evident property of the flow. This term, however, was mainly useful as a diffusion term preventing shocks in the solution, which are numerically difficult to handle.
Still, the second order derivative also has a quite intuitive meaning. It describes how drivers would tend to accelerate along with the cars ahead of them, by 'predicting' the acceleration behaviour downstream. In this interpretation, it is in fact an *anticipation term*.

Kühne [25] and Kerner and Konhäuser [22] use $P = r \cdot c_0^2$, where c_0^2 is a constant approximation of the velocity variance $\Theta \approx c_0^2$. The viscosity parameter is also set at a constant level $\mu = \mu_0$.

The *equilibrium velocity* V_e incorporates both the *desired velocity* V^0 and the interaction term. A driver thus not always relaxates to his desired velocity, but to his equilibrium velocity, since he takes the current crowdiness into account when choosing a suitable velocity.
Many analytical approximations for V^e have been proposed. The accuracy of this approximation can be measured by comparing it to empirical data.
Kerner *et al.* [22] propose the following relation:

$$V_e(r) \stackrel{\text{def}}{=} V^0 \left[\left[1 + \exp\left(\left(\frac{r - r_i}{r_{\max}} \right) / b \right) \right]^{-1} - d \right], \tag{2.3}$$

where

$$d = \left[1 + \exp\left(\left(\frac{r_{\max} - r_i}{r_{\max}} \right) / b \right) \right]^{-1}. \tag{2.4}$$

Here, $r_{\max}$ is the maximum traffic density that fits when traffic has zero velocity. r_i and b calibrate the interaction behaviour; values for these parameters will be specified in section 4.2.

Figure 2.1 shows Kerners approximation (2.3) and Helbings improved approximation [11] with more advanced interaction description. It is clear that Helbings approximation matches empirical data much better, but Kerners approximation shows at least the same qualitative behaviour.

2.2.1 Reactions to Kerners model

Critique

The supposed constant velocity variance in Kerners model is criticised by many. In an equilibrium traffic flow, the velocity variance is known to decrease for increasing traffic density.

A more general critique, shared by all Payne-type models is the so-called *anisotropy*. Although particles in fluid or gas flow react to stimuli from both upstream and downstream, car drivers don't: they are anisotropic. Car drivers react primarily to traffic conditions downstream. Helbing proposes to use a non-local interaction term. In [11] he also remedies the constant velocity variance by introducing a third PDE for the velocity variance. Hoogendoorn [17] used this as well, but both have concluded after numerous simulations that an analytical approximation of Θ is more useful than an extra PDE for it [12, 21].

Daganzo [8] has also shown that Payne-type models can result in negative speeds at the tail of congested regions. This is caused by the second-order viscosity term (or diffusion).

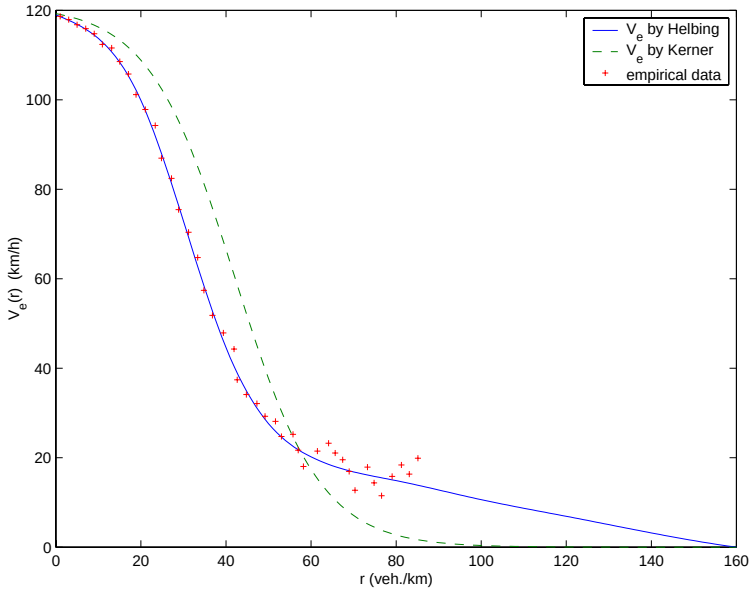

Figure 2.1: Speed-density relation by Helbing and Kerner compared to empirical traffic data obtained from the dutch highway A9.

On the other hand, the viscosity term by Kerner and Konhäuser was the first succesful approach to prevent the formation of shockwaves. Also, the approximation for the traffic pressure by Kühne makes sure that vehicles do not drive into a traffic jam without deceleration, as it did happen in Phillips' model [32].

Phantom jams and stop-start waves

Possibly the most important feature of Kerners model is its ability to represent the formation of *phantom jams* and *stop-and-go waves*. Kerner [22, 23, 24] shows that his model is *metastable* in a certain density area, which means that at certain density levels even the slightest disturbance (e.g. an overtaking truck) can cause a traffic jam. This is partially caused by the competing processes in the relaxation term. On the one hand, drivers tend to accelerate towards their desired velocity. On the other hand, drivers decelerate due to interaction with vehicles downstream. In dense traffic this can lead to so-called stop-start waves where slight disturbations cause small localised traffic jams. Vehicles driving out of these jams with increasing velocity cause a new traffic jam downstream, and the process can repeat.

The density range in which these phantom jams can occur, lies within the steepest part of the $V_e(r)$ line in figure 2.1. When a driver gets in slightly denser traffic r' he wants to decelerate to $V_e(r')$. The deceleration can be quite heavy, since the line is so steep. As drivers cannot react instantly, the density might already have changed as the driver has reached its new velocity. Therefore, he starts accelerating again, and his bahaviour might start to oscillate, causing other drivers to start adapting their velocities as well. It is for this reason that slight perturbations can eventually cause huge traffic jams.

Kerner claims that the above processes were also observed in empirical traffic data, but he has also received quite some critique on his model in the past decade. However, the model itself contains many interesting phenomena, so it wil still be used for simulations in chapter 4.

2.3 Hoogendoorns MultiClass-model

In [15, 16, 17] Hoogendoorn derives a macroscopic model from a mesoscopic gas-kinetic formulation. The first steps are based on earlier work by Helbing [10, 11], but Hoogendoorn integrates the concept of multiple userclasses into the existing formulation. This is also extended to *multiple roadway lanes*, with overtaking behavior and probabilities.

In the following only the multiclass variant will be considered. The multilane extension only results in extra terms in the source term of the PDEs.

2.3.1 Userclasses

The need for multiple userclasses in a macroscopic model stems from the observation that small changes in micro- and mesoscopic traffic configuration can lead to considerable changes in macroscopic traffic flow (i.e. queuelength, flow, capacity). To use the 'best of both worlds' the macroscopic formulation remains intact, but the various quantities are considered for multiple userclasses. Each userclass can have its own specific parameters (like desired velocity or relaxation time).

As a notational convention, variables and parameters that are class-specific, are subscripted with some letter (usually u), as in: $r_u(x,t)$.

Some terms in the PDEs of some userclass u may contain terms or variables from other userclasses u'. This is the case in the interaction term, where vehicles of any class can interact with each other. So in general, the equations for the various userclasses are also coupled.

2.3.2 Phase-Space-Density

At a mesoscopic level, probability density is formulated as the Phase-Space-Density

$$\rho_u\left(x, v, v^0, t\right) = \phi_u\left(v, v^0 | x, t\right) \cdot r_u(x,t), \tag{2.5}$$

denoting the probability of a car of class u, driving at location x and time t, with velocity v and having a desired velocity v^0. The joint probability density function $\phi_u\left(v, v^0|x,t\right)$ specifies for a vehicle at (x,t) the probability of having velocity v and desired velocity v. The actual density of cars is denoted by:

$$r_u(x,t) \equiv \int_v \int_{v^0} \rho_u\left(x, v, v^0, t\right) dv dv^0 \tag{2.6}$$

The *reduced phase-space density* is derived from the phase-space density by making only the desired velocity no longer a dependent variable:

$$\varphi_u\left(x, v, t\right) = \int_{v^0} \rho_u\left(x, v, v^0, t\right) dv^0. \tag{2.7}$$

2.3.3 Method of Moments

The method of moments is a technique from traditional statistics to determine unknown parameters of distribution functions from observations (see also [17]). The concept of *moments* is used to obtain the PDEs that describe the dynamics of the various conservative traffic variables.

The velocity moments and velocity-desired velocity covariance moments are defined as follows:

$$\left\langle v^k \right\rangle_u = \int_{v=0}^{\infty} v^k \frac{\varphi_u\left(x, v, t\right)}{r_u\left(x,t\right)} dv, \quad \text{and} \tag{2.8}$$

$$\left\langle v^k, v^0 \right\rangle_u = \int_{v=0}^{\infty} \int_{v^0=0}^{\infty} v^0 \cdot v^k \frac{\rho_u\left(x, v, v^0, t\right)}{r_u\left(x,t\right)} dv^0 dv. \tag{2.9}$$

Using these notations, Hoogendoorn comes to the following general PDEs.
For $k = 0$:

$$\frac{\partial}{\partial t}\left[r_u\right] + \frac{\partial}{\partial x}\left[r_u V_u\right] = 0, \tag{2.10}$$

and for $k \geq 1$:

$$\frac{\partial}{\partial t}\left[r_u \left\langle v^k \right\rangle_u\right] + \frac{\partial}{\partial x}\left[r_u \left\langle v^{k+1} \right\rangle_u\right] = \frac{k}{\tau_u} r_u \left[\left\langle v^{k-1}, v^0 \right\rangle_u - \left\langle v^k \right\rangle_u\right] + (1-p_u) \sum_{s \in U} r_u r_s \left[\left\langle v^k \xi_u(v) \right\rangle_s + \left\langle v^k \psi_s(v) \right\rangle_u\right], \quad (2.11)$$

where the two source terms at the righthand side denote relaxation and interaction respectively. The overtaking probability is denoted by p_u.

A detailed derivation of (2.10) and (2.11) is given by Hoogendoorn in Ch. 5 of [16].

In [17], Hoogendoorn still uses three moments:

$$M^0 \stackrel{\text{def}}{=} \left\langle v^0 \right\rangle = 1, \quad (2.12)$$

$$M^1 \stackrel{\text{def}}{=} \left\langle v^1 \right\rangle = V, \quad (2.13)$$

$$M^2 \stackrel{\text{def}}{=} \left\langle v^2 \right\rangle = V^2 + \Theta. \quad (2.14)$$

In later work, for example [21], the velocity variance is approximated by an analytical function and equations (2.10) and (2.11) are rewritten to obtain the PDEs for conservative variables. This produces the final system of PDEs that will be used in the remainder of this section.

2.3.4 Conservative 2-PDE formulation of Multiclass Traffic Flow

The eventual system of PDEs is an aggregate-lane version of the model as described in [33], and used by Hoogendoorn in [21]:

$$\frac{\partial}{\partial t}\left[r_u\right] + \frac{\partial}{\partial x}\left[r_u V_u\right] = 0, \quad (2.15)$$

$$\frac{\partial}{\partial t}\left[r_u V_u\right] + \frac{\partial}{\partial x}\left[r_u V_u^2 + r_u \zeta \Theta_u\right] = \frac{r_u \left(V_u^e - v_u\right)}{\tau_u}, \quad (2.16)$$

where the equilibrium velocity is defined as:

$$V_u^e \stackrel{\text{def}}{=} V_u^0 - (1-\pi_u)\tau_u \zeta P_u. \quad (2.17)$$

The symbols used are now explained.

The relaxation time τ_u is the class-specific time that is needed to relax to the equilibrium speed V_u^e. The desired velocity V_u^0 was already introduced in the context of relaxation. The reason that V_u^e is not by definition equal to V_u^0 is that drivers sometimes have to slow down due to interaction.

In the interaction term, π_u denotes the overtaking probability for a vehicle of class u, ζ is a finite space correction factor:

$$\zeta \stackrel{\text{def}}{=} \left[1 - \sum_{u \in U} \left(l_u + d_u + v_u T_u\right) r_u\right]^{-1}, \quad (2.18)$$

where l_u is the length of a vehicle of class u, d_u is the minimal inter-vehicle-distance for class u and T_u is the reaction time.

Introducing ζ corrects the fact that in the original macroscopic equations, vehicles were considered as particles of zero length. Since vehicles have non-zero length and require also some free space as well as a velocity-dependent braking distance, the actual traffic density and interaction are much higher.

Finally, P_u denotes the amount of interaction that exists at a certain location and time. This is an approximation of analytical interaction integrals, made possible by considering the velocities V_u to be Gaussian distributed. In formulae:

$$P_u \stackrel{\text{def}}{=} \sum_{u' \in U} P_{uu'} \quad (2.19)$$

$$P_{uu'} \approx r_u S_{uu'} \left[\delta v_{uu'} \phi\left(\delta v_{uu'}\right) + \left(1 + \delta_{vv'}^2\right) \Phi\left(\delta v_{uu'}\right)\right], \quad (2.20)$$

where ϕ and Φ are the probability density function, respectively the probability distribution function of a standard Gaussian random variate.
To use the *standard* random variate, the speed difference is normalised:

$$\delta v_{uu'} \stackrel{\text{def}}{=} \frac{v_u - v_{u'}}{\sqrt{S_{uu'}}}, \tag{2.21}$$

$$S_{uu'} \stackrel{\text{def}}{=} \Theta_u + \Theta_{u'} - 2\xi\sqrt{\Theta_u \theta_{u'}}. \tag{2.22}$$

where ξ is the correlation between the speed distributions of class u and class u'. In most cases, vehicular chaos is proposed, implying that $\xi = 0$.

For the velocity variance an approximation function $\Theta_u(r, V)$ is used:

$$\Theta_u = \alpha(\hat{r})\, V_u^2, \tag{2.23}$$

where α is the speed-variance pre-factor, described by a smooth, step-like function of the effective density $\hat{r}$, defined by:

$$\alpha(\hat{r}) \stackrel{\text{def}}{=} \alpha_0 + \frac{\delta\alpha_0}{1 + e^{-(\hat{r}-\hat{r}_c)/\delta\hat{r}_c}}, \tag{2.24}$$

$$\hat{r} \stackrel{\text{def}}{=} \frac{1}{d_0 + l_0} \sum_{u \in U} (l_u + d_u)\, r_u. \tag{2.25}$$

The various parameters in (2.24) and (2.25) calibrate the velocity variance to match observed data.

This model will be used for the simulations in chapter 4. Typical values for the various parameters above will be shown there as well.

2.4 The fundamental diagram

Figure 2.1 already showed the relation between density and velocity graphically. This is a variant of a widely used technique, the *fundamental diagram*. This diagram shows the relation between traffic flow quantities. The best-known relation is the one between density r and flow (or momentum) $m \stackrel{\text{def}}{=} rV$. In the ideal situation, the relation between m and r would be a linear one: $m = rV_0$, where V_0 is some fictive constant velocity. However, as the density increases, the velocity in an equilibrium state decreases, because of the stronger interaction influence. For $r \to r_{\max}$, the equilibrium velocity $V_e(r)$ tends to zero, as will the flow m. Figure 2.2 shows the fundamental diagram for Kerners V_e (2.3).

As indicated before, the steep part of the $V(r)$ diagram is unstable, or at least metastable (see section 4.2.1 for a discussion of metastability). The region on the left and right is stable. In the fundamental diagram in figure 2.2, these stable parts have been accentuated by the thick dashed line. What strikes, is that the optimal density $r_{\text{opt}} \approx 33$veh./km is not a stable situation. For low densities, the flow increases rapidly with r, since the equilibrium velocity is still very high. For densities beyond 25-30 veh./km, the velocity can have various values since traffic flow fluctuates somewhere in the unstable area until some new equilibrium state is reached. Traffic might also oscillate between several stable states.

Besides the two discussed diagram types, other diagrams of this kind can be interesting as well. For example the relation between velocity and flow.

A fundamental diagram can also help in understanding the dynamics of certain structures in the traffic flow. If, for example, the traffic distribution on a road consists of a congested and uncongested region, the fundamental diagram can help determine whether the boundary between these two regions will move up- or downstream. The left diagram in figure 2.3 shows such a distribution on a 10 km road. At $x \in [0, 4]$km, $r = 15$veh./km, at $x \in [4.5, 10]$km, $r = 55$veh./km. Placing these densities in the fundamental diagram and connecting them, yields a line (dash-dotted in the right diagram) with negative slope. Thus, the tail of the congested region moves upstream.

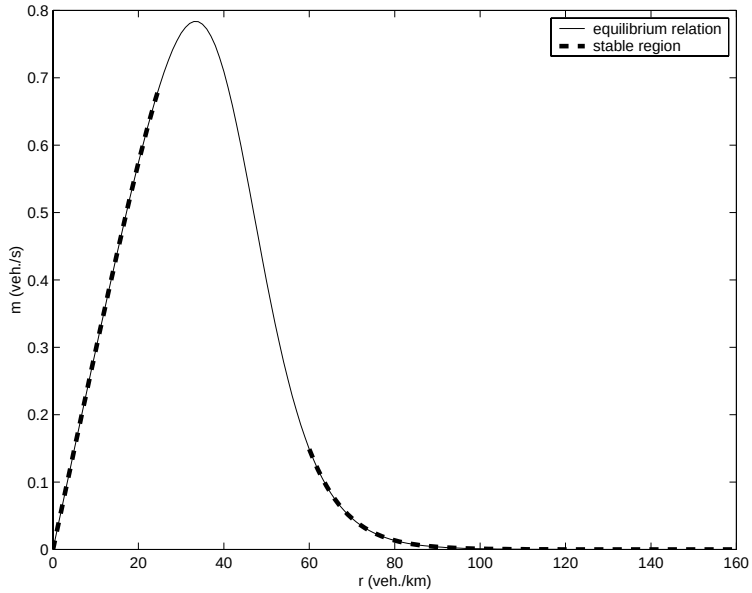

Figure 2.2: Fundamental diagram for Kerners model

This can be interpreted as follows: for $r = 15$, a relatively large flow reaches the congested region (0.41 veh./s). In the congested region, traffic flows downstream at a lower rate (0.24 veh./s), so the structure grows at its tail, or – equivalently – moves upstream.

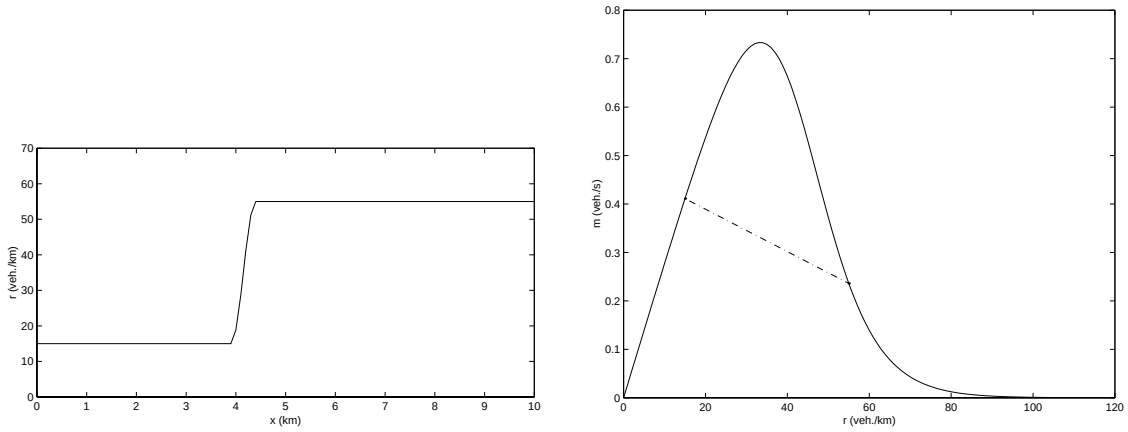

Figure 2.3: A 10km road with a congested and uncongested region (left). The accompanying fundamental diagram is shown right.

Section 4.2 shows several simulations with Kerners model, where results are also explained by the means of these diagrams.

2.5 More model improvements

Much research is still being done, to obtain more and more realistic description of traffic flow. Helbing [11] already proposed a non-local interaction term, for a better description of drivers reaction to traffic conditions downstream.

Another new approach in traffic flow modelling is the further refinement of model parameters. Hoogendoorn already made a first refinement to these parameters by distinguising multiple user-classes and roadway lanes, and making various parameters class-specific. Tampère [37] proposes to do more research in the future to the mechanism behind congestion and traffic flow dynamics in general by considering the microscopic behaviour of individual drivers. By introducing a so-called 'attention level', existing model parameters can be make dependent on how attentive or strained the driver is at that moment. This attent level, or workload, could be introduced as a third model variable, yielding a third PDE.

Chapter 3

Moving mesh methods

Many problems from computational physics give rise to highly varying space- and time-scales. These scale-variations are caused by phenomena like shock-waves and contact-discontinuities. To obtain reliable results from such numerical simulations, one has to use a fine discretization. When using a uniform mesh this often results in an unpractically huge amount of mesh-points.

A better alternative is to use methods that track the areas of high detail (i.e. fine scale) and produce a fine mesh at those areas. One, relatively easy, method is called *static remeshing*, or h-refinement. This method adds or removes meshpoints at locations of high, respectively low detail. The meshpoints themselves do not move.

The basic approach of h-refinement is to use some error-estimate, while solving the problem, and to locate points in the domain where the error is larger than some predefined tolerance-level. At those points, an extra meshpoint is inserted between the two original ones. The algorithm continues until all points satisfy the tolerance requirement. A major advantage of this method is its robustness and reliability: practically all tolerance requirements can eventually be met. The implementation however, requires some extra effort: as the number of meshpoints varies, array-like structures are unsuitable and the solution should be stored in some tree-like datastructure. This requires more effort when addressing certain points in the domain. Another point of attention arises when the solution is time-dependent. At a certain time t_1 great detail is required at some location, which may well be unnecessary later on. Therefore, more advanced implementations of h-refinement also allow removal of meshpoints in areas where less detail is required.

The method that will be used, comes from a class of *dynamic remeshing* methods, also called r-refinement. These methods continuously move meshpoints towards areas of high detail. At the same time meshpoints are 'pulled away' from regions of low detail. The number of meshpoints always remains constant. The moving of the meshpoints could be controlled by again monitoring the error. However, often energy-like functions are used and the meshpoints are moved, so as to minimize this energy.

This method is quite efficient, since it allows extremely accurate representations of sharp solution features, such as shocks. Unfortunately it is less robust than h-refinement, since the maximal accuracy possible, depends on the initial number of meshpoints. Also, points might collapse (move beyond each other), or the 'mesh-energy' might fall into some local minimum.

There exist techniques which combine both approaches: h-r-refinement ([6]). These methods have proved themselves to be both efficient and robust. However, in the following an r-refinement algorithm will be used, that overcomes some of the disadvantages mentioned before.

Dynamic remeshing can be very useful in traffic flow simulation, since often certain areas on the road show interesting traffic behaviour, possibly with shocks or highly varying traffic flow variables, whereas other areas show a fairly constant behaviour of the traffic flow variables. Besides, the areas of interest move along with the traffic stream, so that dynamic adaptation of the mesh is required.

When applied to the system of partial differential equations (PDEs) from the traffic flow model, the method uses a method-of-lines approach (MOL). First, the PDEs are discretized on the spatial

domain, resulting in a semi-discrete system of ordinary differential equations (ODEs) that only depend on time t. Next, this system is discretized in time-direction for each discrete spatial coordinate, resulting in the final, fully discretized scheme.

In the following sections, a moving mesh-method is discussed. First, the mesh generation is introduced, followed by a 1D version of the algorithm. The algorithm has been implemented in MATLAB, some implementation details will be highlighted. Next, the program is applied to Burgers' equation and Sod's shock tube problem, which are especially suitable to study the behaviour of the algorithm at shocks and contact-discontinuities.

3.1 Mesh generation

In the general case, the solution of a system of PDEs is denoted by $\vec{u} = (u_1, u_2, \ldots, u_m)$ and is defined on Ω_p. Here, $m \geq 1$ and Ω_p is the d-dimensional *physical domain* ($d \geq 1$) with coordinates $\vec{x} = (x_1, x_2, \ldots, x_d)$. Introducing a fixed *computational domain* Ω_c, with coordinates $\vec{\xi} = (\xi_1, \xi_2, \ldots, \xi_d)$, a coordinate transformation is denoted by

$$\vec{x} = \vec{x}(\vec{\xi}), \quad \vec{\xi} \in \Omega_c, \tag{3.1}$$

or its inverse

$$\vec{\xi} = \vec{\xi}(\vec{x}), \quad \vec{x} \in \Omega_p. \tag{3.2}$$

The appropriateness or 'quality' of this mesh map $\vec{x}(\vec{\xi})$ can be measured by some costs-function $E(\vec{\xi})$. E is called a *functional*, a function that operates on functions[1]. Finding the best mesh map is equivalent to finding a mesh map $\vec{\xi}$ that minimizes the cost functional $E(\vec{\xi})$. A suitable cost functional is:

$$E(\vec{\xi}) = \frac{1}{2} \sum_k \int_{\Omega_p} \nabla \xi_k^T G_k^{-1} \nabla \xi_k d\vec{x}, \tag{3.3}$$

where the gradient is $\nabla \stackrel{\text{def}}{=} (\partial_{x_1}, \partial_{x_2}, \ldots \partial_{x_d})^T$ and G_k are given symmetric positive definite matrices called *monitor functions*.

3.1.1 Monitor functions

A simple choice of the monitor function is *Winslow's monitor function*: $G_k = \omega I$, where I is the identity matrix and ω is a positive weight function, often incorporating the gradient of the solution, e.g.,

$$\omega = \sqrt{1 + \alpha \|\nabla u\|_2^2}, \tag{3.4}$$

where α is an adaptivity parameter and $\|\nabla u\|_2$ is the L_2-norm as defined in (3.46). When α is set to 0, the monitor values are always equal to 1. The mesh map will eventually be prescribed by (3.8); whenever $\omega = 1$, $(x_\xi)_\xi = 0$ i.e. x_ξ is constant. Thus, x is a linear function of ξ and this is equivalent to a uniform (non-moving) mesh.

The choice of G_k is hardly ever changed; not using a diagonal matrix means introducing non-local terms, i.e. the weight of a certain location is based on values before or ahead of that location, which is rarely useful.

The choice of the weight function, however, leaves much space for variation. As noted before: the weight function often incorporates spatial derivatives of the system variables. Essentially the ones that appear in the PDEs. This results in a denser mesh at locations where the solution changes sharply.

Another argument for choosing a certain weight function is the desire of the researcher to obtain certain parts of the solution in more detail. For example in gas-kinetic problems, areas of high pressure p may be more interesting, because these are the potentially dangerous parts

[1] 'Normal' functions have real values or matrices as their input arguments.

in industrial applications. In that case, the system variables themselves appear in the weight function. Weight functions of this kind possibly look like $\omega = \sqrt{1 + \alpha p^{\beta}}$.

One might also want to include coordinate values, i.e. ω only depends on $\vec{x}$: $\omega(\vec{x})$, in case it is known that a high level of detail is needed in certain parts of the domain. When determining the mesh map from (3.6), having $\omega = \omega(\vec{x})$ results in directly finding the solution $\vec{x}(\vec{\xi})$, independent of time. This static remeshing thus comes down to redistributing the initial domain, and then never changing it again.

Other variants include combinations of the above weight functions.

3.1.2 Mesh map

As mentioned before, the optimal mesh map minimizes the functional in (3.3). Calculus of variations states the *Euler-Lagrange* principle for extrema of functionals.

THEOREM 3.1 (EULER-LAGRANGE VARIATIONAL PRINCIPLE) For functions $\vec{\xi}$ from a set of functions Ξ, a necessary condition for the functional

$$E(\Xi) = \int_{\Omega_p} F(\vec{x}, \Xi, \Xi')d\vec{x},$$

to have an extremum for the function $\vec{\xi}$ is that $\vec{\xi}$ be a solution of

$$E'(\vec{\xi}) = 0,$$

that is

$$\frac{\partial F}{\partial \vec{\xi}} - \frac{d}{d\vec{x}}\frac{\partial F}{\partial \vec{\xi}'} = 0, \quad \vec{x} \in \Omega_p,$$

where the vector gradients are denoted as conventional derivatives for notational purposes.

Substituting (3.3) in the above theorem yields the condition for the optimal mesh map:

$$\nabla \cdot \left(G_k^{-1} \nabla \xi_k\right) = 0, \quad 1 \leq k \leq d. \tag{3.5}$$

Substituting $G_k = \omega I$ yields Winslow's variable diffusion method:

$$\nabla \cdot \left(\frac{1}{\omega} \nabla \xi_k\right) = 0, \quad 1 \leq k \leq d, \tag{3.6}$$

from which the coordinate mapping $\vec{x}(\vec{\xi})$ can be determined.

More details on variational calculus can be found in [3]

3.1.3 1D case

The rest of this section will be focussed on the one dimensional case, i.e. $d = 1$, since traffic flow is essentially one dimensional. Arrows above vector variables will mostly be omitted for notational purposes, e.g. u is a vector, u_j denotes the j^{th} vector element.

Again, $x \in [a, b]$ and $\xi \in [0, 1]$ denote the physical, respectively computational coordinates, that are related by the one-to-one mapping:

$$\begin{aligned} &x = x(\xi), \quad \xi \in [0, 1], \\ &x(0) = a, \quad x(1) = b. \end{aligned} \tag{3.7}$$

Restricting (3.6) to one dimension, and integrating once yields:

$$\left(\frac{1}{\omega}\xi_x\right)_x = 0 \Longrightarrow \frac{1}{\omega}\xi_x = c \Longleftrightarrow \xi_x = c\omega,$$

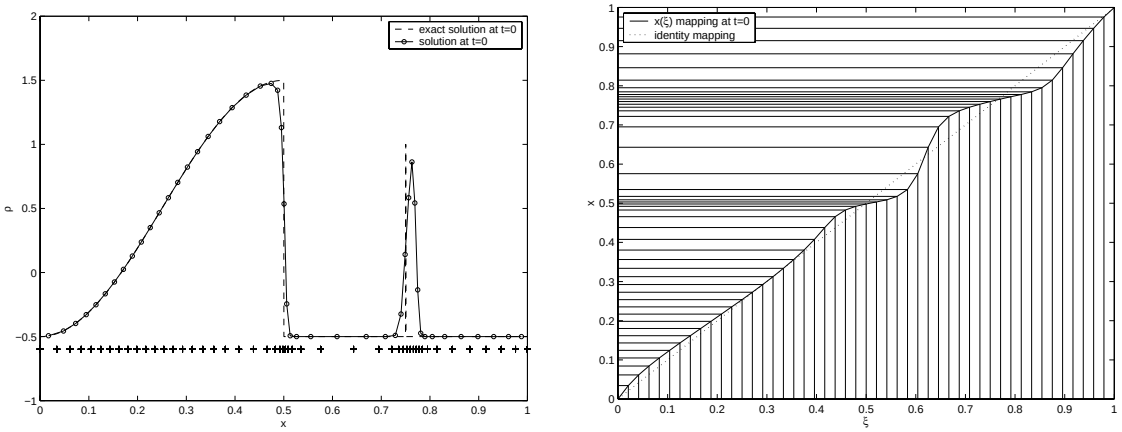

Figure 3.1: Solution and coordinate-mapping in case of a shock and discontinuity.

where c is a scalar constant. Since $x_\xi \equiv 1/\xi_x$, the following holds:

$$\begin{aligned}(\omega x_\xi)_\xi &= \left(\omega \frac{1}{c\omega}\right)_\xi \\ &= \left(\frac{1}{c}\right)_\xi \\ &= 0, \end{aligned} \tag{3.8}$$

which gives, after integrating once, the conventional 1D *equidistribution principle*: $\omega x_\xi = \text{constant}$.

x_ξ is the Jacobian $\mathcal{J}$ of the transformation $x(\xi)$. In order to prevent meshpoints to collapse, this Jacobian should always be nonzero and positive. This is called the mesh-consistency condition:

$$\mathcal{J} \stackrel{\text{def}}{=} x_\xi > 0 \quad \forall \xi \in [0,1], \tag{3.9}$$

or in discrete form (since $\Delta\xi$ is a constant):

$$\Delta x_j \stackrel{\text{def}}{=} x_{j+1} - x_j > 0 \quad \forall j \in \{0, \ldots, N\}. \tag{3.10}$$

3.1.4 Example mapping

As an example, consider an initial solution:

$$\rho(x)|_{t=0} = \begin{cases} 0.5 - \cos(2\pi x) & \text{if } x \leq 0.5, \\ 1 & \text{if } x = 0.75, \\ -0.5 & \text{otherwise.} \end{cases} \tag{3.11}$$

An adaptive mesh was obtained, using the following monitor-function: $\omega = \sqrt{1+\rho_x^2}$. Figure 3.1 shows the interpolated solution $\rho(x)$ at the new mesh, and the meshmap $x(\xi)$ at time $t = 0$.

Figure 3.1 shows the effect of the equidistribution principle: near points where the solution changes abruptly (ρ_x^2 is large), the monitor-value ω is large, so x_ξ should be small. A small value of x_ξ means that meshpoints x_j are lying relatively close to each other, compared to the uniform distribution ξ. The left diagram shows shocks in the solution at $x = 0.5$ and $x = 0.75$, and a flat part between them. Comparing the mapping $x(\xi)$ in the right diagram with the identity mapping shows a small x_ξ near the shocks, i.e. a dense mesh. The horizontal parts of the solution result in large x_ξ, i.e. a relatively coarse mesh. It should be noted that near shocks points are 'pulled away' from the surrounding area. This might result in too coarse meshes (like in $[0.5, 0.75]$ in figure 3.1).

A more sensible way to handle this is using 'monitor-smoothers' which will be discussed in section 3.3.

The interpolated representation of the Dirac-like peak at $x = 0.75$ does not look too good either. Its width has become about 0.05, where it should have been infinitely small. Fortunately, peaks like this do not occur in realistic traffic flow solutions and therefore need not be represented very well. The shock at $x = 0.5$ is represented quite well, and this kind of shocks are more likely to occur in traffic flow solutions (e.g. lanedrops).

The vertical lines in the right diagram start from the equidistant meshpoints ξ in the computational domain. Starting a horizontal line at each intersection with the $x(\xi)$ line and lengthening it to the vertical axis, shows clearly how the adaptive mesh concentrates near the shocks. The +-marks in the left diagram also show the redistributed meshpoints.

3.2 1D algorithm

Now that the mathematical basis is clear, a numerical algorithm can be set up. The following algorithm is based on the description by Tang and Tang in [39].

The spatial domain is divided into $N + 2$ gridpoints ($N \geq 0$) and a fixed, uniform mesh is given on the computational domain: $\xi_j = j/(N + 1)$. The solution u is evaluated between two gridpoints, using a cell-average:

$$u_{j+\frac{1}{2}} = \frac{1}{\Delta x_{j+\frac{1}{2}}} \int_{x_j}^{x_{j+1}} u(x)dx, \tag{3.12}$$

where

$$\Delta x_{j+\frac{1}{2}} = x_{j+1} - x_j. \tag{3.13}$$

The monitor ω is evaluated at the same points as u, derivatives are approximated using finite difference formulas, thus: $\omega = \omega(u)$.

The algorithm basically has four parts, as can be seen in algorithm 1.

1 Generate an initial uniform mesh: $x_j^{[0]} = a + j \cdot \frac{b-a}{N+1}$.
Compute initial values $u_{j+\frac{1}{2}}^{[0]}$ based on cell-average of $u(x, 0)$.
▷ *Do a loop in the time-domain ($n \geq 0$), update mesh and solution, and evolute PDE.*
while $t_n < T$ **do**
  ▷ *Redistribute the mesh in a few steps $\nu \geq 0$:*
  repeat
2 Move grid $\left\{x_j^{[\nu]}\right\}$ to $\left\{x_j^{[\nu+1]}\right\}$, using a Gauss-Seidel iteration (3.14).
3 Compute the solution $\left\{u_{j+\frac{1}{2}}^{[\nu+1]}\right\}$ on the new mesh, using (3.16).
  until $\nu \geq \nu_{\max}$ **or** $\left\|x^{[\nu+1]} - x^{[\nu]}\right\| \leq \epsilon$
4 Evolute the underlying PDEs by using high-resolution finite volume methods on the mesh $\left\{x_j^{[\nu+1]}\right\}$ to obtain the numerical approximations $\left\{u_{j+\frac{1}{2}}^{[\nu+1]}\right\}$ at the time-level t_{n+1}.
end

Algorithm 1: MMFVPDES – 1D moving-mesh finite-volume PDE solver.

3.2.1 Mesh-redistribution

Equation (3.8) can be solved by a few steps of a Gauss-Seidel iteration scheme:

$$x_j^{[\nu+1]} = \frac{\omega\left(u_{j-\frac{1}{2}}^{[\nu]}\right) x_{j-1}^{[\nu+1]} + \omega\left(u_{j+\frac{1}{2}}^{[\nu]}\right) x_{j+1}^{[\nu]}}{\omega\left(u_{j-\frac{1}{2}}^{[\nu]}\right) + \omega\left(u_{j+\frac{1}{2}}^{[\nu]}\right)}, \tag{3.14}$$

where ν is some artificial time-index, allowing a few iteration-steps between t_n and t_{n+1}.

In [39] it is shown that the Gauss-Seidel scheme maintains the *monotonic order of* $x^{[\nu]}$, that is:

$$x_{j+1}^{[\nu]} > x_j^{[\nu]} \Longrightarrow x_{j+1}^{[\nu+1]} > x_j^{[\nu+1]}, \quad \forall\, j \in \{0, \ldots, N\}, \tag{3.15}$$

or, equivalently: x_ξ is strictly monotonically increasing. This is desirable, since otherwise meshpoints might collapse and solution-gradients could explode.

3.2.2 Solution updating on new mesh

Since the meshpoints have been moved in the previous step, the current solution should be determined on the new mesh. Instead of using some conventional interpolation technique, a new method is introduced to update u. This method preserves conservation of mass $\Delta x \cdot u$, which is desirable for a good numerical scheme to hyperbolic conservation laws. Conventional interpolation does not preserve this property.

The mass-conserving method uses the following expression:

$$u_{j+\frac{1}{2}}^{[\nu+1]} = \frac{\left(x_{j+1}^{[\nu]} - x_j^{[\nu]}\right) u_{j+\frac{1}{2}}^{[\nu]} - \left((\widehat{cu})_{j+1}^{[\nu]} - (\widehat{cu})_j^{[\nu]}\right)}{x_{j+1}^{[\nu+1]} - x_j^{[\nu+1]}}, \tag{3.16}$$

where the second-order numerical fluxes are defined by:

$$(\widehat{cu})_j = \frac{c_j}{2}\left(u_j^+ + u_j^-\right) - \frac{|c_j|}{2}\left(u_j^+ - u_j^-\right). \tag{3.17}$$

The method uses a wave speed $c_j^{[\nu]} = x_j^{[\nu]} - x_j^{[\nu+1]}$, and u^+ and u^- are defined by (3.22)

3.2.3 Solving the PDE on new mesh

The fourth step involves the actual evolution of the PDE. This is independent of the previous steps. The solver receives a non-uniform mesh $x^n \overset{\text{def}}{=} x^{[\nu+1]}$ and a solution u^n, and has to determine the new solution u^{n+1} on the same grid x^n. Many efficient numerical techniques exist for this job; in the following a second-order finite-volume approach is described.

The general equation to be solved is

$$u_t + [f(u)]_x = 0. \tag{3.18}$$

Integrating (3.18) over the control volume $[t_n, t_{n+1}\rangle \times \left[x_j^{[\nu+1]}, x_{j+1}^{[\nu+1]}\right]$ leads to the following explicit finite volume method:

$$u_{j+\frac{1}{2}}^{n+1} = u_{j+\frac{1}{2}}^n - \frac{t_{n+1} - t_n}{x_{j+1}^{[\nu+1]} - x_j^{[\nu+1]}}\left(\widehat{f}_{j+1}^n - \widehat{f}_j^n\right), \tag{3.19}$$

where $\widehat{f}_j^n$ is some numerical flux satisfying

$$\widehat{f}_j^n = \hat{f}\left(u_j^{n,-}, u_j^{n,+}\right), \quad \widehat{f}(u,u) = f(u). \tag{3.20}$$

An example of the numerical flux is the Lax-Friedrichs flux:

$$\widehat{f}(a,b) = \frac{1}{2}\left[f(a) + f(b) - \max_u\{|f_u|\}(b-a)\right]. \tag{3.21}$$

In (3.20), $u_j^{n,\pm}$ are defined by:

$$u_j^{n,\pm} = u_{j\pm\frac{1}{2}} + \frac{1}{2}\left(x_j^{[\nu+1]} - x_{j\pm1}^{[\nu+1]}\right)\tilde{S}_{j\pm\frac{1}{2}}, \tag{3.22}$$

where $\tilde{S}_{j+\frac{1}{2}}$ is an approximation of the slope u_x at $x_{j+\frac{1}{2}}^{[\nu+1]}$, defined by:

$$\tilde{S}_{j+\frac{1}{2}} = \left(\text{sign}\left(\tilde{S}^+_{j+\frac{1}{2}}\right) + \text{sign}\left(\tilde{S}^-_{j+\frac{1}{2}}\right)\right)\frac{\left|\tilde{S}^+_{j+\frac{1}{2}}\tilde{S}^-_{j+\frac{1}{2}}\right|}{\left|\tilde{S}^+_{j+\frac{1}{2}}\right| + \left|\tilde{S}^-_{j+\frac{1}{2}}\right|}, \tag{3.23}$$

with

$$\tilde{S}^+_{j+\frac{1}{2}} = \frac{u^n_{j+\frac{3}{2}} - u^n_{j+\frac{1}{2}}}{x^{[\nu+1]}_{j+\frac{3}{2}} - x^{[\nu+1]}_{j+\frac{1}{2}}}, \quad \tilde{S}^-_{j+\frac{1}{2}} = \frac{u^n_{j+\frac{1}{2}} - u^n_{j-\frac{1}{2}}}{x^{[\nu+1]}_{j+\frac{1}{2}} - x^{[\nu+1]}_{j-\frac{1}{2}}}. \tag{3.24}$$

The above method is a MUSCL-type finite volume method and will be applied to some example problems in section 3.5. In practice, however, first some minor changes are made to the schemes described above, to obtain even beter results.

3.3 Program enhancements and MATLAB implementation

A literal implementation of algorithm 1 yields some problems, that can be solved, fortunately, in a not too difficult way.

Since the finite volume method uses adjacent points to approximate spatial derivatives, a problem occurs at the left and right boundary: no left, respectively right adjacent meshpoints exist. Periodicity or boundary conditions are needed.

As noted before, the scheme uses a method of line approach. The finite volume method which results in (3.19), yields the semi-discrete system of ODEs. These however, still have to be solved in the time-domain. A suitable ODE scheme, in combination with stepsize control will be used.

Another problem emerged during the first experiments with the basic program: the mesh was seen to be changed quite abrupt at steep parts of the solution. This causes unnecessary numerical errors while updating the solution on the new grid. The mesh should be moved in a smoother manner. Computational gradients and monitor-smoothers will be used.

Two final enhancements are to make the program capable of solving a *system* of equations, and equations where the righthand side in (3.18) is non-zero.

Further information on the program-code can be found in appendix A.1. Solutions for the above 'problems' will now be discussed and their effect will be shown when applied to an example problem.

3.3.1 Spatial boundaries

Using the higher-order schemes described before, involves finite difference schemes for the spatial derivatives. These schemes have stencils that are typically larger than two points, sometimes even five points. At the two boundaries, either no left or right neighbour points are available. Therefore, at each side two extra points are introduced. The points numbering is slightly modified: the domain $[a,b]$ is discretized using $N-2$ gridpoints, the numbering starts at 1 for the left-most point (which lies two steps beyond a). The numbering of points is depicted in figure 3.2.
The figure also shows the new numbering of the discretized solution:

$$u_j \overset{\text{def}}{=} u|_{x_{j+\frac{1}{2}}}. \tag{3.25}$$

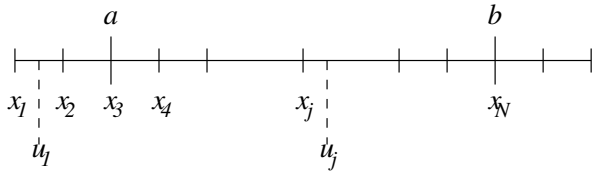

Figure 3.2: The *discretized* spatial domain with 'beyond-boundary'-points.

In some cases, when the solution itself is periodic, the new points can be handled using periodic boundaries:

$$u_1 = u_{N-2}, \quad u_2 = u_{N-1}, \quad u_N = u_3, \quad u_{N+1} = u_4, \tag{3.26}$$

like in example 3.5.1.

Using a periodic solution is only allowed if also a periodic mesh is used.
When moving the mesh, either periodic mesh-distances can be used:

$$\begin{array}{ll} x_2^{[\nu+1]} = x_3^{[\nu+1]} - (x_N^{[\nu+1]} - x_{N-1}^{[\nu+1]}) & x_1^{[\nu+1]} = x_2^{[\nu+1]} - (x_{N-1}^{[\nu+1]} - x_{N-2}^{[\nu+1]}) \\ x_{N+1}^{[\nu+1]} = x_N^{[\nu+1]} + (x_4^{[\nu+1]} - x_3^{[\nu+1]}) & x_{N+2}^{[\nu+1]} = x_{N+1}^{[\nu+1]} + (x_5^{[\nu+1]} - x_4^{[\nu+1]}) \end{array}, \tag{3.27}$$

or periodic mesh-move-speeds:

$$\begin{array}{ll} x_2^{[\nu+1]} = x_2^{[\nu]} + (x_{N-1}^{[\nu+1]} - x_{N-1}^{[\nu]}) & x_1^{[\nu+1]} = x_1^{[\nu]} + (x_{N-2}^{[\nu+1]} - x_{N-2}^{[\nu]}) \\ x_{N+1}^{[\nu+1]} = x_{N+1}^{[\nu]} + (x_4^{[\nu+1]} - x_4^{[\nu]}) & x_{N+2}^{[\nu+1]} = x_{N+2}^{[\nu]} + (x_5^{[\nu+1]} - x_5^{[\nu]}) \end{array}. \tag{3.28}$$

Actually, because of the fixed boundaries $\{a, b\}$, the above two approaches are equivalent.

Whenever periodicity is not desirable, the meshpoints outside the domain $[a, b]$ can be kept fixed at their initial uniform positions. Or, when using a Neumann boundary condition, they might be mirrored in the boundaries:

$$\begin{array}{ll} x_2^{[\nu+1]} - x_3^{[\nu+1]} = -x_4^{[\nu+1]} - x_3^{[\nu+1]} & x_1^{[\nu+1]} - x_2^{[\nu+1]} = -x_5^{[\nu+1]} - x_4^{[\nu+1]} \\ x_{N+1}^{[\nu+1]} - x_N^{[\nu+1]} = -x_{N-1}^{[\nu+1]} - x_N^{[\nu+1]} & x_{N+2}^{[\nu+1]} - x_{N+1}^{[\nu+1]} = -x_{N-2}^{[\nu+1]} - x_{N-1}^{[\nu+1]} \end{array} \tag{3.29}$$

When a Neumann condition

$$u_x|_{\{a,b\}} = 0$$

is used, the solutions outside $[a, b]$ are simply copied from within $[a, b]$.

3.3.2 Effects of enhancements

In the following paragraphs, the program enhancements will be explained and their effects will be illustrated by a single run of the program on the Burgers' problem from example 3.5.1 in section 3.5. All experiments were performed on a Pentium III (600MHz) PC with 128MB RAM running MATLAB 5.3 on Windows 98.

Each experiment will be done with the same amount of meshpoints: $N = 51$, i.e. 49 meshpoints in $[a, b]$.

As a reference, an example run is shown below when the mesh remains fixed, and a simple upwind-scheme (3.30) is used.

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} + \frac{f(u_j^n) - f(u_{j-1}^n)}{\Delta x} = 0. \tag{3.30}$$

The result is shown in figure 3.3, the left diagram shows there is no mesh-movement yet and the right diagram shows the solution $u(x)$ at $t = 0.9$. The numerical solution does resemble the exact solution. However, it does not represent the shock as sharp as necessary and the maximum and minimum are a bit too small.

Some diagnostics on this run are:

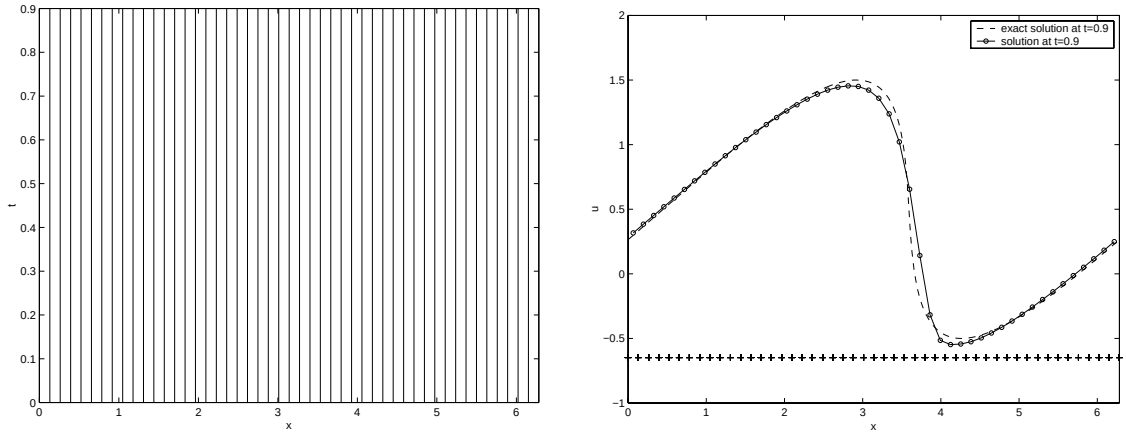

Figure 3.3: Fixed uniform mesh and the solution at $t = 0.9$ for the simplest program (upwind scheme, fixed mesh).

Execution time (s)	Nr.of timesteps ($\Delta t = 0.043$)	L_2-error	L_∞-error
0.16	21	6.27e-2	4.31e-2

3.3.3 Runge-Kutta time-integration

The first enhancement is a more sophisticated integration scheme. In this implementation the finite volume method (3.19) is used in combination with a two-stage Runge-Kutta scheme. By defining the following for the time derivative in the finite-volume part:

$$H^n_{j+\frac{1}{2}} \stackrel{\text{def}}{=} -\frac{1}{x^{[\nu+1]}_{j+1} - x^{[\nu+1]}_j} \left(\widehat{f}^n_{j+1} - \widehat{f}^n_j \right), \tag{3.31}$$

the Runge-Kutta steps are as follows:

$$\begin{array}{rcl} u^{(1)} & = & u^n \\ u^{(2)} & = & u^n + \Delta t H^{(1)} \end{array}, \tag{3.32}$$

where $H^{(i)}$ is H from (3.31) evaluated at $u^{(i)}$. The new solution is determined by:

$$u^{n+1} = u^n + \frac{\left(H^{(1)} + H^{(2)}\right)}{2}. \tag{3.33}$$

This is also known as *Heun's method*.

The effect of this scheme will be illustrated in the next paragraph, in combination with stepsize-control.

3.3.4 Stepsize control and the CFL-condition

In the basic time-integration scheme a fixed time stepsize Δt was used. For the solution to remain stable, Δt should be small enough during the whole simulation. So whenever a very small timestep is required, say beyond $t = 1$, the stepsize is also that small before $t = 1$, which is probably not necessary.

Instead of using a fixed time-stepsize Δt, a suitable value is computed each timestep. Stability analysis of hyperbolic PDEs like (3.18) shows that, when using a central difference scheme and a forward time-integration like Euler-Forward, the iterative solution only remains stable when the following condition is satisfied:

$$\left| \frac{f_u \Delta t}{\Delta x} \right| \leq 1, \tag{3.34}$$

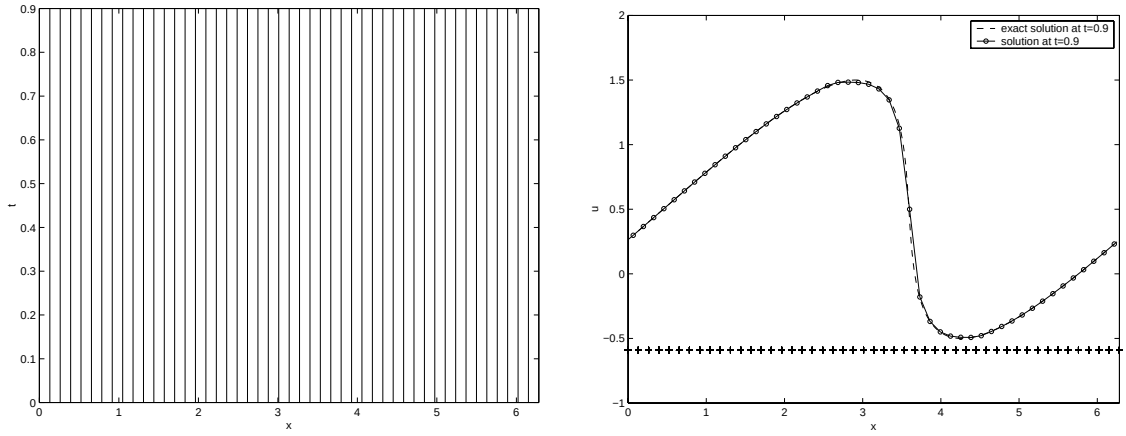

Figure 3.4: Fixed uniform mesh and the solution at $t = 0.9$. A finite volume approach and Runge-Kutta time integrator is used, in combination with stepsize control.

where $f_u \stackrel{\text{def}}{=} \partial f/\partial u$. This condition is also known as the *Courant-Friedrichs-Lewy-condition*. For more details on the CFL-condition and derivation of the general case (system of equations, various time-integrators), see for example [13], Ch. 8. Often the *CFL-number* σ is used in this context:

$$\sigma = \frac{\max(f_u)\Delta t}{\Delta x}.$$

To use the CFL-condition for various PDEs, it suffices to add an extra parameter $\mathcal{C}$ that specifies how close the timestep should be fitted to the CFL-condition. Now, an expression for the maximal timestep to be taken is:

$$\Delta t_{\max} = \frac{\mathcal{C}}{\sigma}, \tag{3.35}$$

where $0 < \mathcal{C} \leq 1$.

Using the new scheme and stepsize-control, the program was run again on the Burgers' problem. In all experiments $\mathcal{C} = 0.5$ was used. The results are shown in figure 3.4.

The left diagram in figure 3.4 shows the mesh-movement (which is still constant) and the right diagram shows the solution $u(x)$ at $t = 0.9$. Around the steepest part ($x \approx 3.5$), the solution is represented much better than in the reference-run. This also becomes clear from the error-estimates as can be seen below:

Execution time (s)	Nr.of timesteps	L_2-error	L_∞-error
0.22	21	1.20e-2	1.05e-2

The execution time has increased only by a factor 1.375, while the error has become smaller by more than a factor 4. These accurate results are also what one would expect from a higher (2nd) order method. In section 3.4 the exact convergence order will be investigated, to see whether the scheme is genuinely second-order accurate.

The number of timesteps has not changed, so stepsize control has not been very effective. This is because the considered solution is very smooth. In the next sections solutions will have (near-)shocks and that is where stepsize control might be more useful.

Although the accuracy has increased, the right diagram still shows that more points near the shock would probably yield far better representation of the shock. This is where adaptive meshing comes in.

3.3.5 Moving the mesh

In the next paragraphs two enhancements to the mesh-moving algorithm will be introduced. First some experiments will be performed to investigate the effects of basic mesh-movement. The

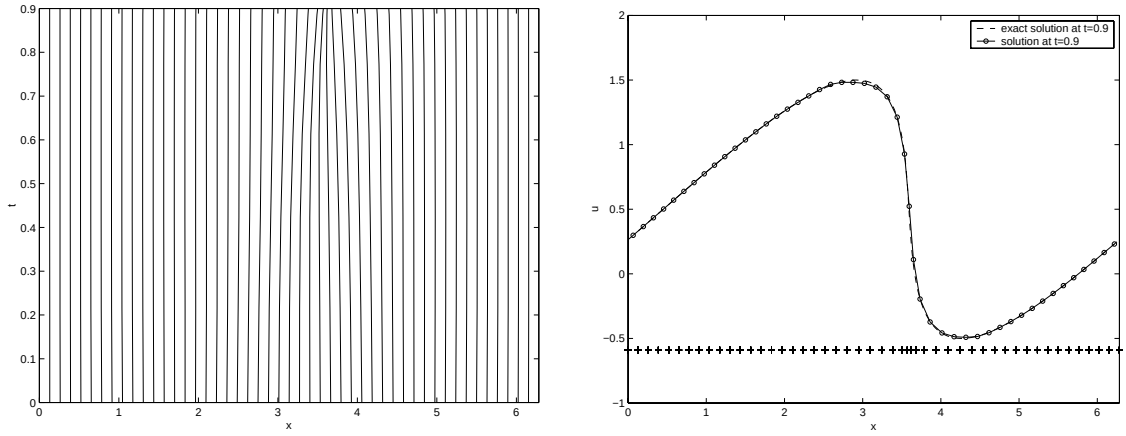

Figure 3.5: Mesh-movement through time and the solution at $t = 0.9$. In addition to the previous enhancements, the mesh is now non-uniform ($\alpha = 0.3$).

monitor function from (3.4) is used, with various values for α. Some results are shown below (note that in the previous experiments, α was actually 0):

α	Execution time (s)	Nr.of timesteps	L_2-error	L_∞-error
0.01	0.38	21	1.19e-2	1.03e-2
0.1	0.44	23	8.74e-3	6.47e-3
0.3	0.5	26	6.38e-3	2.94e-3
0.5	0.52	28	6.20e-3	3.00e-3
1	0.55	31	6.73e-3	3.50e-3

An adaptive mesh (e.g. $\alpha = 0.3$) results in a more accurate solution, but more time is needed as well. Especially the L_∞-norm decreases, when using a moving mesh. This is quite logical, since the overall solution was already fairly accurate, but only at the steep part some large errors were present. The adaptive mesh makes even these errors smaller, because the meshpoints concentrate at these steep parts. The results for $\alpha = 0.3$ are shown in figure 3.5. The mesh-movement diagram shows the movement of meshpoints towards $x \approx 3.5$ and the resulting coarser mesh in the regions around this point.

Again looking at the table, it shows that when the adaptivity is set too large, the mesh concentrates too much at the steep part. These meshpoints have been pulled away from the regions around it (i.e. around $x = 3$ and $x = 4.2$). As a result the accuracy in those regions has decreased quite a bit, yielding a larger overall error. Thus, it is necessary to choose a suitable α for each distinct problem. This will cause the mesh to be moved not too quickly in time and heavily in space.

3.3.6 Computational vs. physical gradients

Whenever sharp peaks or abrupt shocks occur in solutions, the mesh is sometimes being moved too quickly towards the steep parts of the solution. Then, the meshpoints are pulled away from the surrounding areas so quickly, that unnecessary errors are made. Besides, the stepsizes become so small, that execution time becomes very large.

One way of preventing abrupt changes in the mesh-distribution is to use computational gradients u_ξ instead of physical gradients u_x in the monitor function ω. The gradients are numerically approximated by dividing by meshwidths Δx, so these gradients might become huge since at the moved mesh some meshwidths can be very small. Taking computational gradients involves dividing by $\Delta\xi$, which is fixed.

The change to the computational domain can also be made, when reconstructing the solution $u^{[\nu]}$ at the new grid $x^{[\nu+1]}$. In that case, in equations (3.22) and (3.24), the terms involving x-differences should be replaced by $\Delta\xi$, but eventually these will all cancel out in (3.22).

Since the values of the computational gradients are different, the monitor values also change, and the adaptivity parameter α has to be changed accordingly. The optimal value (for still the same number of meshpoints) experimentally turned out to be $\alpha = 0.02$. Again determining the solution at $t = 0.9$ hardly shows any difference with physical gradients (compare it with the $\alpha = 0.3$ entry of the previous table):

Derivatives	α	Execution time (s)	Nr.of timesteps	L_2-error	L_∞-error
computational	0.02	0.49	27	6.40e-3	2.94e-3

However, when the simulation is run up to $t = 2.0$, the solution contains a shock. The difference between the two approaches now becomes clear.

N	Derivatives	α	Execution time (s)	Nr.of timesteps	L_2-error	L_∞-error
51	physical	0.3	6.76	412	8.62e-3	5.54e-3
51	computational	0.02	4.23	141	2.12e-2	1.53e-2
80	computational	0.02	10.33	329	8.64e-3	7.59e-3

The computational gradients do not seem to work very well, the errors have become almost three times as big for the same amount of meshpoints. However, a closer look shows that for the physical gradients, the L_2 and L_∞ error are different, whereas for the computational gradients (at $N = 80$) they are almost the same. This means that for the computational gradients the overall L_2 error is almost entirely caused by one maximal (L_∞) error. This maximal error comes from a point at or very near the shock. One might argue that a valid error-estimate should exclude values at the shock.

At $t = 0.9$ the solution and mesh-movement does not differ too much from the results in figure 3.5. In the next paragraph, the effect of computational derivatives will be shown in combination with monitor-smoothing. Also, a modified error-estimate is then used in order to omit error-values at the shock.

3.3.7 Smoothing of the monitor function

Another technique to prevent the grid from being moved too bruskly, when some local gradient changes rapidly, is to smooth the monitor function. This is done by applying a low-pass filter:

$$\omega^{\text{smooth}}_{j+\frac{1}{2}} \longleftarrow \frac{1}{4}\left(\omega_{j+\frac{3}{2}} + 2\omega_{j+\frac{1}{2}} + \omega_{j-\frac{1}{2}}\right), \tag{3.36}$$

where $\omega_{j+\frac{1}{2}} = \omega\left(u_{j+\frac{1}{2}}\right)$. Optionally, this filter can subsequently be applied several times:

$$\omega^0_{j+\frac{1}{2}} \overset{\text{def}}{=} \omega_{j+\frac{1}{2}}, \quad \omega^{k+1}_{j+\frac{1}{2}} = \frac{1}{4}\left(\omega^k_{j+\frac{3}{2}} + 2\omega^k_{j+\frac{1}{2}} + \omega^k_{j-\frac{1}{2}}\right), \quad k \geq 0,$$

This way, abrupt changes in the solution are smoothed by averaging its monitor value with the values at the neighbour points.

Again, the enhanced program is used to solve the Burgers' problem, now up to $t = 2.0$. As mentioned before, it would be sensible to omit error-values at the shock. In section 3.4 a modified error-estimate is defined, which is specially set up for the Burgers' problem. It detects the location of the shock, and computes the error-norm only for points outside the shock. The error at the shock is based on the error in the *location* of the shock. The table below again shows the experiments with both physical and computational gradients, extended with application of the monitor smoother. The last two columns show the steepness of the shock.

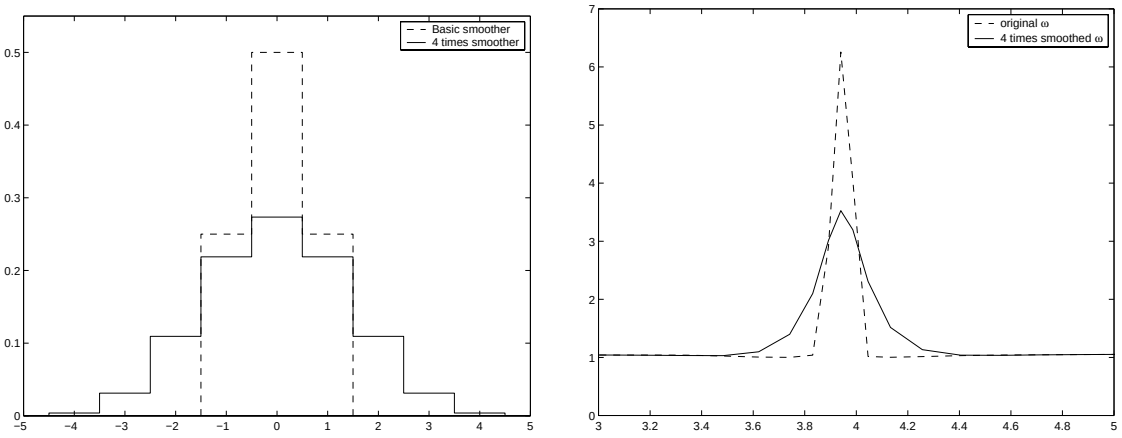

Figure 3.6: Effect of repeated monitorsmoothing. Left: Basic smoother and resulting 4-times smoother; right: Effect of this smoother on monitor-values.

Smooth. steps	Deriv.	α	Exec. time	Time steps	Error outside shock		Rel.error shock loc.	Avg. deriv.	Max. deriv.
					L_2-error	L_∞-error			
0	Phys.	0.3	7.57	412	3.21e-3	2.70e-3	2.40e-5	-8.96	-44.6
0	Comp.	0.02	2.31	132	3.09e-3	2.33e-3	3.44e-4	-7.00	-20.5
1	Comp.	0.02	1.98	109	2.50e-3	1.90e-3	3.88e-4	-8.23	-20.1
2	Comp.	0.02	1.81	98	2.02e-3	1.39e-3	4.48e-4	-8.42	-18.9
4	Comp.	0.02	1.7	84	1.60e-3	1.12e-3	6.76e-4	-7.73	-16.2
6	Comp.	0.02	1.59	77	1.39e-3	9.49e-4	6.19e-4	-7.15	-14.5
10	Comp.	0.02	1.48	68	1.21e-3	8.67e-4	4.05e-4	-6.26	-12.4

In the previous paragraph, the introduction of computational derivatives at first did not look too good. Comparing the first two rows of the table above, shows that they do produce a slightly more accurate solution, whenever the error is not measured at the shock. Major advantage of this new approach is the execution time, which is about 30% of the original time. The location and steepness of the shock in the approximated solution are quite good as well.

Keeping the derivatives in the computational domain, now the effect of monitor smoothing is investigated. The smoother from (3.36) is applied several times, and the accuracy is seen to improve whenever the smoother is applied more often. Still, the error at the shock itself should be considered as well. The error in the shock location already became almost 15 times bigger when the change to computational derivatives was made, but it is still very good. For increased smoothing, this errors grows very little. More important is the steepness of the shock. The physical derivatives did very well, having a maximal slope of almost -45. For strong smoothing the slope becomes less and less steep. Choosing a suitable value for the number of smoothings depends on what kind of accuracy one wants. In the following experiments no more than 4 smoothings were used.

Figure 3.7 compares three experiments: using physical and computational derivatives and computational derivatives with a 4-times smoother. The mesh-movement diagrams indeed show a smoother adaptation of the grid. The solution at the shock is seen to become less accurate (less steep) for computational gradients and smoothing, but it is also seen that the overall error in the rest of the domain becomes significantly more accurate, so these techniques still proved to be useful.

Apart from the above low-pass filtering, other techniques exist for smoothing the monitor function. Zegeling et al. [41] use a more advanced smoothing-technique that ensures 'local quasi-uniformity'. This means that the variation in size of two adjacent gridcells never exceeds a certain value. As mentioned before, meshpoints are pulled away from regions near very steep parts of

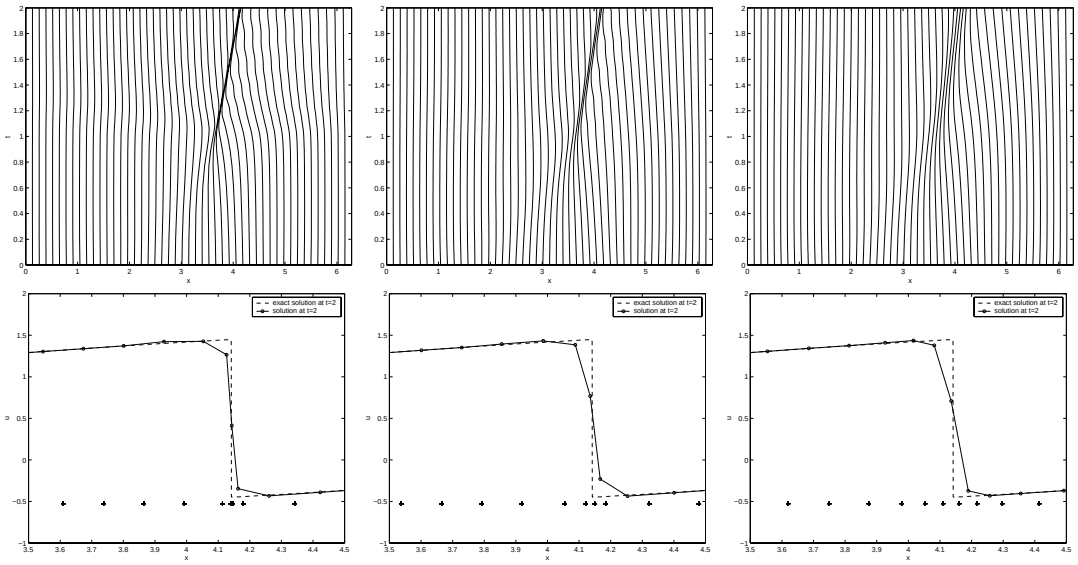

Figure 3.7: Burgers' problem, top row: mesh-movement, bottom row: detail of solution near shock. Left: physical gradients, middle: computational gradients, right: computational gradients using 4 smoothings.

the solution. In this approach however, all gridcells in the domain are shifted slightly towards these parts, so that the mesh outside these steep parts does not become too coarse. Although better, this approach is also more expensive, since the computation of monitor values is more complicated. Besides this spatial smoother, they also uses a temporal smoother to prevent the mesh from moving too much between two successive timesteps.

3.3.8 A system of PDEs

Until now, the considered PDE problems (3.18) had only one variable, like u or ρ. Often however, applicationsinvolve a system of PDEs, now with a vector variable $\vec{u} = (u_1, u_2, \ldots, u_m)$, with $m \geq 1$. The system to be solved now is:

$$\vec{u}_t + \vec{f}(\vec{u})_x = \vec{0}, \tag{3.37}$$

where $\vec{u}$, $\vec{f}$ and $\vec{0}$ are m-dimensional vectors and $\vec{0}$ is a vector of zeros.

The implemented solver should be capable of handling these systems as well. Mathematically speaking, not much changes. Each variable u_i ($i \in \{1, \ldots, m\}$), and its accompanying one dimensional PDE acts on its own. In fact there is a set of m PDEs like (3.18), that can be solved semi-independently. The equations are only related because any variable u_i may occur in any function $f_{i'}$ ($i, i' \in \{1, \ldots, m\}$). Since an explicit time integration scheme is used, all arguments u_i are already available when evaluating $f_{i'}$ at some time t_n.

Extending the program to systems of PDEs thus comes down to some changes in implementation, like datastructures and variable-indexing. The only part of the algorithm that *does* change is the part that involves the Jacobian f_u. Since f_u is now a matrix of partial derivatives, additional measures should be taken.
In the Lax-Friedrichs flux (3.21), the diagonal of f_u is taken to obtain again an m-dimensional vector. In the CFL-criterium (3.34) the maximal absolute eigenvalues of f_u is used, instead of $|f_u|$.

3.3.9 Non-zero source term

Until now the righthand side of (3.18), or (3.37), was always equal to zero. Future applications have a function for the righthand side, which will generally be non-zero:

$$\vec{u}_t + \vec{f}(\vec{u})_x = \vec{g}(\vec{u}), \tag{3.38}$$

where $\vec{g}(\vec{u})$ is an m-dimensional vector function.

The difference formulas for the space-derivatives do not change, however the time integration scheme has to be slightly modified. Equation (3.19) now becomes:

$$u_{j+\frac{1}{2}}^{n+1} = u_{j+\frac{1}{2}}^{n} - \frac{t_{n+1} - t_n}{x_{j+1}^{[\nu+1]} - x_j^{[\nu+1]}} \left(\widehat{f}_{j+1}^{n} - \widehat{f}_j^{n} \right) + (t_{n+1} - t_n)\, g(u_{j+\frac{1}{2}}^{n}). \tag{3.39}$$

The time-derivative approximation from (3.31) now becomes:

$$H_{j+\frac{1}{2}}^{n} \overset{\text{def}}{=} -\frac{1}{x_{j+1}^{[\nu+1]} - x_j^{[\nu+1]}} \left(\widehat{f}_{j+1}^{n} - \widehat{f}_j^{n} \right) + g\left(u_{j+\frac{1}{2}}^{n} \right). \tag{3.40}$$

Using the Heun-scheme (3.33), two evaluations of $\vec{g}(\vec{u})$ are required.

3.4 Error estimates and parameter finetuning

Now that the algorithm with all its modifications is complete, some experiments will be performed to obtain some intuition on the choice of parameters. One would like to have some guidelines to choose an appropriate α. One requirement for this is a valid error-estimate. Several choices for this will be discussed.

3.4.1 Error estimates

Estimating the error actually requires the exact solution. For the Burgers' equation an exact solution can still be found. For more advanced problems this is usually almost impossible. Therefore a 'semi-exact' reference solution will be obtained by running the program with a huge amount of meshpoints, typically near $N = 1000$.

A first basic error estimate is the absolute error:

$$e = |u^n - u(t_n)|\,, \tag{3.41}$$

where u^n is the approximated solution u at time t_n.

For an honest comparison of various points, the relative error might be used:

$$e_{\text{rel}}|_j = \frac{e_j}{|u_j|} \quad j \in [3, N-1], \tag{3.42}$$

thus the error is actually normalized. However, when solutions have values near 0, the relative error might become huge because of the division by a very small exact solution. This was also seen to occur with the Burgers' example problem, therefore the error is not normalized:

$$e_{\text{rel}} = e. \tag{3.43}$$

When using adaptive meshes some errors 'live' on a larger interval Δx_j than others do. Therefore a 'weighed' error estimate is introduced:

$$e_{\text{w}}|_j = e_{\text{rel}}|_j \cdot \Delta x_j \quad j \in [3, N-1]. \tag{3.44}$$

This is also useful when comparing experiments with varying amounts of meshpoints: all errors are eventually scaled to the same spatial domain, independent of the amount of meshpoints[2].

Note that all estimates above result in an error *vector*. Mostly, some vector norm is used to obtain a scalar expression for the error. In general the p-norm of a vector e is:

$$\|e\|_p \stackrel{\text{def}}{=} \left(\sum_j |e_j|^p \right)^{1/p} \tag{3.45}$$

In this case the L_2 and L_∞-norm are used:

$$\|e\|_2 \equiv \sqrt{|e| \cdot |e|}, \qquad \|e\|_\infty \equiv \max_j |e_j|. \tag{3.46}$$

3.4.2 Errors at a shock

When dealing with shock waves, the above error-estimates are not always appropriate, especially not when the solution at the shock is near 0. Only a small shift in x-direction results in a huge difference in the solution u. Several alternative error-estimates have been proposed for this. In [26] the errors are computed at a smooth pre-shock time. This was also done in some of the previous experiments at $t = 0.9$.

Other experiments are also performed beyond the critical time, i.e. at times where the solution does contain a shock. Therefore in this program, the location of the shock is detected by (3.48), and the above errors and norm are applied only to points before and after the shock[3]:

$$\begin{aligned} j_{\text{start}} &= \text{argmax}(u) \\ j_{\text{end}} &= \text{argmin}(u) \\ \mathcal{J} &= \{3, \ldots, j_{\text{start}}\} \cup \{j_{\text{end}}, \ldots, N\} \end{aligned} \tag{3.47}$$

Still, it is quite odd to just omit some points at which the error might be large. At those locations it is much more interesting to check the *location* of the shock instead of its *value*. The location of the shock can be defined by:

$$x_{\text{shock}}, \text{ such that } u(x_{\text{shock}}) = \frac{u_{\text{top}} + u_{\text{bot}}}{2}, \tag{3.48}$$

where u_{top} and u_{bot} are the solution u at the top and bottom of the shock respectively.
The relative error in the shock-location can then be defined as:

$$(e_x)_{\text{rel}} \stackrel{\text{def}}{=} \frac{|x_{\text{shock}} - (x_{\text{shock}})_{\text{exact}}|}{b - a} \tag{3.49}$$

Figure 3.8 illustrates the location of the shock. The half-height is determined and the x-coordinate of this height is the location of the shock (3.48), here: $x = 4.141$, which is almost the same as the exact location. The two dash-dotted lines are the average and maximal derivative value at the numerical shock.

Finally, the slope of the shock can serve as an error estimate. For genuine shocks the slope is ∞ (here: $-\infty$). For the numerical solution holds: the bigger the slope, the better. At uniform meshes the slope is seriously limited by the fixed mesh-size. Since the part of the domain 'outside the shock' is defined, the part of the domain that forms the shock is also known. Two slopes can be computed: the maximal slope (at one Δx), and the average slope over the shock part of the domain, see also the dash-dotted lines in figure 3.8.

[2] When using a uniform mesh, the weighed error is actually the relative error, multiplied by the domain-size (i.e. some fixed constant), and divided by the amount of meshcells. Thus the weighed error can also be seen as 'the average error per mesh element'.

[3] The summation in (3.45) is now taken over $j \in \mathcal{J}$ from (3.47)

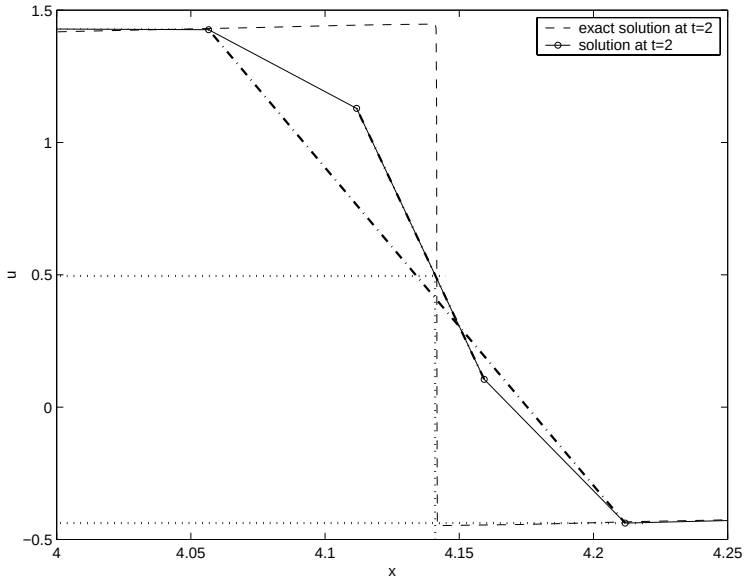

Figure 3.8: Finding the location of a shock by the x-coordinate of its half-height.

3.4.3 Choosing the adaptivity parameter

In the previous experiments the optimal value of the adaptivity parameter α was easy to choose, since the error could be computed. When applied to more complex problems, where it is not possible to compute the error in a relatively cheap way, it would be best if some rule of thumb existed to choose an optimal α. As α controls the size of the monitorvalues ω, actually the optimal value of the monitor function is to be chosen. This depends on the value of the gradient ∇u, so α probably depends on the range of u and the size of the domain $[a, b]$, as well as the number of meshpoints (N).

In the following experiments the overall maximum and average of ω will be monitored, as well as the time-average of the maximum ω:

$$\omega_{\max} = \max_{0\le t_n\le T}\ \max_{3\le j\le N-1} \omega_j^n, \tag{3.50}$$

$$\overline{\omega} = \left(\sum_{t_n=0}^{T}\sum_{j=3}^{N-1}\omega_j^n\right) / \left(n_t\cdot(N-1)\right), \tag{3.51}$$

$$\overline{\omega_{\max}} = \left(\sum_{t_n=0}^{T}\ \max_{3\le j\le N-1}\omega_j^n\right) / \left(n_t\cdot(N-1)\right), \tag{3.52}$$

where $\omega_j^n \stackrel{\text{def}}{=} \omega|_{x_{j+\frac{1}{2}}}^{t_n}$ and n_t is the amount of timesteps.

First experiments with 5 different meshsizes were considered at pre-shock time $t = 0.9$. Many experiments with different adaptivity α were compared, and the optimal α's (i.e. the smallest L_2-errors) were selected:

N ($N-3$)	α	L_2-error	L_∞-error	$\omega_{\max}$	$\overline{\omega}$	$\overline{\omega_{\max}}$	p
27 (24)	0.04	2.59e-2	1.50e-2	2.35	1.32	1.93	-
51 (48)	0.02	6.42e-3	2.83e-3	2.61	1.19	1.83	2.01
99 (96)	0.04	1.38e-3	5.68e-4	4.37	1.34	2.63	2.22
195 (192)	0.04	2.72e-4	9.18e-5	6.09	1.41	3.17	2.35
387 (384)	0.04	4.87e-5	1.62e-5	6.95	1.34	3.08	2.48

The last column shows for each experiment the convergence order p with respect to the previous experiment. This clearly shows that the schemes are indeed second-order accurate.

The optimal adaptivity is when $\overline{\omega} \approx 1.3$.

The same experiments have been performed, only now up to $t = 2.0$. The results are a little different:

N $(N-3)$	α	Error outside shock			Error entire domain			$\omega_{\max}$	$\overline{\omega}$	$\overline{\omega_{\max}}$
		L_2	L_∞	p	L_2	L_∞	p			
27 (24)	0.002	9.63e-3	8.98e-3	-	1.14e-1	1.08e-1	-	1.17	1.03	1.12
51 (48)	0.02	1.59e-3	1.12e-3	2.59	3.80e-2	3.76e-2	1.59	3.55	1.23	2.91
99 (96)	0.04	1.48e-4	7.83e-5	3.43	9.85e-3	9.76e-3	1.95	9.41	1.41	7.76
195 (192)	0.02	2.02e-5	8.08e-6	2.87	2.61e-3	2.58e-3	1.91	13.2	1.28	11.1
387 (384)	0.02	4.45e-6	2.64e-6	2.18	5.90e-4	5.28e-4	2.15	26.3	1.30	23.0

The errors outside the shock for fixed N varied very little for different α. Therefore the choice of the optimal α has also been based on the error at the entire domain, the location and the steepness of the shock.

The convergence order is less uniform than before. Possibly the schemes act less regular in presence of shocks. A more plausible explanation is that the error estimates still aren't as suitable as they are without shocks. Still, for the error outside the shock the order is always larger than two, and for the overall error, it is mostly close to two, so the integration schemes still prove to be second-order accurate.

The optimal adaptivity again shows $\overline{\omega} \approx 1.3$, which is quite logical. This value apparently moves the mesh effectively towards steep parts of the solution, but it still does that in such a mild way, that the mesh is not moved too abruptly. The accompanying value of α now lies at $\alpha = 0.02$. There might be even more accurate values, but the resulting errors differ so little that this suffices.

In the previous two tables, the values of α seem not directly related to N, this is because ω is also strongly influenced by the Δu between two successive points. In more detail: Doubling $(N-3)$ halves $\Delta\xi$. This does not mean that $\partial u/\partial\xi$ becomes twice as big, since Δu almost certainly has changed as well. Δu is expected to halve as well, since the meshpoints lie two times closer to each other. The resulting $\Delta u/\Delta\xi$ thus would not change at all, so α should not have to be changed either. The previous tables indeed show this: except for one experiment, the optimal adaptivity paramater α is always the same (0.4, resp. 0.2).

The range of u probably also affects the optimal α. α is probably reversed proportional to the range of u. Whenever the range of u doubles, $\Delta u/\Delta\xi$ also doubles, so α should be halved. To verify this, the Burgers' equation from example 3.5.1 is slightly modified:

$$u_t + \left(\frac{u^2}{4}\right)_x = 0, \quad 0 \le x \le 2\pi, \tag{3.53}$$

subject to the 2π-periodic initial data

$$u(x,0) = 2 \cdot (0.5 + \sin(x)), \quad x \in [0, 2\pi]. \tag{3.54}$$

The u-range is now twice as big. The PDE has been changed a little so that the shock still arises at critical time $t_c = 1$. In the original problem the optimal adaptivity was $\alpha = 0.02$. Now, since u has doubled, u_ξ^2 becomes four times as big, so α should be four times as small. The optimal α now is expected to be 0.005.

For various α the simulation is run up to $t = 2.0$, for $N = 51$:

α	Error outside shock		Error entire domain		$\omega_{\max}$	$\overline{\omega}$	$\overline{\omega_{\max}}$
	L_2	L_∞	L_2	L_∞			
0	9.32e-3	7.71e-3	1.84e-1	1.83e-1	1	1	1
0.001	4.18e-3	3.82e-3	7.49e-2	5.91e-2	1.90	1.07	1.62
0.002	3.70e-3	3.36e-3	7.42e-2	6.56e-2	2.44	1.12	2.03
0.005	3.26e-3	2.54e-3	8.41e-2	8.31e-2	3.56	1.23	2.91
0.0075	3.49e-3	2.41e-3	7.13e-2	7.07e-2	4.25	1.31	3.47
0.01	3.69e-3	2.39e-3	5.65e-2	5.59e-2	4.83	1.38	3.96
0.02	4.32e-3	2.53e-3	3.61e-2	3.46e-2	6.65	1.62	5.47

The results show that $\alpha = 0.005$ is indeed the optimal value. Possibly $\alpha = 0.0075$ would do as well. The average monitor value is again near 1.3.

Next, the dependence on the domain size is investigated. Now the modified Burgers' equation is considered on the domain $[0, 1]$:

$$u_t + \left(\frac{u^2}{4\pi}\right)_x = 0, \quad 0 \le x \le 1, \tag{3.55}$$

subject to the 2π-periodic initial data

$$u(x, 0) = 0.5 + \sin(x), \quad x \in [0, 1]. \tag{3.56}$$

Since ω depends on $\Delta u/\Delta\xi$, the optimal adaptivity is not expected to change[4]. The experimental results are shown below:

α	Error outside shock		Error entire domain		$\omega_{\max}$	$\overline{\omega}$	$\overline{\omega_{\max}}$
	L_2	L_∞	L_2	L_∞			
0	7.27e-4	5.88e-4	1.47e-2	1.46e-2	1	1	1
0.001	3.72e-4	3.68e-4	1.08e-2	1.07e-2	1.32	1.02	1.20
0.005	3.13e-4	2.84e-4	6.79e-3	6.68e-3	2.05	1.08	1.72
0.01	2.96e-4	2.43e-4	5.32e-3	5.20e-3	2.66	1.14	2.19
0.02	2.89e-4	2.04e-4	4.17e-3	4.06e-3	3.54	1.23	2.89
0.04	3.08e-4	2.01e-4	3.38e-3	3.28e-3	4.82	1.38	3.91
0.08	3.66e-4	2.30e-4	2.86e-3	2.79e-3	6.63	1.62	5.41

The optimal adaptivity parameter is $\alpha = 0.02$, as expected. The average $\overline{\omega}$ is again almost the same.

In this case, the choice of α thus depends on the average monitor value. The above shows that for one specific problem, the optimal adaptivity always has $\overline{\omega} \approx 1.3$.

It would be interesting to know whether this also holds for other problems like example 3.5.2. Specifically, the possible dependence of $\overline{\omega}$ on the number of shocks in the solution is interesting. Therefore some final experiments are performed, based on the Burgers' problem, with varying domain-sizes:

$$u_t + \left(\frac{u^2}{2}\right)_x = 0, \quad 0 \le x \le n \cdot 2\pi, \tag{3.57}$$

subject to the 2π-periodic initial data

$$u(x, 0) = 0.5 + \sin(x), \quad x \in [0, 1]. \tag{3.58}$$

Since the domain has become n times as big, whereas the initial solution is still 2π-periodic, equation (3.57) will produce solutions with n shocks. The number of meshpoints was set at $(NX - 3) = 50n$, and the optimal adaptivity was selected from experiments with various α. For an honest comparison, the maximum number of remeshing-iterations (the inner **repeat**-loop in algorithm 1) is set unrestricted ($\nu_{\max} = \infty$), whereas the tolerance ϵ is set proportional to the

[4] $\Delta\xi$ does not depend on the domainsize $(b - a)$.

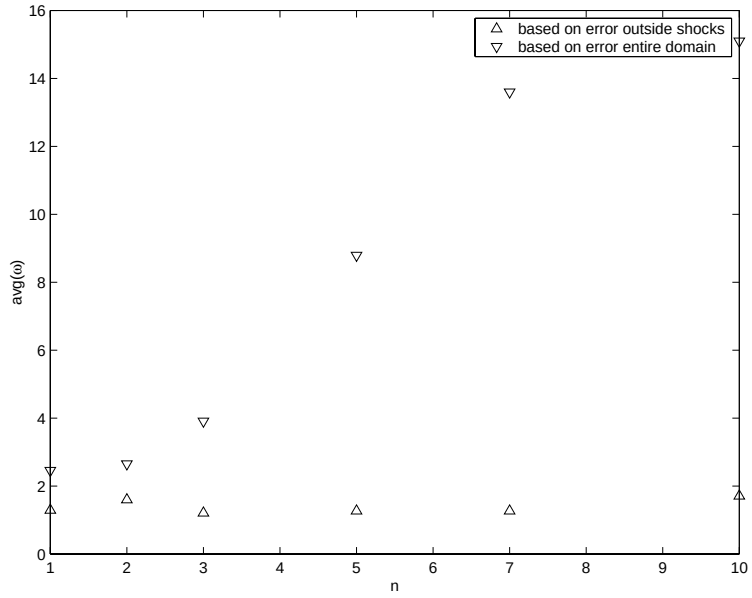

Figure 3.9: The optimal values for $\overline{\omega}$ for various number of shocks n.

domain and number of gridpoints: $\epsilon = 10^{-4}/((b-a)\cdot(NX-3))$. This guarantees that each mesh gets a 'fair chance' to be adapted, independent of the domainsize or number of meshpoints.

The 'best' adaptivities were selected by taking either the smallest error at the entire domain, or the smallest error outside the shock. The following table thus shows two lines for each n:

n	N $(N-3)$	α	Error outside shock		Error entire domain		$\omega_{\max}$	$\overline{\omega}$	$\overline{\omega_{\max}}$
			L_2	L_∞	L_2	L_∞			
1	53 (50)	0.03	2.45e-2	1.97e-2	1.57e-3	1.22e-3	4.39	1.29	3.047
1	53 (50)	0.3	1.84e-2	1.76e-2	5.13e-3	3.63e-3	12.9	2.46	8.72
2	103 (100)	0.02	2.50e-2	1.79e-2	3.12e-3	1.26e-3	6.90	1.60	4.86
2	103 (100)	0.09	1.60e-2	9.46e-3	5.74e-3	2.58e-3	14.2	2.65	10.0
3	153 (150)	0.002	4.90e-2	2.06e-2	2.64e-3	1.24e-3	3.51	1.21	2.77
3	153 (150)	0.1	2.25e-2	1.27e-2	8.32e-3	3.07e-3	22.2	3.91	15.8
5	253 (250)	0.001	6.60e-2	2.64e-2	2.96e-3	1.18e-3	4.06	1.27	3.32
5	253 (250)	0.2	2.65e-2	8.72e-3	1.20e-2	2.79e-3	52.1	8.79	39.3
7	353 (350)	0.0005	7.61e-2	2.47e-2	3.49e-3	1.15e-3	4.02	1.27	3.29
7	353 (350)	0.25	3.13e-2	8.48e-3	1.41e-2	2.71e-3	81.5	13.6	64.8
10	503 (500)	0.001	5.50e-2	1.33e-2	5.68e-3	9.27e-4	7.70	1.72	6.42
10	503 (500)	0.15	3.68e-2	8.12e-3	1.60e-2	2.44e-3	90.3	15.1	75.6

The above values for $\overline{\omega}$ are shown in figure 3.9.

It becomes clear that the choice of optimal adaptivity strongly depends on the kind of error-estimate that is used. For the error outside shocks, $\overline{\omega}$ does not change very much, it remains between 1 and 2. When the error is measured over the entire domain, strong adaptivity results in large $\overline{\omega}$.

As mentioned above, the choice of α also depends on the kind of accuracy that is desired. Whenever an extreme accurate representation of the shock is desired, adaptive meshing definitely proves its worth, and a fairly large α may be chosen. For $N-3 = 384$ and $\alpha = 0.16$, the slope at the shock was over -2000. A uniform mesh would require at least 12500 points to represent this steepness, and probably much more.

3.4.4 Choosing a sensible monitor-function

In the previous section, the relation between the accuracy of one specific monitor-function and its adaptivity parameter α was thoroughly investigated. However, it should be noted that to be entirely correct, it is not the *value* of ω that matters, but the values of ω over the entire spatial domain in relation to each other. In other words: it is the *normalized range* of ω that matters.

This is because the accuracy of the moving mesh solver depends on the speed of moving meshpoints between two successive timesteps, as determined by (3.14).

For example, using the following monitor function

$$\omega = 10 \cdot \sqrt{1 + \alpha \|\nabla u\|_2^2}, \tag{3.59}$$

would produce exactly the same results as the original monitor function in (3.4), that is for equal α. This is because in solving the moving mesh equation (MMPDE) in (3.14), the monitor-values also appear in the denominator, thereby 'normalizing' the monitor values in the nominator.

In the investigation of these monitor functions a baselevel can be introduced:
The baselevel ω_0 of a monitor function ω is its minimum value. More specifically, for ω's that involve first-order derivatives, the baselevel is the value of ω when $u_\xi = 0$.

The reason that (3.4) and (3.59) are in fact the same is that the monitor-value at a certain slope u_ξ divided by the baselevel, is the same for both. Their baselevel is $\omega_0 = 1$ and $\omega_0 = 10$, respectively.

There also exist monitor functions that change the baselevel as well. One example, that will be used in example problem 3.5.2, is the BM-monitor by Beckett and Mackenzie (see for example [5]). In the current context, it is defined as follows:

$$\omega_j \stackrel{\text{def}}{=} \alpha + \left|(u_\xi)_j\right|^{\frac{1}{2}}, \tag{3.60}$$

$$\alpha \stackrel{\text{def}}{=} \int_{\Omega_c} |u_\xi|^{\frac{1}{2}}. \tag{3.61}$$

In this case, not the derivative itself is prefixed with a multiplication factor α, but the entire function ω is placed at a solution-dependent baselevel α. Since this baselevel α is based on the entire spatial domain, this monitor function is a lot smoother by itself, and Becket *et al.* claim that no further monitor smoother is required.

In example problem 3.5.1 the BM-monitor did not prove to be any better than the original monitor with computational derivatives. For more complex solutions, its smoothness may play a more important role.

Another monitor function is a generalized normalization of (3.4):

$$\omega_j \stackrel{\text{def}}{=} \sqrt{1 + \alpha \left(\frac{(u_\xi)_j}{M}\right)^2}, \tag{3.62}$$

$$M \stackrel{\text{def}}{=} \max_{j=0..N-1} \left|(u_\xi)_j\right|, \tag{3.63}$$

where α is a predefined constant, that is usually of order $\mathcal{O}(10)$ for optimal use.

Both monitor functions (3.60) and (3.62) will be used for example problem 3.5.2.

3.5 Example problems

Example 3.5.1 (Burgers' equation) This example is the 1D inviscid Burgers' equation:

$$u_t + \left(\frac{u^2}{2}\right)_x = 0, \quad 0 \le x \le 2\pi, \tag{3.64}$$

subject to the 2π-periodic initial data

$$u(x,0) = 0.5 + \sin(x), \quad x \in [0, 2\pi]. \tag{3.65}$$

The following monitor-function is used:

$$\omega = \sqrt{1 + \alpha u_\xi^2}, \tag{3.66}$$

the choice of α has already been discussed in section 3.4.3.

A shock will be formed in the solution at the critical time:

$$t_c = \frac{1}{\max |u^0(x^0)|}, \tag{3.67}$$

where u^0 is the initial solution. For (3.65) this results in $t_c = 1$.

EXAMPLE 3.5.2 (THE SHOCK TUBE OR RIEMANN PROBLEM) The shock tube problem was initially proposed by G.Sod in 1978 in [35], and can also be found thoroughly discussed in [14]. It is especially interesting since the underlying PDEs resemble the traffic flow PDEs. The problem deals with the one dimensional Euler equations of gas dynamics:

$$\begin{bmatrix} \rho \\ \rho v \\ E \end{bmatrix}_t + \begin{bmatrix} \rho v \\ \rho v^2 + p \\ v(E+p) \end{bmatrix}_x = 0, \tag{3.68}$$

where ρ, v, p and E are density, velocity, pressure and total energy, respectively. The above system is closed by the equation of state

$$p = (\gamma - 1)(E - \rho v^2/2), \tag{3.69}$$

where

$$\gamma \stackrel{\text{def}}{=} \frac{c_p}{c_v} \tag{3.70}$$

is the ratio of specific heat coefficients under constant pressure, c_p and constant speed, c_v (see also: [13]). For an ideal gas, γ is equal to 1.4.

The initial state of the system determines the further evolution of the solution. In this case a one dimensional tube is considered with a diafragm at x_0. This diafragm separates the left and right part of the tube, in which the gas has different velocities, densities and pressures:

$$\begin{array}{llll} v = v_L, & p = p_L, & \rho = \rho_L & x < x_0 \quad t = 0 \\ v = v_R, & p = p_R, & \rho = \rho_R & x > x_0 \quad t = 0 \end{array} \tag{3.71}$$

with $p_R > p_L$ and the two regions contain the same gas.

At time $t = 0$, as the diafragm opens, the pressure discontinuity propagates as a shock wave to the right into the region of lower pressure. Meanwhile a growing region of decreasing pressure propagates as an expansion fan to the left. In addition a contact discontinuity separating the two disturbed gas regions propagates to the right. Figure 3.10 shows a schematic top view of the tube when the gas has evoluted for a short time and the time-trajectories of the shock- and expansion-waves. Originally started with two regions L(eft) and R(ight), now five regions can be distinguished. L and R are the (yet) undisturbed regions at pressure p_L and p_R. Region A is separated from R by a shock wave and contains the disturbed[5] low pressure gas. Region B is separated from A by the contact discontuity and contains the disturbed[6] high-pressure gas. The growing expansion fan region is denoted by C. Flow variables vary continuously in C.

[5]The disturbation of the low-pressure gas was caused by the shock from the initial distribution, propagating to the right.

[6]The disturbance of the high-pressure gas is caused by the expansion fan propagating into the left part of the tube.

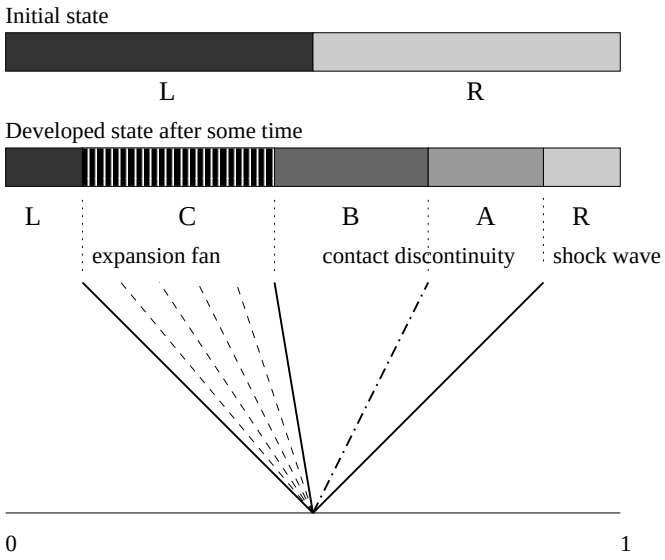

Figure 3.10: Schematic representation of Sod's shock tube, two top-views of the tube, and the time-trajectories of the various waves.

Flow quantities

The Euler equations (3.68) already introduced three flow quantities: the *density* ρ, the *mass flow* (or momentum) ρv, and the *total energy* E. The *flow velocity* v follows directly from the mass flow. The total energy consists of *internal* and *kinetic energy*:

$$E = \rho e + \frac{1}{2}\rho v^2. \tag{3.72}$$

The pressure can be reformulated by rewriting the equation of state (3.69):

$$p = (\gamma - 1)e\rho. \tag{3.73}$$

The entropy is defined as:

$$s = c_v \log\left(\frac{p}{\rho^\gamma}\right). \tag{3.74}$$

Finally, the Mach number is defined as:

$$M = \frac{|v|}{c}, \tag{3.75}$$

where

$$c^2 = \frac{\gamma p}{\rho} \tag{3.76}$$

is the square of the speed of sound.

Exact solution

The analytical solution of this shock tube problem is further discussed in [14]. To determine the accuracy of the numerical solutions, the results of an extraordinary dense discretization ($N = 2000$) are used as a reference solution. A first experiment was performed with the initial data from problem 16.25 in [14]. The analytical pressure in region A is $p_A = 3.0313 \cdot 10^4$. A first test, with only $N = 100$ resulted in $p_A = 3.031 \cdot 10^4$. This is a relative error of $\mathcal{O}(10^{-4})$, so increasing N to 2000 will definitely produce accurate solutions, since the numerical method already proved to be

convergent. For $N = 2000$, the pressure turns out to be $p_A = 3.031304 \cdot 10^4$, which is accurate enough to serve as a reference-solution.

Figure 3.11 shows the variations of the various flow quantities with the initial data specified in (3.77), at time $t = 0.2$. One striking feature is that the pressure has only one shock, whereas the density has two shocks. As the diafragm opens, the high pressure gas propagates to the right, resulting in a pressureshock, the left region loses some pressure because of this pressure shock moving away to the right, resulting in an expansion fan of decreasing pressure. In the region between the fan and the shock (region A and B in figure 3.10), the gas has reached a stabilized level. As mentioned, the density has an extra shock (between regions A and B). This is because in region A, the 'aroused' low-pressure gas has a higher internal energy e, therefore the density ρ is lower. This also follows from (3.73).

Besides the shapes of the various flow quantities, the time-evolution of one specific quantity also gives useful insight into the characteristics of this shock-tube model.

Figure 3.12 shows the density distribution at three different times. The initial shock moves to the right as the diafragm opens, resulting in the rightmost shock. in the left part of the domain gas is extracted, resulting in the expansion fan. Another shock shows between the left and right disturbed regions. The states at $t = 0.05$ and $t = 0.2$ clearly show the 5 distinct regions as discussed before; the left- and rightmost unaffected areas. The two discontinuities and the expansion fan region. The +-marks again show the concentration of meshpoints in regions where the solution is steep.

Numerical solution

The experiments were performed with the following initial data:

$$(\rho, \rho v, E) = \begin{cases} (1, 0, 2.5), & \text{if } x < 0.5, \\ (0.125, 0, 0.25), & \text{if } x > 0.5. \end{cases} \tag{3.77}$$

The spatial domain (the tube) is equal to $[0, 1]$ with the diafragm in the middle at $x_0 = 0.5$. Again an ideal gas is considered ($\gamma = 1.4$).

To simulate an infinite tube, where boundary-effects play no role, a Neumann-condition is imposed upon the left- and right boundary:

$$\left.\frac{\partial u_i}{\partial x}\right|_{\{0,1\}} = 0, \quad \text{for } i = 1, 2, 3, \tag{3.78}$$

where $u_1 = \rho$, $u_2 = \rho v$ and $u_3 = E$.

In [39], Tang and Tang control the adaptivity of the mesh by a monitor function, proposed by Stockie et al. [36]:

$$\omega = \sqrt{1 + \alpha_1 \left(\frac{v_\xi}{\max_\xi |v_\xi|}\right)^2 + \alpha_2 \left(\frac{s_\xi}{\max_\xi |s_\xi|}\right)^2}, \tag{3.79}$$

where α_1 and α_2 are nonnegative constants, here set to $\alpha_1 = 20$ and $\alpha_2 = 100$.

The BM monitor (3.60) is compared to the above monitor function (3.79) and the normalized monitor ξ_{norm} (3.62). Experiments are performed with 50, 100 and 200 points ($N = 53, 103, 203$) and are run until $t = T_{\text{end}} = 0.2$. The experiments are compared by the previously discussed error estimates, both including and excluding the shock-locations. Table 3.1 lists the results[7].

For reference, also two experiments were performed on a uniform mesh. Even for a very small $\mathcal{C}$, no accurate results were possible. The solution becomes unstable; it doesn't explode, but it shows oscillatory peaks. Of course, a larger mesh should be tried as well, but no good results could be obtained at denser uniform meshes within reasonable time.

For comparing, the BM monitor is used as a reference ($\mathcal{C} = 0.5$). The other two monitors have two entries for each mesh-size. One for the same $\mathcal{C}$, and one adapted to result in the same

[7] The results in table 3.1 were obtained from experiments run at a Pentium 4, 1.3GHz. Comparing running times with previous experiments is not possible.

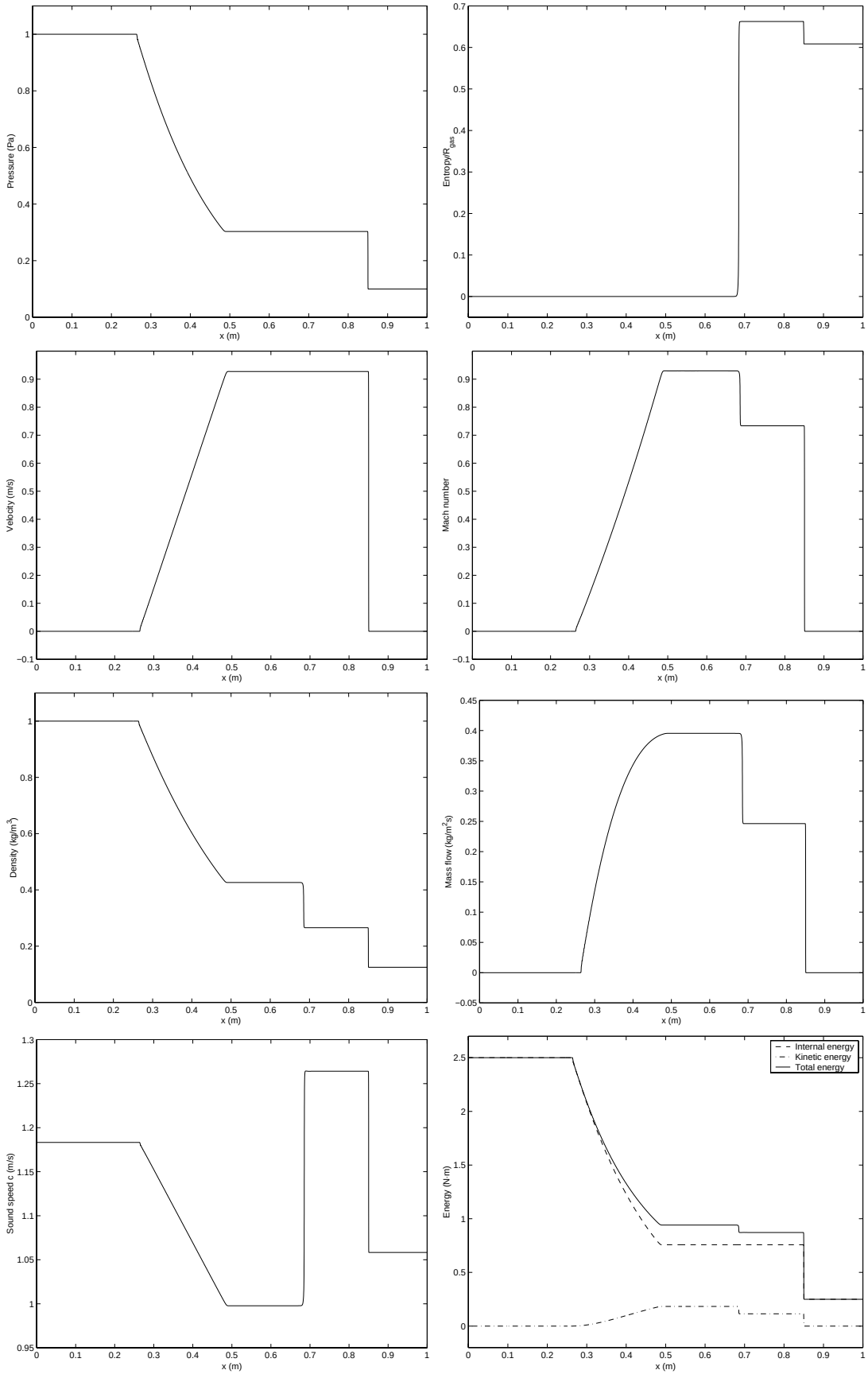

Figure 3.11: Variation in flow quantities at $t = 0.2$.

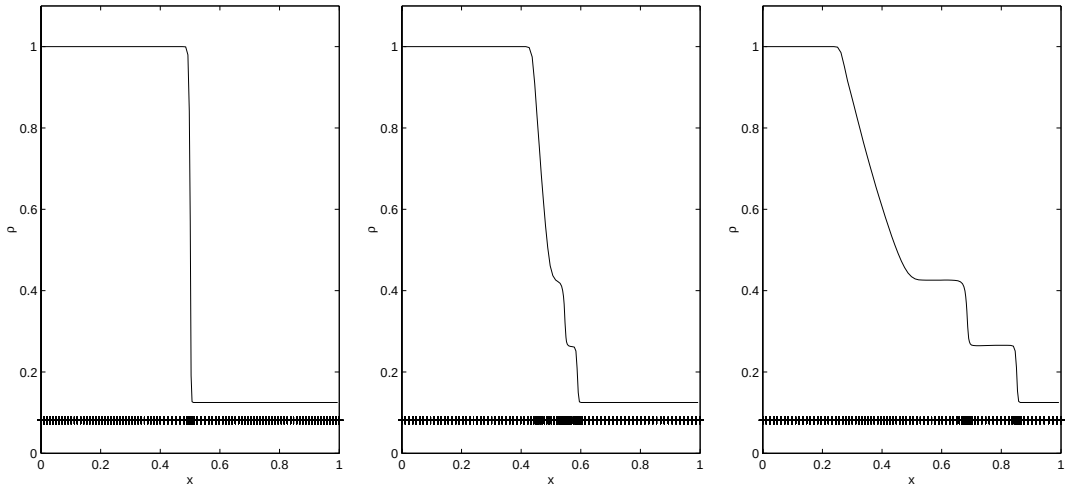

Figure 3.12: Density-solution of the shock-tube problem at time $t = 0$, $t = 0.05$ and $t = 0.2$ respectively.

average timestep $\overline{\Delta t}$ over the entire run. This is done for an honest comparison between the three monitors. Since the BM monitor produces a dense mesh at shocks, the adaptive timestep is also very small, which by itself is also beneficial for the overall accuracy. Decreasing $\mathcal{C}$ for the other two monitors produces approximately the same $\overline{\Delta t}$ as yet.

Comparing BM with ξ_{norm} shows at $N = 53$ a slightly smaller error outside the shock for ξ_{norm}. At the entire domain, BM is more accurate. For the larger meshsizes, BM is always more accurate than ξ_{norm}, at the entire domain even with a factor 2. BM takes a little bit more running time, but the gain in accuracy outweighs these additional costs.

Comparing BM with Stockie shows that BM always produces better results. For all mesh sizes the errors by BM are between 3 and 4 times smaller than by Stockie. Stockie runs about 10% faster, but again this is nothing compared to the difference in accuracy.

Another thing that comes out of the experiments is the convergence order of the solver. This turns out to be quite constant: an order of about 1.5. Apparently the remeshing slightly decreases the convergence order of the solver, which by itself was equal to 2.

Four of the above experiments are shown in figure 3.13. The left column shows the solution for ρ at $t = 0.2$. The right column shows the movement of the mesh. The four experiments shown are the ones with $N = 103$, with the $\mathcal{C}$ value adapted to match the BM experiment ($\overline{\Delta t} \approx 2.5e-4$). The topmost diagrams are for the uniform mesh, and show the instability of the solution.

Especially the mesh movement diagrams show the differences between the various monitors. BM and ξ_{norm} are quite similar, but Stockie is not able to locate all important meshparts very well. The expansion fan hardly receives any more meshpoints than the flat parts of the solution. This is caused by prominent place of the entropy s in the Stockie monitor; $s = 0$ in regions L, C and B. The left diagram shows the effect: the solution shows some minor 'bumps' at the expansion fan, for example at $x \approx 0.4$. Better results may be obtained with Stockie by changing α_1 and α_2. The (rightmost) shockwave is represented moderately. The other two monitors represent the expension fan very well, but they also produce a dense mesh at the contact discontinuity and the shock wave. The (rightmost) shockwave is represented somewhat sharper than the contact discontinuity. Figure 3.11 shows thats for the three model variables ρ, ρv and E the rightmost shock is indeed stronger.
A closer look at the BM and ξ_{norm} monitors shows that BM represents shocks slightly sharper. Of course this is because ξ_{norm} is smoothed and BM is not.

As a final illustration of the BM monitor, it is used in a traffic flow simulation from chapter 4. At each time the minimal, average and maximal monitor value is determined, measured over the

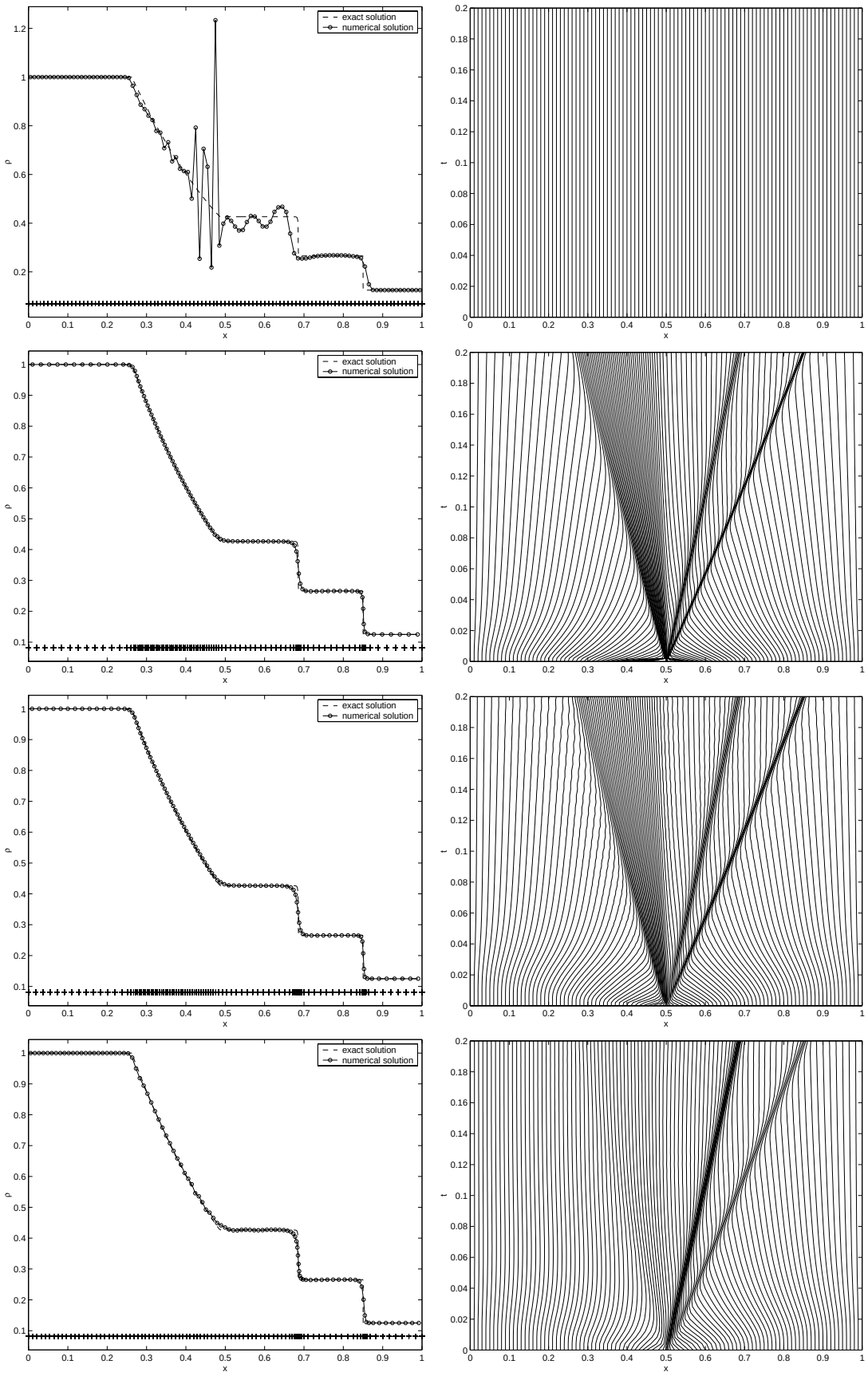

Figure 3.13: Results for the shocktube problem with various monitors. Left column shows the solution at $t = 0.2$, right column shows the mesh movement. All experiments use $N = 103$, and the four monitor functions are, from top to bottom: uniform mesh, BM monitor, ξ_{norm} monitor and Stockie monitor.

monitor	N	$\mathcal{C}$	$\overline{\Delta t}$	exec. time	Err. excl. shocks		Err. incl. shocks		$\omega_{\max}$	$\overline{\omega}$	$\overline{\omega_{\max}}$
					L_2	L_∞	L_2	L_∞			
unif	53	0.01	7.89e-5	139.6	2.50e-2	1.11e-2	2.39e-2	1.11e-2	1.00	1.00	1.00
unif	103	0.01	3.79e-5	648.1	1.77e-2	1.01e-2	1.55e-2	1.01e-2	1.00	1.00	1.00
BM	53	0.5	1.04e-3	13.6	4.16e-3	2.78e-3	1.63e-3	8.06e-4	74.7	5.93	20.6
ξ_{norm}	53	0.5	1.44e-3	8.6	4.38e-3	2.41e-3	2.33e-3	1.27e-3	10.2	3.46	9.13
ξ_{norm}	53	0.37	1.01e-3	11.9	4.02e-3	2.21e-3	2.30e-3	1.23e-3	10.2	3.52	9.15
Stockie	53	0.5	8.89e-4	14.6	6.44e-3	3.40e-3	4.26e-3	2.77e-3	9.45	2.21	8.79
Stockie	53	0.58	1.06e-3	11.7	6.95e-3	4.28e-3	4.38e-3	3.09e-3	9.39	2.19	8.71
BM	103	0.5	2.55e-4	99.4	1.22e-3	7.22e-4	4.46e-4	2.30e-4	149	5.80	34.7
ξ_{norm}	103	0.5	5.45e-4	40.9	1.54e-3	9.29e-4	8.34e-4	3.77e-4	10.2	2.23	9.11
ξ_{norm}	103	0.25	2.35e-4	94.8	1.65e-3	1.08e-3	8.48e-4	4.69e-4	10.2	2.28	9.22
Stockie	103	0.5	3.63e-4	76.5	2.47e-3	1.53e-3	1.38e-3	6.96e-4	9.45	1.64	8.97
Stockie	103	0.37	2.58e-4	85.3	2.79e-3	2.04e-3	1.54e-3	7.65e-4	9.38	1.67	9.06
BM	203	0.5	5.91e-5	884.2	3.37e-4	1.58e-4	1.61e-4	6.56e-5	297	5.66	64.8
ξ_{norm}	203	0.5	2.33e-4	192.4	6.28e-4	3.88e-4	2.93e-4	8.73e-5	10.2	1.58	9.10
ξ_{norm}	203	0.16	6.07e-5	768.9	5.84e-4	2.49e-4	2.96e-4	1.45e-4	10.1	1.61	9.30
Stockie	203	0.5	1.63e-4	294.8	1.08e-3	7.55e-4	5.40e-4	3.06e-4	9.45	1.32	8.99
Stockie	203	0.2	5.94e-5	761.4	9.46e-4	5.49e-4	5.83e-4	3.48e-4	9.49	1.38	9.32

Table 3.1: Experimental results for the shock tube problem with 4 different monitor functions.

entire spatial domain. The time-dependent base level $\alpha(t)$ from (3.61) is determined as well. Figure 3.14 shows the results.

Since at any time, the domain has at least one location in its domain with $u_\xi = 0$, the minimal value of ω is always equal to α, and therefore has been omitted.
Now, α is in fact the average of the gradients

$$\alpha = \left\langle |u_\xi|^{\frac{1}{2}} \right\rangle_\xi , \tag{3.80}$$

where $\langle\cdot\rangle_\xi$ denotes the average at the computational domain, i.e. the 'standard' average without taking variable cellsizes into account. The shown average of the monitor values ω_j is

$$\langle\omega\rangle_\xi \equiv \left\langle \alpha + |u_\xi|^{\frac{1}{2}} \right\rangle_\xi = 2 \cdot \left\langle |u_\xi|^{\frac{1}{2}} \right\rangle_\xi , \tag{3.81}$$

so the average ω is twice the baselevel. This is the reason that the BM monitor works so well. Regions that are (nearly) horizontal still have a monitor value that is only one half of the average monitor value. The very steep parts in the solution still have high monitor values, and this is desirable.

The notion that the $\omega_{\min} = \alpha$ in case one or more points have $u_\xi = 0$, requires some further attention. In case the initial solution is entirely constant, i.e. $(u_\xi)_j = 0$ for all j, the base level α is also equal to zero, as well as all monitor values ω_j. If these monitor values are used during solving the MMPDE (3.8) by Gauss-Seidel iterations (3.14), divisions by zero occur, resulting in a breakdown of the solver. Therefore, it is useful to use a baselevel α' that always has a value unequal to zero:

$$\alpha' = \varepsilon + \alpha, \tag{3.82}$$

where $\varepsilon > 0$ is some very small baselevel.

Conclusion

The insights and results obtained from this example are very useful in further traffic flow experiments, since the Euler-equations (3.68) have a great similarity with the traffic flow PDEs (2.15)

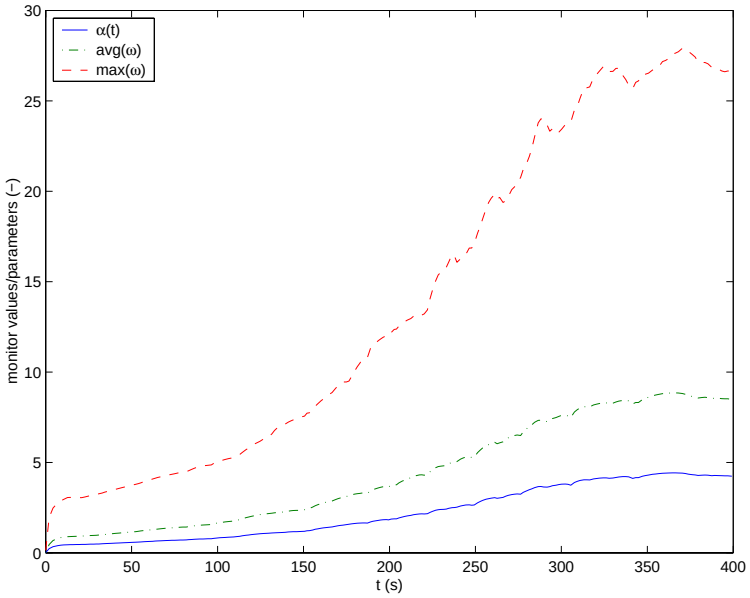

Figure 3.14: Values of BM monitor through time for a traffic flow problem.

and (2.16). Furthermore, the BM monitor has shown to be a very powerful monitor function.

In the first place, it is self-regulating, by integrating the spatial derivative over the entire domain and setting this average value as a base level. This way, no user intervention is required, whereas the other monitors require the user to enter value(s) for α. If any further finetuning for the BM monitor is desired, the base level α (3.61) could be multiplied by some factor to make the mesh more adaptive (multiply by factor< 1) or less adaptive (multiply by factor> 1).

In the second place, the adaptivity is optimally fitted for each timestep. Whereas the prefactor α in the basic monitor function (3.4) is set to a fixed value right before the experiment, the baselevel α of the BM monitor is adapted to the solution each timestep. This is of course much better, since some fixed α that is suitable for remeshing the solution at $t = 0$ might not be so suitable at all for remeshing at some time later on in the experiment.
However, the two other monitors, ξ_{norm} and Stockie also have likewise behaviour. Since the values of the gradients are normalized each timestep, the prefactor α (or α_1 and α_2 for Stockie) can always be aimed at gradient-values between 0 and 1. All three monitors thus have dynamic monitors.

In the third place, there is no need to use a monitor-smoother in combination with the BM monitor. This saves time, and – possibly more important – a monitor that doesn't need a smoothing technique that diminishes its effect is the best monitor anyway.

In the fourth, and not the least place, the BM monitor produces the most accurate results, sometimes by several factors, without considerable timecosts.

Chapter 4

Traffic flow simulation

This chapter brings together the traffic flow models from chapter 2, the techniques from chapter 3 and the software from chapter 5. Various simulations will be performed with Kerners model and Hoogendoorns model, for various problem settings.

From the viewpoint of traffic science, simulations are mainly useful for two reasons:

Validation of the model The results from certain traffic flow simulations can clearly show whether the flow quantities evolve in a sound way. That is, whether the approximations in the PDE formulation of traffic flow yield a realistic description of real-life traffic flow.

Investigation of scenarios Certain scenarios in traffic flow can be tested a lot easier in a computer simulation than in real-life set-ups. A good example is the investigation of effectiveness and calibration of certain traffic management systems, like *ramp metering* and *VMS*[1] or newly developed systems. Also, the effects of incidents or lane-drops can be simulated.

From the viewpoint of mathematics, traffic flow simulations are interesting because of:

Testing the robustness of numerical methods The example problems, treated in the previous chapter were relatively easy systems. They fitted neatly in the prescribed PDE-format. However, traffic flow models are generally more complex and have nonzero source terms. The simulations will show whether the numerical solver is robust enough to handle these 'real' models.

Comparing model characteristics Certain characteristics that come out of the simulations may be compared to models from other fields. Often, results from one field serve as an inspiration for new ideas in other fields.

4.1 Problem settings

4.1.1 Ringroad

The simulations that are shown in the next sections deal with a ringroad. The advantage of a ringroad is that it is the easiest way to handle domain boundaries: periodical boundaries are imposed on the discretized solution. Figure 4.6 shows such a ringroad schematically. In the simulation results no circular domain will be drawn. Quantities or solution characteristics that flow out of the domain at the right, re-enter at the left, and vice versa.

A ringroad is also useful for long-time simulations, since all solution characteristics remain in the domain. In case of a roadsegment, the traffic flows out of the domain at the downstream boundary and doesn't return.

In case no ringroad is used, boundary handling will be specified explicitly.

[1] VMS=Variable Message Signs

4.2 Simulations with Kerners model

The governing equations of Kerners model were specified in (2.1) and (2.2). These can be recast to the matrix-vector format of (3.18):

$$\frac{\partial}{\partial t}[r] + \frac{\partial}{\partial x}[rV] = 0, \tag{4.1}$$

$$\frac{\partial}{\partial t}[V] + \frac{\partial}{\partial x}\left[\frac{1}{2}V^2 + c_0^2 \log r\right] = \frac{1}{\tau}[V_e(r) - V] + \frac{\mu_0}{r}\frac{\partial^2}{\partial x^2}[V], \tag{4.2}$$

where $V_e(r)$ is still the same as in (2.3).

The required parameters for (4.2) and (2.3) are specified in table 4.1 and were taken from Helbings book [11]. The ringroad has length $b - a = 11$km. In the following sections some characteristics of the Kerner model will be introduced and illustrated by simulations.

parameter	value
τ (s)	11
μ_0 (veh.·m/s)	121.111...
c_0 (m/s)	15
r_i (veh./m)	0.042
$r_{\max}$ (veh./m)	0.168
b (-)	0.06
V^0 (m/s)	33.333...

Table 4.1: Used parameters for Kerners model.

4.2.1 Metastable behaviour

Chapter 2 already introduced the concept of stop-start waves and phantom jams. In the following simulations, the occurence of these phenomena is studied in more detail. Since the appearance or absence of these phenomena mainly depends on the traffic density, the initial traffic distribution is set at various different base levels r_0.

Before looking at the appearance of traffic jams, the evolution of a small perturbation in the baselevel is studied in more detail.

A small perturbation

Kerner *et al.* [23] propose a small local perturbation at $x \approx 1/3L$:

$$\Delta r(x) = \Delta r_0 \cdot \left[\cosh^{-2}\left(\frac{160}{L}\left(x - \frac{5L}{16}\right)\right) - 0.25 \cdot \cosh^{-2}\left(\frac{40}{L}\left(x - \frac{11L}{32}\right)\right)\right], \tag{4.3}$$

where $L \stackrel{\text{def}}{=} b - a$ is the roadlength and Δr_0 is the amplitude of the local perturbation.

The density base level is set at $r_0 = 38$ veh./km, and the perturbation amplitude is equal to $\Delta r_0 = 1$ veh./km. Traffic has a uniform velocity of 20m/s. The evolution of traffic is studied only for the first few minutes. Figure 4.1 shows the density and velocity distribution during these first few minutes. The initial small dimple in r at $x = 3.9$km has become even lower (less dense traffic), since the slight bump before it has slowed down the inflowing traffic a little. This bump at $x = 3.7$km has almost disappeared already in the first minute. Behind the dimple, a new bump

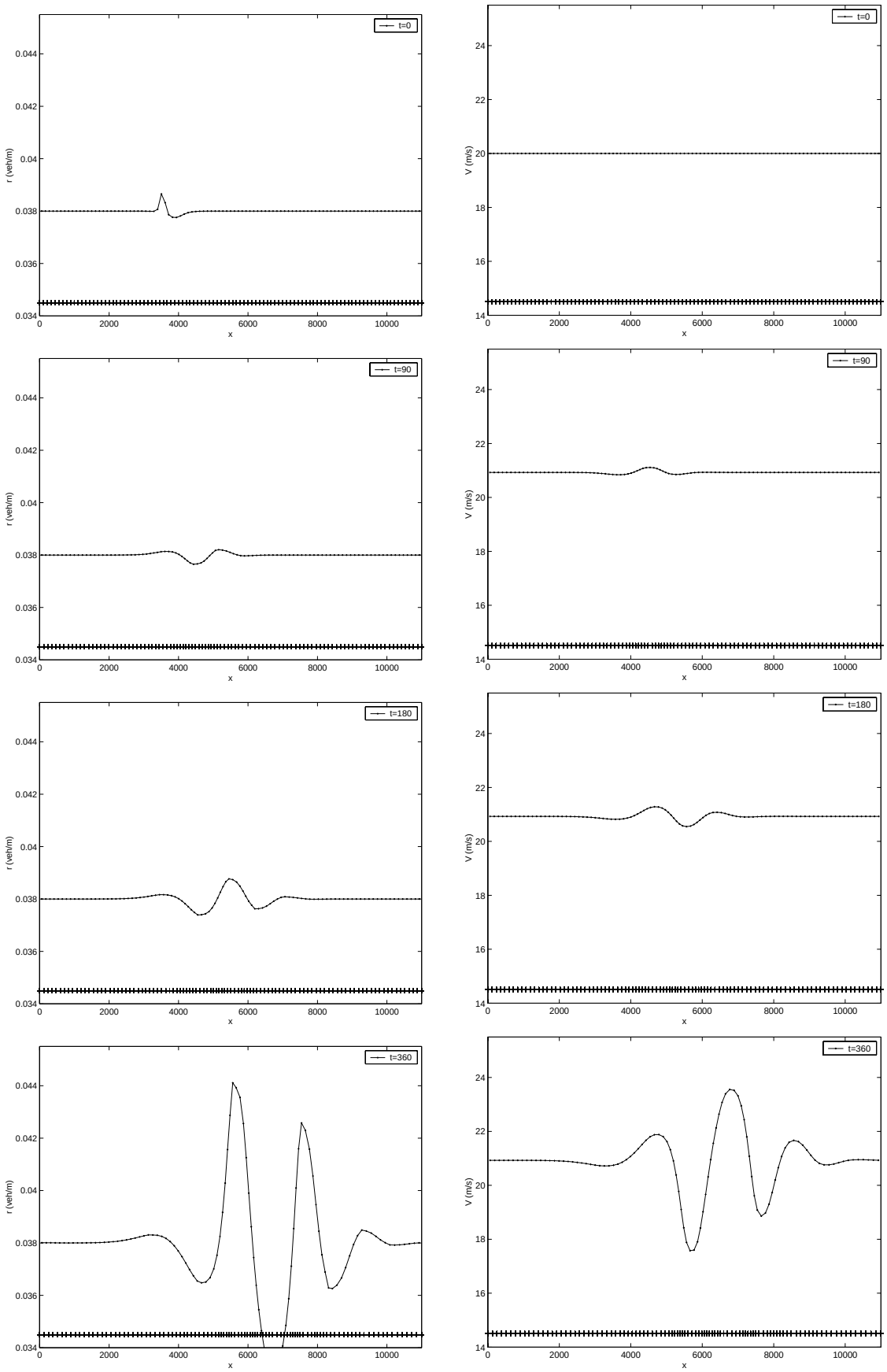

Figure 4.1: Evolution in Kerners model with an initial uniform distribution of 38 veh./km and a small perturbation at 3.7km. Left column shows density at 4 distinct times, right column shows the accompanying velocity.

is formed downstream, caused by the traffic that drives out of the quiet region with relatively high velocity. The structures have moved downstream a bit, because of the convection of the entire flow.

The formation of new 'bumps' of higher density continues between $t = 90$s and $t = 180$s, the bump at $x = 5.5$ km has grown, and a new one is formed at $x = 7$km. Regions of increasing density slow down the inflowing traffic from upstream, resulting in a density decrease downstream. Because of the lower density, downstream drivers increase their velocity, but after only a short distance they catch up with the original constant level of traffic density. Since they have to slow down again, the density at the tail of this constant region increases and a new 'bump' is formed.

It should be noted that the bumps and dimples hardly move (wave speed is nearly zero), for example for $t = 90$s, 180s and 360s, the bump at $x = 5.5$km is one and the same. As time goes by, the density peaks grow bigger and may eventually cause a traffic jam and possibly stop-start waves. In that scenario the structures will start moving upstream (negative wave speed).

In this case, the $r_0 = 38$ is indeed dense enough to cause such phantom jams and stop-start waves, as can be seen in the next simulations.

Phantom jams and start-stop waves

The appearance or absence of traffic jams depends on the base level r_0 of the density and the amplitude Δr_0 of the perturbation. If both are high enough, the small perturbation will grow and will eventually cause a traffic jam that moves upstream.

In the following simulations, again a ringroad of 11km is considered, with a base level of respectively $r_0 = 10$, 38 and 60 veh./km. The perturbation is set to $\Delta r_0 = 1$veh./km. The initial velocity is set at a constant level of respectively $V = 32$, 20 and 5 m/s. The velocities are approximately the equilibrium velocities from (2.3), so that the simulation starts from a situation that is initially stable, except for the perturbation.

Figure 4.2 shows the evolution of the traffic flow (density and velocity) in the x-t plane for the three simulations.

For $r_0 = 10$veh./km, the initial perturbation is visible in r at $t = 0$s. The other two simulations have the same perturbation, but it is not visible because of the much larger scale of the r-axis. At $r_0 = 10$veh./km the perturbation does not cause any waves, let alone jams. It flows downstream with the convection speed of about 32m/s. As time goes by, the perturbation damps away, i.e. the traffic has spread and reached an almost uniform equilibrium state. Due to the structure of the moving mesh, the presence of the perturbation can still be seen at $t = 1800$s, but the amplitude at that time has decreased to 10^{-6}veh./m.

For $r_0 = 38$veh./km, the perturbation doesn't spread out. Instead, it slightly grows, and causes about 3 consecutive waves as was shown in detail in figure 4.1. These waves continue to grow slightly until $t = 7$ minutes. The waves are then too strong to flow convectively downstream. Within 3 minutes the density reaches peaks of 80 veh./km, and the peaks start to move upstream. Eventually densities near 130 veh./km are reached. The density diagram clearly shows the appearance of two phantom jams at $t \approx 400$s near $x = 6$km and 8km. At $x = 10$km also an increasing wave can be found, but it just has the critical density at which it doesn't move downstream, nor upstream. This density wave thus remains at the same location $x = 10$km, and after $t = 11$ minutes, its amplitude doens't grow anymore either. At $t = 1300$s, this wave meets the re-entering congestion wave, and they form a merged congestion wave with even bigger amplitude.
The two congestion waves that are now in the domain are the so-called stop-start waves. Traffic is driving in relatively low densities (22veh./km) at fairly high velocities (31m/s), but when these waves are encountered, the density rises to about 130veh./km and the velocity drops to 1m/s. The traffic thus nearly comes to a stop, and after 1 or 2 minutes it can drive on at full speed.

The bottom two diagrams in figure 4.2 show that at the high initial density $r_0 = 60$veh./km, traffic is not able to moderate the small perturbation. It moves upstream right from the beginning and starts growing.
What strikes first is that the actual traffic jams are formed *later* than for the $r_0 = 38$ case. Here, the strong growth of density takes place near $t = 600$s, so 3 minutes later than for $r_0 = 38$veh./km.

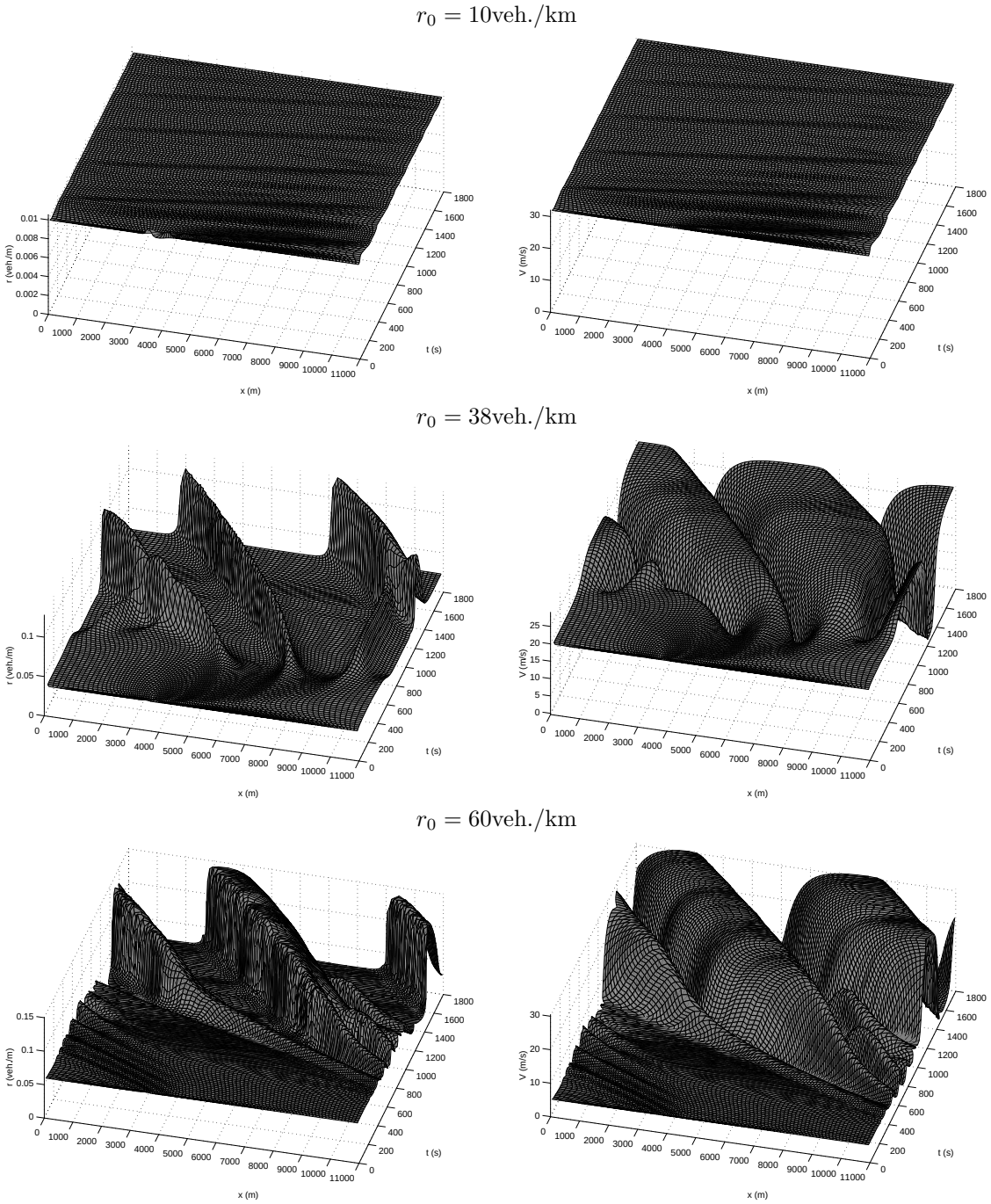

Figure 4.2: The evolution of a perturbation $\Delta r_0 = 1$veh./km at three density base levels r_0. Left column shows the traffic density, right column shows the velocity.

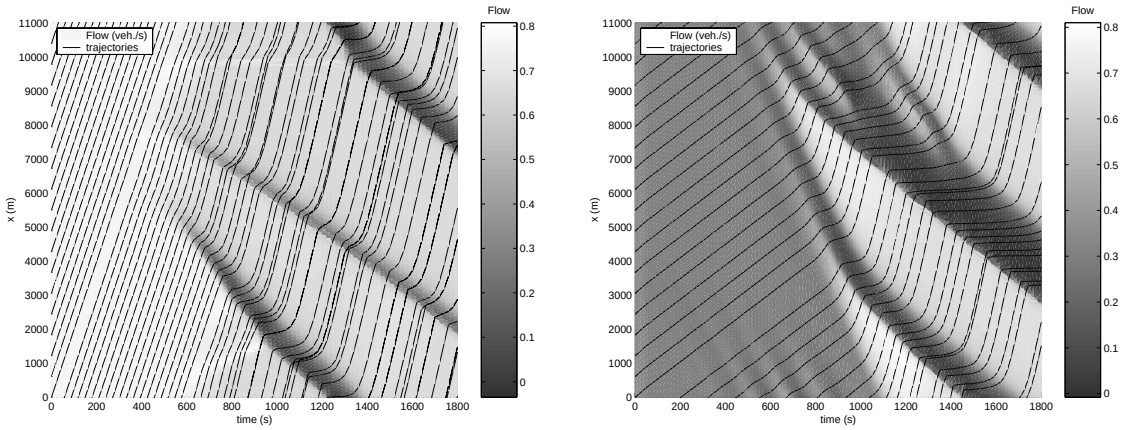

Figure 4.3: The x-t-trajectories and the underlying flow (veh./s) for the simulation with $r_0 = 38$veh./km. (left) and $r_0 = 60$veh./km. (right)

However, the density then increases rapidly and finally reaches values up to 150 veh./km.
Again two congestion waves are formed, but the shape is different from the previous simulation. The congestion waves are no narrow peaks anymore, but fairly wide regions of congestion, up to 2 km long. These are no longer called stop-start waves, Kerner calls these moving jams or *local clusters* [24] instead. It also shows clearly that traffic reaching the tail of the jam forms a shock between the congested and uncongested regions. At the head of the jam, traffic drives out of it more spread.

Trajectories and optimal flow

There is still more information contained in the previous simulation. Figure 4.3 shows the x-t-trajectories and the flow (veh./s) for the last two simulations.

The trajectories were determined by placing 18 imaginary vehicles on the road at $t = 0$, and integrating the computed velocity through time.

A first look at the trajectories for $r_0 = 38$veh./km shows that at $t = 550$s indeed all traffic has left the domain, but also re-entered at $x = 0$, all still at constant velocity of 20m/s. As soon as the congestion waves start to grow, the trajectories are seen to bend a little when velocity changes. At locations of extremely low velocity, the trajectories are nearly horizontal, meaning that for increasing time the location of the vehicle hardly increases. However, these regions are still traversed in a relatively short time: about 1 minute. The drivers are in a congested state of just several 100 meters.
At $t = 600$s, $x = 800$m, a light wave of high flow values is seen to move downstream, until it meets the upstream congestion waves. The same holds for the already mentioned stationary wave at $x = 10$km, beyond $t = 500$s. Apparently the density is slightly larger there, but doesn't affect the velocity too much, resulting in a higher flow.
But now, a difference between the two phantom jams becomes clear. The 'leftmost' jam (at $x \approx 2$km for $t = 1800$s) has a relatively high flow, compared to the other jam ($m = 0.24$veh./s, repectively 0.06veh./s), whereas the densities differ much less (110veh./km, respectively 126veh./km). This is caused by the nonlinear relation between V_e and r (2.3). For optimum usage of a traffic network, the optimal density should be reached, for which the flow is maximal. However, as discussed before in section 2.4, not all density ranges can produce stable states. Figures 2.1 and 2.2 show that the optimal flow belongs to an unstable r. Still, the accentuated stable areas in figure 2.2 are only an approximation, and considering the maximum flow in figure 4.3 (which is almost 0.8 for the small downstream wave at $t = 600, x = 1000$) this optimal value can be reached. Apparently the stable density range is somewhat bigger, and even includes the optimal density.

The right diagram in figure 4.3 shows for $r_0 = 60$veh./km a much darker flow coloring. This is because the initial velocity is only 5m/s. All wave-like structures move upstream and are even darker than in the previous simulation. Since the traffic nearly comes to a stop, the flow tends to zero. The congestion waves are also much wider than before: traffic takes about 7 minutes to traverse the traffic jam. At some fixed t, the with of the jam is seen to be up to 2 or 3 kilometers. This is also the difference between stop-start waves and local clusters: the first are narrow peaks where the upstream and downstream front are wider than the body itself, whereas the second ones are wider regions of congestion, usually several kilometers wide.
The individual drivers however, only drive a distance of about 0.5 kilometer between the tail and the front of the jam. This is because of the negative wave speed of the congestion waves. The drivers of course have to traverse the entire congestion wave, but since the wave speed is higher – in absolute value – than the actual velocities of the vehicles, the vehicles do not drive very far before the the head of the jam reaches them.

Critical density range

The characteristics that are studied here, appear in simulations that were started with very specific initial conditions. Putting aside that these initial conditions will probably never occur in real traffic distributions, the defined initial states can be more or less stable from the start. Figure 2.2 already showed that there is an unstable density range, at which no equilibrium states can occur. It is for this reason that in the $r_0 = 38$veh./km variant from the simulations before, the formation of traffic jam is at an earlier time ($t = 420$s) than for $r_0 = 38$veh./km ($t = 600$s). $r_0 = 38$veh./km is just a very unstable density.

Figure 2.2 also indicates that for even bigger densities, the situation becomes more stable again. In other words: A suitable density $r_0 = r_j$ for causing traffic jams is not only required to be bigger than some treshold (or boundary) value r_b, but also less than some critical density r_{cr}:

$$r_b \leq r_j \leq r_{cr}. \tag{4.4}$$

Kerner *et al.* [22, 23, 24] have performed much quantitative analysis on the values of these r_b and r_{cr} and their dependence on the various model parameters. No such analysis will be performed here, but a qualitative study of this critical density range will be made.

The used parameters are slightly adapted, as specified in table 4.2. The parameters were taken from Kerner [24] and a ringroad of 31km is considered. This way, the perturbations do not quickly influence the other (periodical) side of the domain.

parameter	**value**
τ (s)	5
μ_0 (veh.·m/s)	58.333...
c_0 (m/s)	11.07
r_i (veh./m)	0.0427
$r_{\max}$ (veh./m)	0.18
b (-)	0.04
V^0 (m/s)	28

Table 4.2: Used parameters for Kerners model.

Two simulations were run, for $r_0 = 60$veh./km and $r_0 = 72$veh./km, with initially uniform velocities of respectively 2 and 1 m/s. Figure 4.4 shows the results for these runs in an $r(x,t)$

diagram, the solution $r(x)$ at $t = 1000$s, and the fundamental r-V diagram, with a certain part of particular interest strongly magnified.

The 3D diagrams immediately show that the base level $r_0 = 72$veh./km is stable enough to let the perturbation damp away. Of course, it moves upstream, but it doesn't cause any congestion waves (Note the difference in r-scaling in the $r(x)$ diagrams). In the less denser traffic ($r_0 = 60$veh./km) a traffic jam *is* formed.

First concentrating on $r_0 = 72$veh./km, the $r(x)$ diagram shows that at the end time $t = 1000$s, the density distribution is indeed almost uniform, and the $V(r)$ fundamental diagram thus shows only one state of $r = 72$veh./km at a very low velocity. A more detailed look in the bottom diagram still shows some specific driver behaviour. The circles are the r-V pairs at all meshpoints, the arrows connect pairs of two consecutive meshpoints (downstream). First, 10 points lie at $(0.072, 0.4704)$. This doesn't show from the bottom diagram, but they are the first 10 meshpoints before the perturbation, as the $r(x)$ diagram shows. Next, drivers see a (small) density peak ahead of them, and start decelerating. This is mainly caused by the 'traffic viscosity' term $\partial_x^2 V$. Since drivers decelerate even before the density starts to grow, the circles lie underneath the equilibrium line (V has decreased, but r has not yet fully increased). Then, as the density has reached its maximum value (7 or 8 circles further), i.e. the drivers are on top of the perturbation, the drivers accelerate again, since they see a relatively strong decrease in density ahead of them. The arrows now show a fast transition from this state to the state of low density (bottom of the perturbation). The drivers decelerate again, since that last density increase lies ahead of them. After they have traversed that part, drivers keep their velocity, since the density is constant. This is shown by the many circles that overlap at $(0.072, 0.4704)$.

A same kind of analysis can be performed for $r_0 = 60$veh./km, but first of all, the $r(x,t)$ and the $r(x)$ diagram already show some special characteristics. The perturbation of course moves immediately upstream, but at $t \approx 300$s, a traffic jam occurs that becomes wider and moves upstream slower than the original perturbation. Behind the jam (upstream) a growing region of relatively low density forms, and downstream the original $r = r_0 = 60$ remains intact (although it is a decreasing region, because of the periodical boundaries). These three states also show clearly in the fundamental diagram. Many circles lie near $r = 26$veh./km, many at $r = r_0 = 60$veh./km and also a few near $r \approx 220$veh./km. This effect was discovered by Kerner and is called the dipole effect. It occurs in a certain critical density range, for fairly high density values.
The bottom diagram shows a detail of $V(r)$ in the region of lowest density. Drivers see a huge traffic jam appear ahead of them, and therefore decelerate heavily, even before the density increases. At the jam itself, the density is even bigger than the maximum density, wich indicates that traffic has not reacted strong enough to set velocity to 0m/s before $r_{\max}$ was reached. In this situation the circles also move quickly back to the $r = 60$veh./km state. From this point, after quite some time the transition back to the $r = 26$veh./km is made.

Connecting the three states and neglecting the few transition points, and placing this in an $m(r)$ fundamental diagram shows the typical structure of this dipole effect, see figure 4.5. The three states have been labelled: $r_{\min} = 26$veh./km, $r_0 = 60$veh./km and $r_{\max} = 220$veh./km.

The transition layer between $r_{\min}$ and r_0 moves upstream (the dashed line in the left diagram has negative slope). This also holds for the two other transition layers. Since $\Delta m/\Delta r = V$, the exact wave speed of the three layers can be computed. Table 4.3 shows the three transitions and their corresponding wave speeds. This shows that the head of the jam moves upstream slowly.

transition	**wave speed (m/s)**	**line style**
$r_0 \longrightarrow r_{\min}$	-15.7	dashed
$r_{\min} \longrightarrow r_{\max}$	-3.61	dash-dotted
$r_{\max} \longrightarrow r_0$	-0.903	dotted

Table 4.3: Three transitions in the dipole-effect, and their corresponding wave speeds.

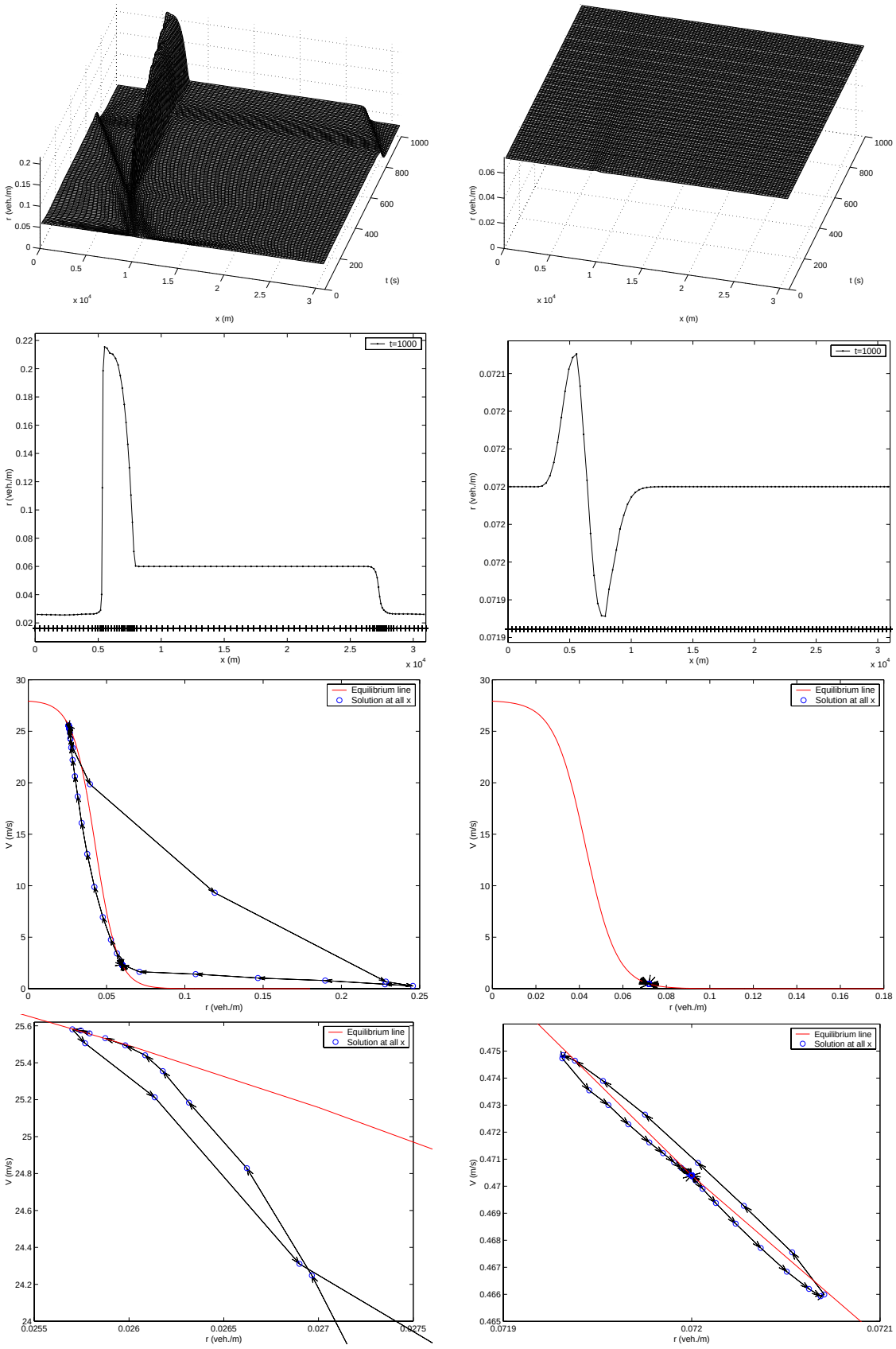

Figure 4.4: Evolution in Kerners model with an initial uniform distribution of 60 veh./km (left) and 72 veh./km (right) and a small perturbation at 10km. The 4 diagrams show respectively $r(x,t)$ in 3D, $r(x)$ at $t = 1000$s, the $V(r)$ fundamental diagram and a detail of this diagram.

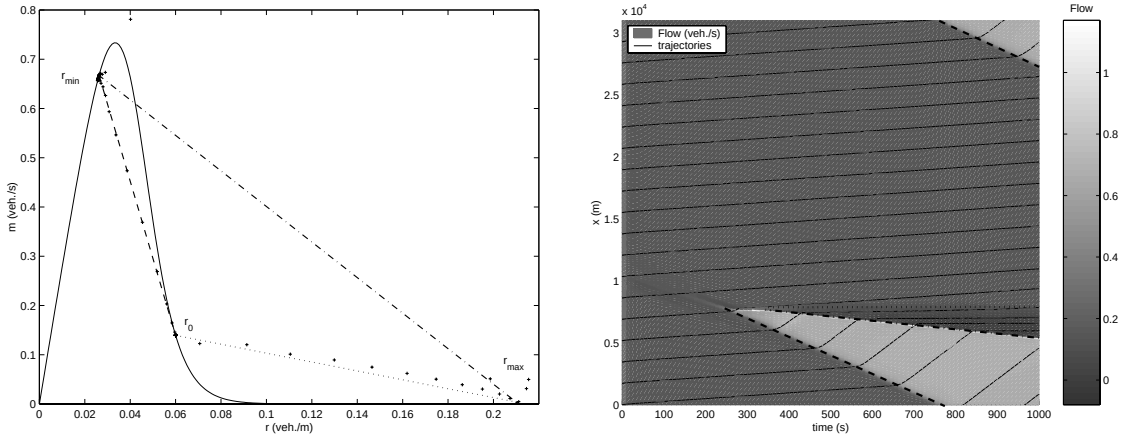

Figure 4.5: Dipole effect in Kerners model for the perturbation of $r_0 = 60$. Left: fundamental $m(r)$ diagram, right: x-t trajectories and flow.

Its tail moves upstream faster, so the jam has a width that grows with $-0.903-(-3.61) = 2.71$m/s. The region of minimal density grows with $-3.61-(-15.7) = 12.1$m/s.

These speeds can also be seen in the right diagram of figure 4.5, the transition layers have been accented by a fat line, table 4.3 also mentions the linestyle.

Variable Speed Control

To investigate the efficiency of certain speed-regulating measures a ringroad of 6km is considered with a very dense region ('blockade') in the middle. Figure 4.6 represents the domain schematically.

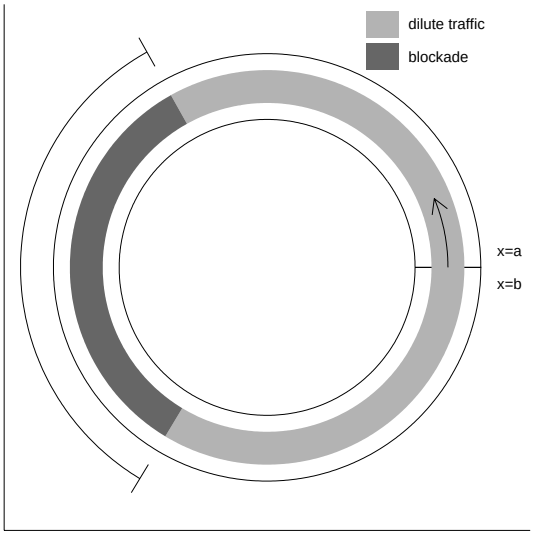

Figure 4.6: Schematic representation of a ringroad with a blockade in the middle.

Hoogendoorn [17] has studied the effect of Variable Message Signs (VMS). VMS are placed at regular distances (here: 500 m) above the highway. They receive measures of average velocity from detectors 200m ahead of them. If the measured average velocity is below 50kmh, The VMS

displays a speed limit of 50kmh. At an average velocity between 50kmh and 70kmh the speed limit is set to 70kmh, and between 70kmh and 90kmh the speed limit is set to 90kmh.

Figure 4.7 shows the evolution of the blockade without VMS, and with VMS enabled.

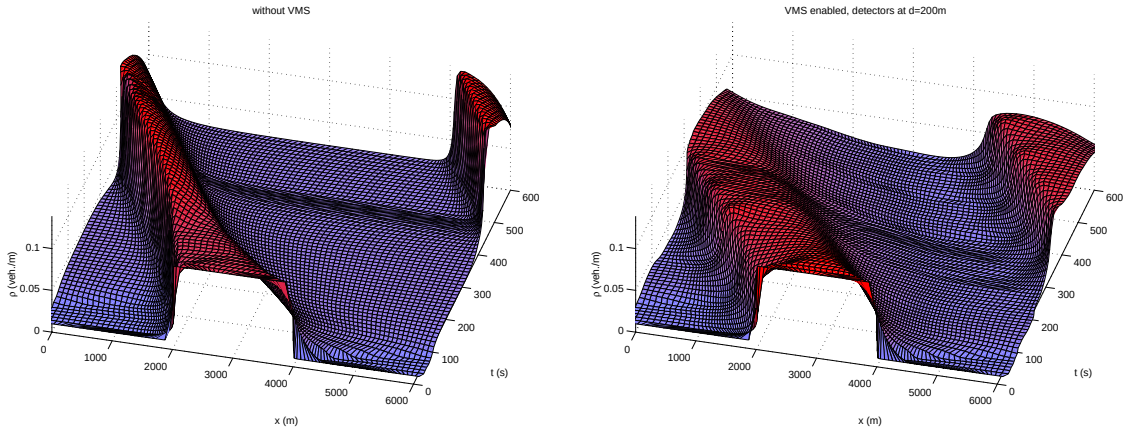

Figure 4.7: Evolution of a blockade. Left: without any traffic management, right: with VMS enabled.

When VMS is disabled, at the tail of the blockade an even more serious traffic jam occurs, because of the inflowing traffic. Densities beyond 150veh./km are reached. Still, at the head of the blockade, traffic is able to drive downstream quite fast.

When VMS is enabled, the density does not grow, since the drivers that approach the blockade adapt to the speed limit. However, at the blockade itself, most drivers, except the last few are also restricted to 50 kmh. Therefore, the blockade spreads out very slowly. This is also a theoretical property of the VMS system: if all drivers obey the speed limits, nobody re-accelerates after he has passed the dense region.

However, this example was for illustrational purpose only. Hoogendoorn has added an 'inobedience'-term that causes some of the drivers to drive faster than the speed-limits. This eventually causes the speed-limits to be increased, so that the rest of the traffic can accelerate as well.

4.3 Simulation with Hoogendoorn

The following experiments were performed with one user-class ('slow person car'), with the parameters in table 4.4.

4.3.1 Instability

Some of the first experiments with Hoogendoorns model and the developed solver showed some unexpected unstable behaviour. As an example, consider the following initial conditions on a ringroad of 11 km:

$$r(x,0) = 0.01 + 0.001\sin(2\pi \cdot x/11000), \quad (4.5)$$

$$m(x,0) = 0.25. \quad (4.6)$$

The above initial functions denote a baselevel of 10 veh./km, perturbed by a sinus-like function with an amplitude of 1 veh./km, and a uniform flow of 0.25 veh./s.

Normally this perturbation would damp away, or at least move convectively along with the traffic flow, since the base level of 10 veh./km is very low. The MMFV solver however, produced unexpected results as can be seen in figure 4.8.

parameter	**slow person car**
V_u^0 (m/s)	27.5
$\hat{r}_{\max}$ (veh/m)	0.2
τ_u (s)	10
T_u (s)	1.4
l_u (m)	3.7
d_u (m)	1.0
α_0 (-)	0.007
$\delta\alpha_0$ (-)	0.03
r_c (veh/m)	0.04
δr_c (veh/m)	0.006

Table 4.4: Used parameters for Hoogendoorns model.

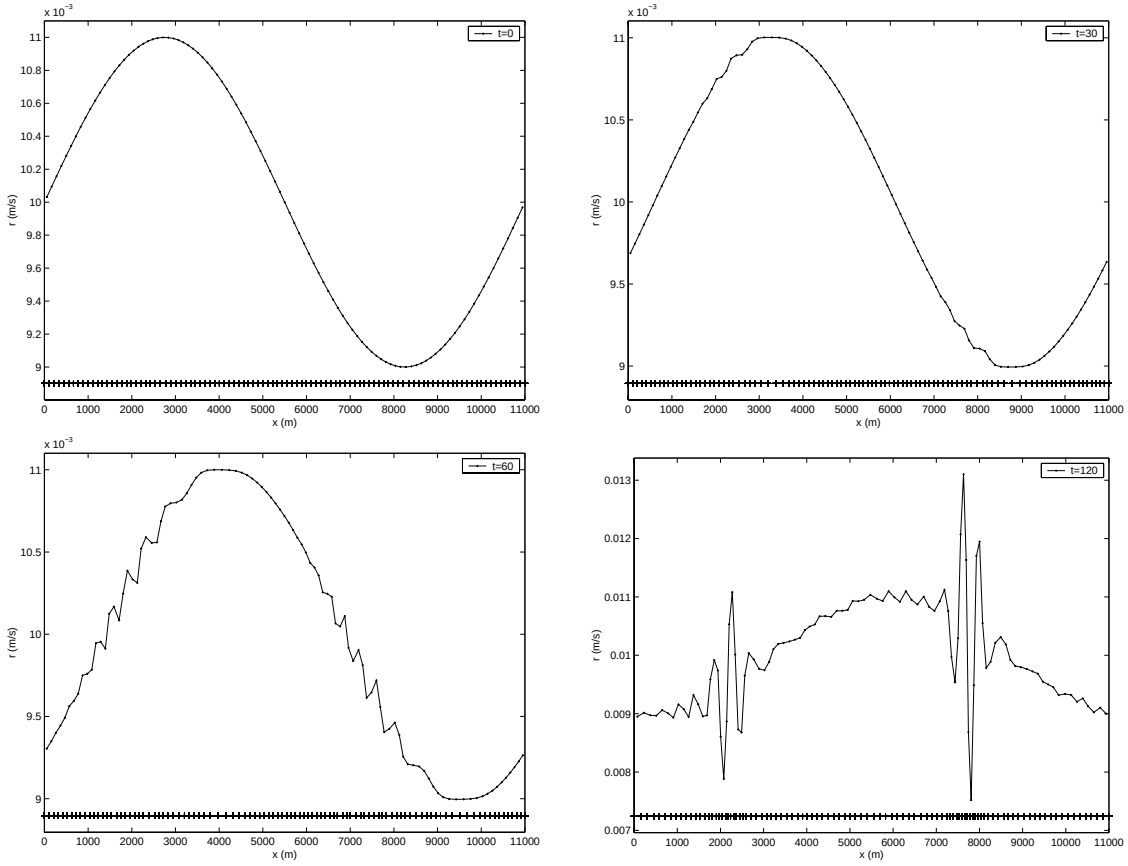

Figure 4.8: Instability of MMFVPDES with Hoogendoorns 2-PDE model. An initial smooth solution blows up for increasing time ($t = 0, 30, 60, 120$s).

Just before the extrema of the sinus, small waves appear (see the solution at $t = 30$s), This is

in a region where second-order derivatives are at their maximum. The original PDEs (2.15) and (2.16) do not contain 2nd-order derivatives, but the Jacobian f_u does contain all kinds of chained derivatives.

4.3.2 McCormack solver

To see, whether it is indeed the Jacobian that causes instability, a simple McCormack solver has been implemented as a substitute to the Finite Volume MUSCL-solver. The second-order accurate McCormack solver as described in [13] is used:

$$\overline{u_j^{n+1}} = u_j^n - \frac{\Delta t}{\Delta x_{j-1}} \left(f_j^n - f_{j-1}^n \right), \tag{4.7}$$

$$u_j^{n+1} = \frac{u_j^n + \overline{u_j^{n+1}}}{2} - \frac{\Delta t}{2\Delta x_j} \left(\overline{f_{j+1}^{n+1}} - \overline{f_j^{n+1}} \right). \tag{4.8}$$

The difference is that McCormack does not use an advanced flux-term, but a simple flux $\partial f / \partial x$.

The McCormack solver does not show the unstable behaviour as in figure 4.8, no further diagrams of this are shown. McCormack has, however, a major disadvantage compared to MMFVPDES : it can not handle shocks very well. Taking another sinusoidal initial distribution, now with a base level at 20 veh./km, a perturbation amplitude of 5 veh./km and a uniform flow of 0.35 veh./s, a steepening wave emerges. This can be compared with the observed results for Burgers' equation in chapter 3. Figure 4.9 shows the effect of this emerging shock on the McCormack approach.

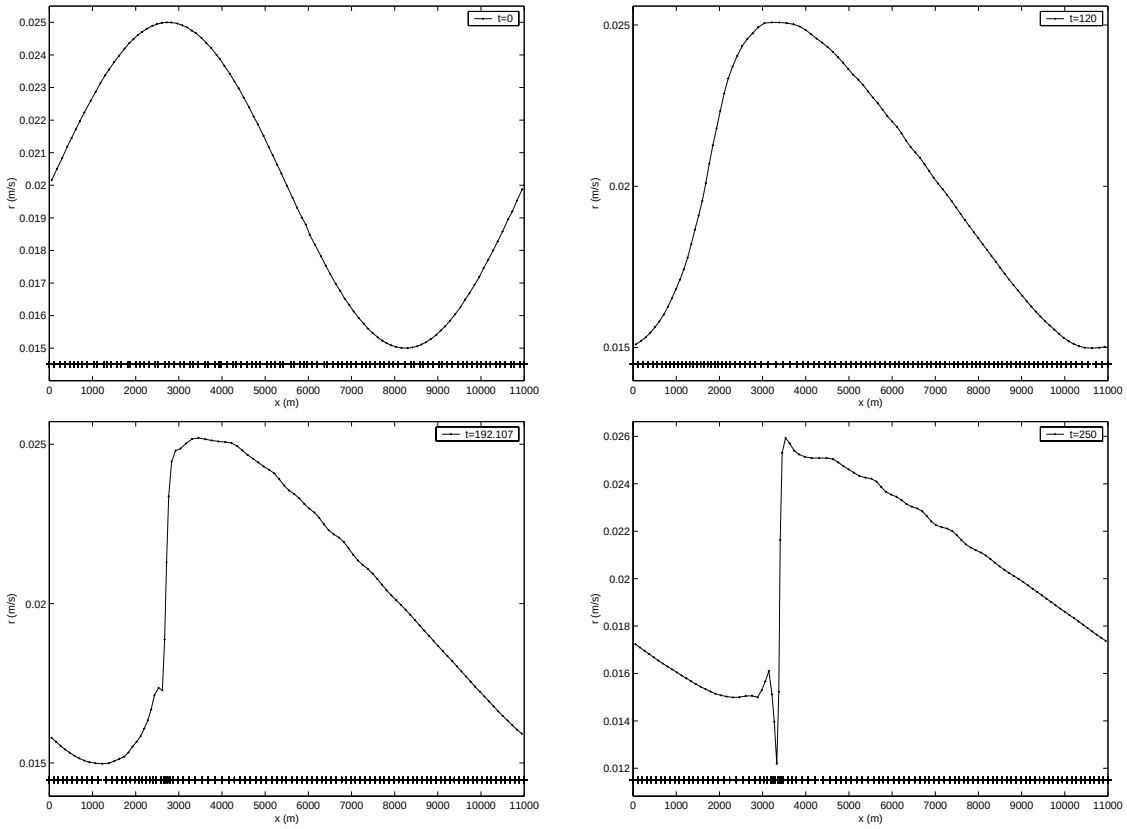

Figure 4.9: Effect of emerging shock on McCormack solution at $t = 0, 120, 192$ and 250s.

The first two diagrams show the steepening wave, which is smoothly represented by the McCormack solver. When, near $t = 192$s, the shock has formed, the solution starts to contain sharp

peaks, which is even more clear at $t = 250$s. The solution then also contains additional small waves.

4.3.3 Conclusion

Although numerical experiments in chapter 3 and practical experiments in section 4.2 proved the solver and simulator to be quite successful, the conservative 2PDE formulation by Hoogendoorn can not be simulated very well with it. A part of the problem lies within the computation of the Jacobian which can lead to instabilities. Applying the McCormack solver avoids this issue, but McCormack is unstable at shocks.

Another observation during these experiments was the heavy fluctuation of the finite-space correction factor ζ. It turns out to be very important to choose equilibrium initial conditions for Hoogendoorns model, since ζ is not automatically limited to be always bigger than 1.

In general, further research is necessary to identify the terms that cause the observed instabilities, both in the traffic flow PDEs, and in the numerical schemes.

Chapter 5

TraFlow$_{\text{PACK}}$ software documentation

The numerical PDE solver MMFVPDES (algorithm 1) and the traffic flow simulator TRAFLOW have been implemented in MATLAB . MATLAB comes with extensive built-in support for mathematical operations and scientific visualizations. Furthermore it allows flexible programming with all valid control structures, functions and classes. Finally, MATLAB still can run programs with high performance, especially when the code is optimized[1]. For all these reasons MATLAB was chosen as developing-environment.

One of the purposes of this research (see also section 1.1) was to develop a simulation/solving tool that is suitable for various models. Also various parameters for both model, problem and solver should be easy configurable. Therefore an approach with 'settingsfiles' was chosen. Settingsfiles are plain MATLAB files with a predefined structure, in which the required information can be specified. When running the solver or simulator, the user only has to specify the name of this settingsfile to pass all required information to the program.

In the next sections the two programs are separately introduced and documented. Further information on the program-code can be found in appendix A.1.

The latest information and documentation of possible new releases of TRAFLOW$_{\text{PACK}}$ can be found at http://www.inro.tno.nl/five/traflow/. The documentation in this report describes version 1.0.

5.1 MMFVPDES : the numerical PDE solver

mmfvpdes.m implements the PDE solver in algorithm 1. The internals and mathematical properties have already been discussed in chapter 3. There are a number of control-parameters, some of which can be set at the MATLAB -prompt and others are specified in the settingsfile.

5.1.1 Commandline syntax

The way to invoke mmfvpdes from the MATLAB -commandline is as follows:

> varargout = mmfvpdes(action, f_settings, N, Tstart, Tend,
mm_tol, *mm_maxit*, *mm_nsmooth*, *cflC*, *solverId*, *timesample*, *plotIdx*, *saveAll*, *errorCheck*)

The *slanted* parameters are optional. Explanation of the parameters:

action: Indicates what task the solvers should perform; one of the following:

[1] MATLAB -code can be optimized, for example, by vectorizing loops. Further information on this and other optimization techniques can be found at the official website (http://www.mathworks.org). Search the technical documentation for 'optimizing code'.

- *'comp'* Computes the solution u and returns it at all x-points and all sampled times: varargout = {MM_ALLT, MM_ALLX, MM_ALLU};
- *'plot'* Computes the solution u and shows it on the screen. No values are returned.
- *'exact'* Computes the solution u and stores u at Tend in a file mmfvexact.mat, together with the x-mesh at t = Tend. No values are returned.

f_settings: The name of the settingsfile without the .m suffix.

N: The number of meshpoints. The domain, including boundaries, contains N−2 points. Outside the domain 4 extra points are used. see also figure 3.2.

Tstart: The start-time of the PDE-evolution. This allows the initial solution or parts of the PDEs to depend on time t.

Tend: The end-time of the PDE-evolution.

mm_maxit: The maximum amount of Gauss-Seidel iterations (3.14) for solving the moving-mesh equations. Default: *3*.

mm_tol: The tolerance-level for preliminary interruption of the Gauss-Seidel iterations. Default: *1E-6*.

mm_nsmooth: The number of smoothing-iterations to use for smoothing the monitor-values. Default: *0*.

cflC: The constant C in eq.(3.35) for fitting the maximal timestep to the CFL-condition. Default: *0.5*.

solverId: Indicates which solver to use:

- *1* MUSCL-type finite volume solver, most accurate and powerful.
- *2* McCormack finite volume, second-order accurate, but unstable at shocks.
- *3* Forward-time upwind finite differences, for reference only; usually inaccurate or unstable.

Default: *1*.

timesample: The amount of time between two successive timesamples. u is not stored at each time-instant t since this would mostly produce unnecessary large datasets. Default: *1*.

plotIdx: In case u is multidimensional (a system of PDEs): the variable that should be used for plotting (i.e. only used whenever action=*'plot'*). Default: *-1* (show all variables, in subwindows).

saveAll: Whenever set to *'save'* all sampled solution-values, mesh-values and times are stored in a .mat file which can be reloaded for future analysis.

errorCheck: Whenever set to *'errorcheck'* some error-diagnostics are computed and displayed. A file mmfvexact.mat is required for this.

Details on the second argument, the settings-file can be found in the next section.

An example call for the shocktube-problem (example 3.5.2) could be:

Example:

```
> mmfvpdes('plot', 'settingsST', 200, 0, 0.2, 3, 1e-6, 0, 0.5, 1, 0.02, 1, 'save', 'errorcheck');
```

In the next paragraphs, the above example is further exemplified by showing the source of the settingsfunction and other required MATLAB -code.

5.1.2 Settingsfile syntax

To let the solver know *what* specific system of PDEs it has to solve, it is passed a settingsfile that contains this information.

First, the solver was designed for a general (system of) hyperbolic PDEs with possible non-zero source term at the right (3.38). Therefore the solver should be provided with a specific implementation of the functions f and g. Besides that, the solver also needs an implementation of the Jacobian of f, i.e. f_u.

Second, the solver needs to know some information on the specific problem that should be solved within this model. This information includes the size of the domain $[a, b]$, a function producing an initial solution and the function that handles the boundary conditions.

Third, to be able to move the mesh, the name of the monitorfunction should be specified.

The above information should be put in a MATLAB -cell-array, and returned by the settings-function. The solver evaluates the settings function as follows:

> [MM_A, MM_B, f_init, f_boundaries, f_monitor, f_f, f_dfdu, f_g] = feval(f_settings)

Example:

```
function [MM_A, MM_B, f_init, f_monitor, f_f, f_dfdu, f_g, f_boundaries]=settingsST()
MM_A = 0;
MM_B = 1;
f_init = 'initST';
f_monitor = 'monitorBM';
f_f = 'fEuler';
f_dfdu = 'dfduEuler';
f_g = 'gEuler';
f_boundaries = 'neumann_sol';
```

5.1.3 Model- and problem-functions syntax

The solver now knows where to find (or how to call) the various functions that will compute needed values. These functions have to be implemented by the user. Each function is passed a predefined set of parameters, and the solver will make certain variables available in the global workspace, so that the user-functions are able to use these variables. The names of all globally available variables have been prepended with 'MM_' by the solver:

MM_NX: Amount of meshpoints, as specified by the user (MM_NX = N). See also figure 3.2.

MM_A: The x-coordinate of the left boundary of the spatial domain.

MM_B: The x-coordinate of the right boundary of the spatial domain.

MM_NU: The number of PDEs, or equivalently, the number of model-variables (e.g. MM_NU = 3 in (3.68)).

The solver can store the computed solutions in globally available matrices:

MM_ALLT: The time-values at which a sample of the solution was stored. MM_ALLT = [Tstart, $\ldots, t_k, \ldots$, Tend]. For now, let's call the amount of timesamples nt = length(MM_ALLT).

MM_ALLX: The dynamic meshcoordinates x_j^k at the sampled times. size(MM_ALLX) = [nt, MM_NX+2].

MM_ALLU: The computed solution at all sampled times and all x-points. size(MM_ALLU) = [MM_NU, nt, MM_NX+1].

Some variables thay may be used in the following paragraphs are generally defined as follows:

xa: $1 \times (\mathsf{MM\_NX} + 1)$-vector containing $x_{\frac{1}{2}}, \ldots, x_{n+\frac{3}{2}}$.

dx: $1 \times (\mathsf{MM\_NX} + 1)$-vector containing $\Delta x_{\frac{1}{2}}, \ldots, \Delta x_{n+\frac{3}{2}}$. See also (3.13).

Any other parameters needed inside the user-defined functions can best be declared global and assigned a value, prior to invocation of the solver.

Function for initial solution

The f_init function is called by the solver as follows:

> uo = feval(f_init, xa);

f_init should return a vector uo of the same length as xa, of which each element has dimension MM_NU.

Example:

```
function u = initST(x)
% Besides setting the initial solution, it is also possible
% to define some other needed parameters:
global gamma;
gamma = 1.4;

% Produces a step-like function with shock at x=0.5
il = find(x < .5);
u(1:3,il) = [1;0;2.5] * ones(1,length(il));

ir = find(x >= .5);
u(1:3,ir) = [0.125;0;0.25] * ones(1,length(ir));
```

Function for monitor values

The f_monitor function is called by the solver as follows:

> w = feval(f_monitor, u, xa, dxi);

f_monitor should return a vector w of the same length as u (i.e. MM_NX+1), of which each element denotes the monitor-value at that specific point. The following function implements the simple monitor function (3.66).

Example:

```
function w = monitorxi(u, x, dxi)
global MM_NU;
global alpha;

t = MM_NU / length(alpha);
alpha = reshape((alpha*ones(1, t))', 1, MM_NU);

dudxi = ddxi(u, dxi);

w = sqrt(1 + alpha * dudxi.^2);


function dudxi = ddxi(u, dxi)

n = size(u, 2); % array-length
dudxi(:,2:n-1) = (u(:,3:n) - u(:,1:n-2)) /(2.0*dxi);
```

Function for f

The f_f function is called by the solver as follows:

> f = feval(f_f, u(1:MM_NU, 3:MM_NX);

f_f should return a vector f of the same length as its input-vector u (i.e. MM_NX − 2), of which each element has dimension MM_NU (since f_f implements the vector function $\vec{f}$).

Example:

```
f(1,:) = u(2,:);

v = u(2,:) ./ u(1,:);
p = pEuler(u);

f(2,:) = u(2,:).* v + p;

f(3,:) = v .* (u(3,:) + p);


%%%%
function p = pEuler(u)
global gamma;

p = (gamma - 1) * (u(3,:) - u(2,:).^2 ./ u(1,:) / 2);
```

Function for f_u

The f_dfdu function is called by the solver as follows:

> dfdu(1:MM_NU,3:MM_NX) = feval(f_dfdu, u(1:MM_NU, 3:MM_NX));

f_dfdu should return a vector dfdu of the same length as u (i.e. MM_NX − 2), of which each element is a Jacobian matrix (MM_NU × MM_NU).

Example:

```
function dfdu = dfduEuler(u)
n = size(u, 2); % number of points

for i = 1:n
  dfdu(:,:,i) = jac(u(:,i)); % jacobian at one specific point.
end


%%%%
function J=jac(u)
global gamma;

v = u(2) / u(1);
p = pEuler(u);
J = [ 0, ...
      1, ...
      0 ...
    ; 0.5 * (gamma-3) * v^2, ...
      (3-gamma) * v, ...
      gamma-1 ...
    ; 0.5 * (gamma-1) * v^3 - v * (u(3)+p) / u(1), ...
      (u(3)+p) / u(1) - (gamma-1) * v^2, ...
      gamma * v ...
    ];
```

Function for g

The f_g is called by the solver as follows:

> g = feval(f_g, u, xa);

f_g should return a vector g of the same length as u (i.e. MM_NX + 1), of which each element has dimension MM_NU (since f_g implements the vector function $\vec{g}$).

Example:
```
function g = gEuler(u, xa)
g = 0*u;
```

Function for boundary-conditions

The f_boundaries function is called by the solver as follows:

```
> u=feval(f_boundaries, t, x, u);
```

f_boundaries should update its input array u with the 'beyond-boundary-values'. Figure 3.2 shows that this involves the points $\{u_1, u_2, u_n, u_{n+1}\}$.

Example:
```
function u = neumann_sol(t, x, u)
global MM_NX MM_NU;

u(1:MM_NU,[1, 2, MM_NX, MM_NX+1]) = u(1:MM_NU,[4, 3, MM_NX-1, MM_NX-2]);
```

Array-lengths and non-locality

Of the above functions f_monitor and f_g are the only ones that receive the entire arrays u and xa. f_f and f_dfdu are always passed prefiltered arrays of length MM_NX − 2.

As a result, f_monitor and f_g explicitly use the globally defined MM_NX. This is because both functions might include spatial derivatives of u, and therefore also need points outside 3:MM_NX-1.

f_f and f_dfdu don't have to pay attention to the length of their input arrays. This is because of the implicitly assumed local nature of these functions ($f|_{x_j}$ only depends on u_j and not on any neighbour-points). Introducing non-local behaviour (e.g. human drivers looking ahead for traffic-density) is of course possible, but the user should pay attention to produce arrays of the correct length.

5.1.4 Command window output

During execution, each timestep is shown, optionally marked with '[S]' whenever a timesample is taken. Finally, some statistics on solving, remeshing and monitor-values are shown.
In case the user has requested for storing all samples in a file, he is asked for a short comment-message on this run.
In case the user has requested error diagnostics, they are displayed. More details on storing, reloading and error-checking solutions is found in section 5.1.6.

An example run of MMFVPDES could come up with the following output in the MATLAB command window:

Example:

```
...
t=0.195881 (\Delta t=0.00102687)         [S]
t=0.196903 (\Delta t=0.00102232)
t=0.197923 (\Delta t=0.00101965)
t=0.198944 (\Delta t=0.00102106)
t=0.199968 (\Delta t=0.00102446)
t=0.2 (\Delta t=3.16305e-005)         [S]
+--------------------------------------+
| Execution took 30.484 seconds        |
| Total number of time steps: 253      |
| Average number of remeshing steps: 3 |
+--------------------------------------+
| maxw   : 17.3519                     |
| avgw   : 3.9206                      |
| avgmaxw: 11.9664                     |
+--------------------------------------+
Data written to 'd:\traflowpack\work\settingsST_731377.9308.mat':
  Name          Size         Bytes  Class

  MM_ALLT       1x41           328  double array (global)
  MM_ALLU       3x41x101     99384  double array (global)
  MM_ALLX      41x102        33456  double array (global)

Grand total is 16646 elements using 133168 bytes

Please enter your own information for this run (within 'quotes').
It will be written to the file 'experiments.txt'.
Enter 'E' to open an editor for advanced editing: 'ST-example from swdocs'
Rel. error in u        : L_1 norm=0.0212904
Rel. error in u        : L_2 norm=0.00422599
Rel. error in u        : L_\infty norm=0.00286391
Rel. error in u, outside shock        : L_1 norm=0.0212904
Rel. error in u, outside shock        : L_2 norm=0.00422599
Rel. error in u, outside shock        : L_\infty norm=0.00286391
```

5.1.5 visualization output

Whenever action='plot' is issued, the solver comes up with various figure windows. The main routine that handles the visualization is

```
> show_results(plotIdx, plotExact);
```

where plotIdx indicates which variable to plot and plotExact indicates whether (*1*) or not (*0*) an exact solution should also be drawn. There are various visualization routines, that will separately be discussed in the following paragraphs.

2D plot of solution at specific time t_d

plotIdx only affects the 2D-plots; whenever plotIdx $\in$ 1..MM_NU, the indicated model-variable is drawn as a single plot. Whenever plotIdx=-1, all model variables are drawn as a set of subplots. Other values for plotIdx may be defined by the user.

By default, the plot-time t_d is set to Tend, but the 'mesh-movement'-plot will allow the user to interactively browse through all time-samples.

By redefining the global cell-array MM_plotlabels, the labels of the vertical axes of all subplots can be configured, for example:

Example:

```
> global MM_plotlabels; MM_plotlabels = '\rho', 'm', 'E';
```

The actual plotting is done in plot_data.m, but the user should not need to edit this file.

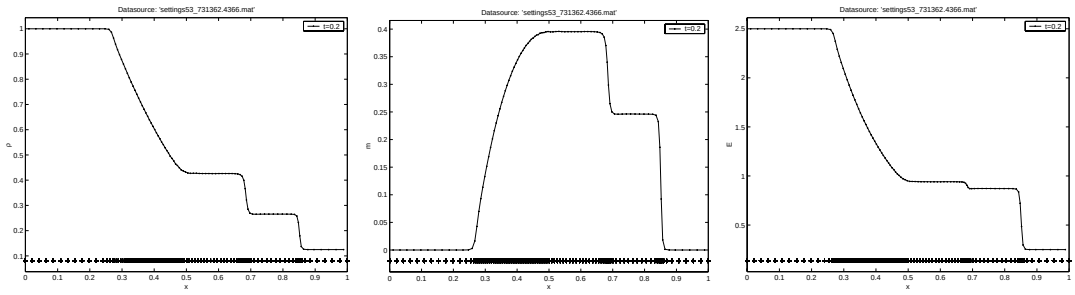

Figure 5.1: 2D visualization of all three model variables in the Euler-shocktube problem at sampled time t_d.

Whenever plotExact=1 is specified, the routine tries to retrieve an exact solution from the file mmfvexact.mat and plots the exact version of the requested model variables in the same subplots. More information on 'exact' solutions is given in section 5.1.6.

An example of this 2D visualization for the shocktube problem is shown in figure 5.1.

Mesh-map $x(\xi)$ at specific time t_d

Just the same as for the 2D solution plot, at any sampled time t_d a meshmap $x(\xi)$ can be shown. This was used a few times in chapter3, and another example is shown in figure 5.2.

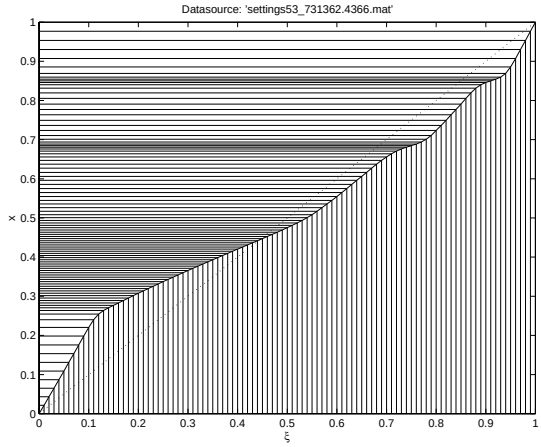

Figure 5.2: Mesh map $x(\xi)$ in the Euler-shocktube problem at sampled time t_d.

The actual plotting is again done in plot_data.m.

Mesh movement

The previous two plots were visualizations at specific sampled times t_d. A mesh-movement plot show values at all sampled times. It shows the movement of x-coordinates in an x-t diagram. Figure 5.3 shows an example of this.

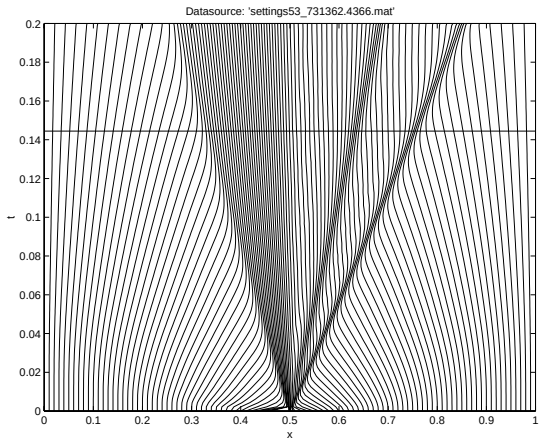

Figure 5.3: Mesh movement in the Euler-shocktube problem.

The location of each x-coordinate is drawn by a line through time. The mesh movement is also a quick and useful visualization for some basic insight on the behaviour of the solution itself.

Besides visualizing, the mesh-movement figure also allows the user to choose a sampled time t_d by placing a mouseclick anywhere in the axes. The closest t_d is selected and the 2D- and meshmap visualization are redrawn for the selected time. A horizontal line $t = t_d$ in the mesh-movement figure shows the currently selected time.

The combination of these three visualization allows for a flexible, interactive investigation of solution behaviour in time and space.

The actual plotting is again done in plot_data.m.

3D visualization

It is also possible to show one specific solution variable in the x-t-u-box. show_results contains some predefined calls to the actual visualization routines plot_surf. An example for the density is shown in figure 5.4.

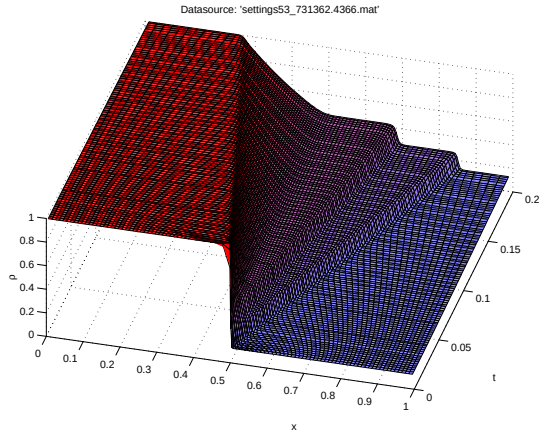

Figure 5.4: 3D visualization of ρ in the Euler-shocktube problem.

More visualizations

There are some more visualizations that are specifically aimed at traffic flow. These will be discussed in the next section on TRAFLOW .

The user can always define his own visualizations. The best approach for this is to create the new visualization routine in a separate file, and call it from show_results.

5.1.6 Input-/Output routines and error checking

Since simulation often takes quite some time, and recomputing solutions each time they are needed is not sensible anyhow, an option to store solutions in and retrieve them from files is very useful. The built-in MATLAB functions save and load are suitable for this, but the solver includes its own functions for data-storage.

Save, list and reload solutions

The current solution values in MM_ALLT, MM_ALLX and MM_ALLU can be stored in a file by issuing:

Example:
```
> save_data('c:\myfiles\', 'ST100');
```

This stores the mentioned variables in a file c:\myfiles\ST100.mat by calling save. Omitting the first directory argument (i.e. the filename is the first and only argument) stores the file in the TRAFLOW$_{PACK}$ workroot, which can be found by workroot.

When MMFVPDES is asked to store all sampled data in a file (plotExact=1), it chooses a unique filename by including a serial datetime-representation, for example: settingsST_731362.4396.

The user is also asked for a short comment-message on the stored solution. This will be appended to the file experiments.txt, which contains a list of all stored files, with a short description for later reference. This file can be easily displayed by issuing

Example:
```
> list_data
```

Solutions can be retrieved by issuing:

Example:
```
> load_data('c:\myfiles\', 'ST100');
```

This displays some short information on the retrieved variables and adds them to the global workspace. Again, omitting the directory defaults to the workroot. Note that the .mat suffix is always omitted on the command line.

Since the visualization routine show_results makes use of the model variables that are currently in the global workspace, stored solutions can be visualized easily and fast with only two commands:

Example:
```
> load_data('mydatafile'); show_results(-1,0);
```

Error diagnostics

It is also possible to compute some error-diagnostics on the current solution. A file mmfvexact.mat is required for this. This file should contain a (pseudo-) exact[2] solution at the same Tend.

[2] As discussed in chapter 3, a pseudo-exact solution is obtained by running a simulation with many (like 1000) meshpoints.

The file mmfvexact.mat is also a standard binary MATLAB file, containing the variables tn, xn and un. It can be created automatically by the solver whenever action=*'exact'*. The solver calls save_exact to actually save it in a file.

These files can be renamed for future reference, otherwise they will be overwritten each time action='exact' is issued.

Both error_check and show_results check for existence of the file mmfvexact.mat in the workroot, otherwise no error-handling is possible.

5.2 TraFlow : traffic flow simulation tool

TraFlow can be considered a 'shell' on top of MMFVPDES , specially aimed at traffic-simulations. It allows the user to specify intuitively a traffic flow model, and a specific problem (e.g. which userclasses are present, initial density and speed-distribution etc.). Subsequently, TraFlow transforms the specified settings into settings that are suitable for the solver. Finally the solver is called and its output further processed by TraFlow .

5.2.1 Commandline syntax

The way to invoke traflow from the MATLAB -commandline is as follows:

> traflow(action, f_msettings, f_psettings, f_monitor, n, Tstart, Tend, *varargin*)

The *slanted* parameters are optional. Explanation of the parameters:

action: Indicates what task the solvers should perform; one of the following:

- *'info'* Processes al user-specified input and settingsfiles, prints information about model- and problem-settings on the screen, end exits.
- *'sim'* Runs the simulation with the specified settings.

f_msettings: The name of the model-settingsfile without the .m suffix.

f_psettings: The name of the problem-settingsfile without the .m suffix.

f_monitor: The name of the monitor-function to be used without the .m suffix.

n: The number of meshpoints. TraFlow automatically passes MMFVPDES the right mesh-size so that the number of meshpoints *at the domain* is *exactly*[3] n.

Tstart: The start-time of the traffic flow simulation.

Tend: The end-time of the traffic flow simulation.

varargin: A variable-length list of optional arguments, for overriding some default solver-settings. These come in name, value-pairs. The currently supported parameters are listed below:

- *'mm_maxit'* The maximum amount of Gauss-Seidel iterations (3.14) for solving the moving-mesh equations. Default: *3*.
- *'mm_tol'* The tolerance-level for preliminary interruption of the Gauss-Seidel iterations. Default: *1E-6*.
- *'cflC'* The constant C in eq.(3.35) for fitting the maximal timestep to the CFL-condition. Default: *0.5*.
- *'solver'* The PDE should be used, see previous section on MMFVPDES for details. Default: *1*.

[3] Instead of $n - 2$.

- *'storeInterval'* The amount of time between two successive timesamples. u is not stored at each time-instant t since this would mostly produce unnecessary large datasets. Default: *5*.
- *'plotVar'* Exactly the same as plotIdx in MMFVPDES . Default: *-1* (show all variables, in subwindows).
- *'saveData'* Whether (*1*) or not (*0*) to store all sampled solution-values, mesh-values and times in a .mat file which can be reloaded for future analysis.
- *'errorCheck'* Whether (*1*) or not (*0*) to compute some error-diagnostics. A file mmfvexact.mat is required for this.

An example call could be:

Example:
```
> traflow('sim', 'kerner','kerner1', 'monitorBM', 100, 7*3600, 9*3600,'storeInterval', 60, 'cflC', 0.2);
```

In the next paragraphs, the above example is further exemplified by showing the source of the settingsfunctions and other required MATLAB -code.

5.2.2 Variables, userclasses and roadway lanes

In the previous section on MMFVPDES , the solver was shown to be suitable for a general system of MM_NU PDEs. Traffic flow equation typically involve two or three PDEs. However, each equation (or equivalently, each model-variable) is considered for each userclass and each roadway lane. This shows from the subscripts: $\rho_{(u,j)}$; j is not to be mistaken for the 'spatial index' as in x_j.

Again, the program makes some variables globally available:

TF_q: The number of model variables, usually two or three.

TF_m: The number of userclasses.

TF_l: The number of roadway lanes.

TF_n: The number of meshpoints within the domain; the solver works with TF_n+2 points.

To be able to pass this type of PDE systems to the solver, the solution-values for various userclasses and roadway lanes are not stored in new dimensions (i.e. no $\mathsf{TF\_q} \times \mathsf{TF\_m} \times \mathsf{TF\_l} \times K \times (\mathsf{TF\_n}+2)$-matrix). Instead, the two extra dimensions are incorporated into the first dimension of model-variables. To ensure correct addressing, the following predefined numbering-scheme is used:

$$\vec{w} = \left[w^{(i)}_{(u,j)}\right] \equiv [w_\imath], \quad i \in \{1,\ldots,q\}, \quad u \in \{1,\ldots,m\}, \quad j \in \{1,\ldots,l\} \tag{5.1}$$

$$\imath = (i-1)\cdot(m\cdot l) + (u-1)\cdot l + j, \quad \imath \in \{1,\ldots,q\cdot m\cdot l\} \tag{5.2}$$

For three model variables, two userclasses and two roadway lanes, this would result in a solution vector:

$$\vec{w} = \left[w^{(1)}_{(1,1)}, w^{(1)}_{(1,2)}, w^{(1)}_{(2,1)}, w^{(1)}_{(2,2)}, w^{(2)}_{(1,1)}, w^{(2)}_{(1,2)}, w^{(2)}_{(2,1)}, w^{(2)}_{(2,2)}, w^{(3)}_{(1,1)}, w^{(3)}_{(1,2)}, w^{(3)}_{(2,1)}, w^{(3)}_{(2,2)}\right]^T. \tag{5.3}$$

This approach results in a $(\mathsf{TF\_q}\cdot\mathsf{TF\_m}\cdot\mathsf{TF\_l}) \times K \times (\mathsf{TF\_n}+2)$-matrix, which is suitable for MMFVPDES .

Getting the right indexes and dimensions

Because of the tricky indexing in the solution-matrix, some user-friendly functions were developed:

getIdx(i): Returns all indices i for one specific model variable $w^{(i)}$. For example, in the above example (5.3), getIdx(2) would return [5, 6, 7, 8]. getIdx can also be called with 0, 2 or 3 variables. See the file documentation for details.

sameAllUCs(p): Duplicates the entire matrix p up to the number of userclasses TF_m and stacks the duplicate(s) on top of each other.

sameAllLanes(p): Duplicates each row of p up to the nr of lanes TF_l.

sameAllPoints(p): Duplicates each column of p up to the nr of points TF_n.

More details can be found in the in-file documentation.

5.2.3 Model-settingsfile syntax

The properties of a traffic flow model are specified in the f_msettings-parameter of TraFlow . TraFlow evaluates the settingsfile as follows:

> [desc, w, f_f, f_dfdu, f_g] = feval(f_msettings);

The three f_... parameters are the same as for MMFVPDES , desc contains a short description of this model and w contains information on the number, symbol and name of model variables. The example below illustrates the use of the model-settingsfile.

Example:

```
function [desc, w, f_f, f_dfdu, f_g] = kerner()

density  = struct('symbol','\rho', 'name','traffic density');
speed    = struct('symbol','V', 'name','average velocity');
w        = [density; speed];

f_f = 'kerner_f';
f_dfdu = 'kerner_dfdu';
f_g = 'kerner_g';

desc = sprintf('Kerner''s  2-eq model.\nRestricted to one roadway-lane, one user-class.');
```

Note that the number of userclasses m and roadway lanes l is not explicitly mentioned here. This has to be done by the problem-settings function. The PDE-functions have to be programmed for general values of m and l. The next examples shows this for f_f:

Example:

```
function f = kerner_f(w)
global c0;

ridx = getIdx(1);
vidx = getIdx(2);

f(ridx,:) = w(ridx,:).*w(vidx,:);
f(vidx,:) = 0.5*w(vidx,:).^2 + sameAllPoints(sameAllLanes(c0).^2).*log(w(ridx,:)+1e-14);
```

5.2.4 Problem-settingsfile syntax

The problem-settings involve information about domain-size and -bounds, userclasses, roadway lanes and initial distribution, the settingsfile is evaluated as follows:

> [a, b, bounds, ucs, lanes, f_init] = feval(f_psettings);

All parameters are fairly straightforward, except ucs. ucs contains not only information about the userclasses, but also all the required class-specific parameter values, like relaxation-times and desired velocities. The parameters should be put in a MATLAB -struct. TRAFLOW processes the settings and then makes each parameter globally available (using globaliseUCs) as an m-dimensional array with the name specified in the original struct. The example below illustrates the problem-settings:

Example:

```
function [a, b, bounds, ucs, lanes, f_init] = hd_test1()
a = 0;
b = 30E+3;
bounds = 'periodic';
ucs=struct(...
  'name',{'slow person-car','fast person-car','truck'},...
  'v0',{27.5, 32.5, 24},...
  'remax',{0.2, 0.2, 0.2},...
  'tau',{10, 10, 10},...
  'T',{1.4, 1.2, 2.6},...
  'l',{3.7, 3.7, 11.2},...
  'd',{1.0, 1.0, 1.0},...
  'alpha0',{0.007, 0.007, 0.007},...
  'dalpha0',{0.03, 0.03, 0.03},...
  'rcrit',{0.04, 0.04, 0.02},...
  'drcrit',{0.006, 0.006, 0.006}...
  );
lanes = ['lane1'];
f_init = 'hd_test1_init';
```

Since no experiments with multiple roadway lanes were performed, no further functionality is built in yet. However, the datastructures are suitable for containing multiple lanes (TF_l>1), so future improvements of the software are possible.

5.2.5 Input-/Output routines

The I/O functionality is completely handled by MMFVPDES . More details can be found in section 5.1.6. Error-diagnostics are rarely useful in traffic flow simulations, since hardly ever pseudo-exact solutions are computed.

5.2.6 Visualization routines

The visualization is still handled by show_results, as section 5.1.5 already discussed. For traffic flow, some additional visualization routines were developed.

Space-time trajectories

When the average velocity of traffic flow is known, imaginary cars can be placed at the domain, and time-integration of the velocity yields space-time trajectories for these imaginary cars. Figure 5.5 shows an example of this.

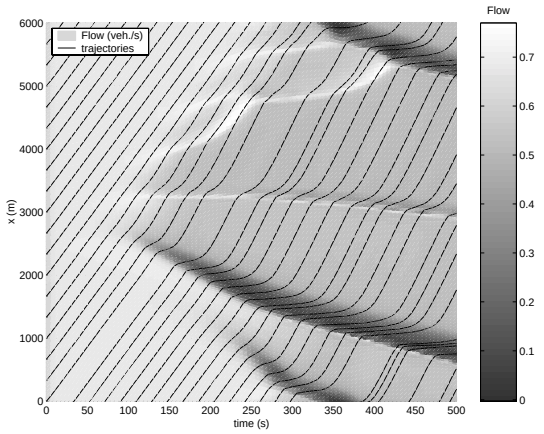

Figure 5.5: Space-time trajectories of 18 imaginary cars, and the average flow.

The trajectories-plot also shows the average flow as a gray-scaled background of the plot.

Real world density visualization

For an intuitive interpretation of the average density, a real-world visualization can be made. This is a 3D-view of the road at one specific time t_d, possibly with multiple roadway-lanes. The density is shown as a colored ribbon above it. Figure 5.6 shows an example of this roaddensityplot.

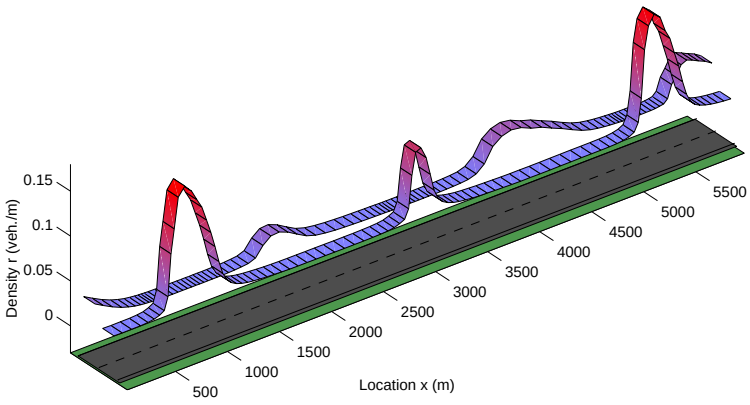

Figure 5.6: Real-world plot of the density at two roadway-lanes.

traffic flow movie

Based upon the real-world density plot, a movie-creating routine densityAnimation was developed. It builds a movie that runs from Tstart to Tend. Just above the road-surface from the real-world density plot, moving white dots are drawn, representing imaginary cars, as computed by trajectories.

Chapter 6

Conclusions and Recommendations

In this chapter the findings of the research are summarized. Based upon these, directions for possible further research can be given, as well as hints for program- and model-improvements.

6.1 Conclusion

Traffic Flow Models

Modelling traffic flow has proved to be a huge field of research. Restricting to macroscopic models still shows that many different formulations exist. These differ mainly in the approximation of interaction and anticipation behaviour.
The studied model by Kerner is a primitive 2-PDE formulation with an aproximation of equilibrium velocity by a relatively simple exponential function. More advanced approximations exist, for example a Boltzmann approximation of the stochastic interaction integral by Hoogendoorn and Helbing, and a non-local anticipation term by Helbing.
Another important improvement was to distinguish multiple userclasses by Hoogendoorn, e.g. person cars and trucks.
A new approach, that is currently being investigated, is making the various model-parameters not only userclass-specific, but also dependent on a third model variable, the workload. This 'workload' denotes the level of attention of a driver and influences for example reaction times.

As the macroscopic description of traffic flow improved through the years, it has become more and more important to formulate traffic behaviour based upon the individual drivers, and less important to use parallels with already known laws from gas and fluid dynamics. For example, models that used traffic viscosity have been criticised for using supposedly self-evident second-order derivatives, whereas in fact these terms were mainly useful for an easier numerical evaluation of the model.

More concerning the research in this report, the macroscopic formulations proved to be very interesting systems of PDEs, that served as suitable test-systems for the developed solver.

Moving Mesh Solver

Application of a moving mesh solver has proved to be advantageous in most situations, especially the more complicated problems like gas-dynamics and of course traffic flow. Not only a gain in accuracy is reached at relatively low costs, but in some cases also results were obtained that were impossible to obtain on uniform meshes at all.

A very important part of making an efficicient moving mesh solver is the choice of a suitable monitor function ω. Many variants of the basic square root formulation (3.4). All these monitors

share the same disadvantages that the user has to specify the multiplier α, and that α is not time-dependent.
The time-dependent normalized monitor ξ_{norm}, developed during this research has proved to be a good improvement, compared to the basic monitor functions. However, all these monitors also requires some smoothing to prevent the mesh from being moved too brusque.
Finally, the BM monitor proved the best in this context. It uses an alternative formulation, where not the derivatives are normalized, but the entire base level is dynamically adapted. No α has to be specified by the user and no smoothing is needed.

The finite volume approach yields quite accurate results, but the main disadvantage is the need for computing the Jacobian $\mathcal{J} \stackrel{\text{def}}{=} \partial_u f$. For more complicated systems, especially traffic flow models with multiple userclasses and multiple roadway lanes, it takes some effort to determine this Jacobian.
Besides, the solver is aimed at a quite general class of nonlinear systems of hyperbolic PDEs. Not much problem-specific tricks for faster runs and higher accuracy can be implemented. Still, this generality is of course also an advantage. Many different PDE systems can be solved by this solver. Some first experiments have already been performed with magneto-hydrodynamic PDE models, and many other systems may be possible.

Traffic Flow Simulation

Kerners model has been criticised for various reasons, but still it contains many interesting characteristics. *Local cluster formation*, *local breakdown* and *dipole layers* were phenomena that indeed could all be reproduced, with the developed simulator. the phenomena could also be explained by the means of fundamental diagrams. Main cause for all these features are the so-called *critical density ranges*. Within these density regions, only slight perturbations can cause huge effects, like phantom jams and stop-start waves.
Besides whether these phenomena are realistic or not, an important critique is that the presence or absence of them depends strongly on the proper combination of model-parameters.

Hoogendoorns model turned out to be unstable for the developed solver. Simulations with older models of Hoogendoorn were successful, but the conservative 2-PDE formulation was not. An additional analysis of what precisely causes the instability, and a more specific solver suitable for this term should solve the problem.

A short illustration of simulating traffic management measures, such as speed regulation was given. Although only qualitative results were obtained, indeed clear differences were observed between unmanaged and managed traffic flow.

TraFlow$_{\text{PACK}}$

The developed software turned out to be quite useful. TRAFLOW$_{\text{PACK}}$, consisting of the PDE solver MMFVPDES and the traffic flow simulator TRAFLOW , was not only used within this project, but also by both supervisors Chris Tampère and Paul Zegeling. The import and export functionality and the various traffic-flow-visualization routines made an efficient investigation of the traffic models possible.
In the near future some minor enhancements will be made, to make the package more user-friendly and easily distributable.

General

As a whole, this graduation project has turned out to be a very good 'Computational Science project'. Starting with an introduction into the traffic flow matter (application), a general class of PDEs was chosen to be the basis of further research (modelling). For this PDE class moving mesh techniques and a finite volume solver were investigated (numerical mathematics). The numerical techniques were implemented in useful MATLAB programs (software development), and finally, various interesting traffic flow simulations could be performed (computer simulation).

6.2 Improvements and further research

As mentioned before, the PDE solver is very general, and not suitable or efficient for just any problem. Possibly, additional PDE solving techniques could be implemented and used in combination with the moving mesh technique. For each problem to be solved, the most suitable PDE solver could then be used.

The existing MUSCL-type solver could also be improved. Currently the dynamic timestep adaptation is done quite straightforward, using the CFL condition. For systems of PDEs, things get more complicated and to resolve breakdowns, the 'pre-factor' $\mathcal{C}$ was introduced. A more thorough investigation of timestep adaptation would make the solver more robust.

The BM monitor that is finally used, is quite good, but possibly even better ones exist. This would make the moving-mesh approach even more efficient and possibly more accurate.

To gain more and better insight into the dynamics of traffic flow, more simulations should be performed. Also more complicated scenario's, like multiple userclasses and multiple lanes including lane-change behaviour should be considered. This also holds for the investigation of traffic management measures.
An important part of this is finding out why the solver was not able to handle Hoogendoorns concervative 2-PDE model correctly. If the term causing the instability can be identified, a much wider range of realistics traffic flow models can be handled.

References

[1] Anderson R.L., Herman R., Prigogine I., On the Statistical Distribution Function Theory of Traffic Flow, Operations Research, Vol. 10, No. 2, 1962 4

[2] Arem, van B., Katwijk, van R.T., and Zegwaard, G.F. Detailed specification of MIXIC 1.3, TNO Inro, 97/NV/021, Delft, The Netherlands, 1997 3

[3] Arthurs, A.M., Calculus of Variations, Routhledge & Kegan Paul, London, 1978 15

[4] Antoniotti, M., and Göllü, A., SHIFT and SmartAHS: A Language for Hybrid Systems Engineering, Modeling, and Simulation. *Proceedings of the USENIX Conference of Domain Specific Languages*, Santa Barbara, CA, U.S.A., October 1997 3

[5] Beckett, G., Mackenzie, J.A., Ramage, A., and Sloan, D.M., On the Numerical Solution of One-Dimensional PDEs Using Adaptive Methods Based on Equidistribution. *Journal of Computational Phsyics, Vol. 167, p.372-392*, 2001 33

[6] Capon, P.J., and Jimack, P.K., On the Adaptive and Finite Element Solution of Partial Differential Equations Using h-r-refinement. *School of computer studies, research report series, Report 96.13*, University of Leeds, April 1996 13

[7] Cortese, C. et. al., Macroscopic Modelling of Traffic Flow, Dept. of Mathematics and Statistics, University of Massachusetts, 1999

[8] Dahanzo, C.F., Requiem for second-order fluid approximations of traffic flow, *Transportation Research B, Vol. 29*, 1995, pp. 277-268 6

[9] Helbing, D., Traffic Modelling by Means of Physical Concepts, Pages 87-104 in: D. E. Wolf, M. Schreckenberg, and A. Bachem (eds.) Traffic and Granular Flow (World Scientific, Singapore), 1996 5

[10] Helbing, D., Gas-kinetic derivation of Navier-Stokes-like traffic equations. *Physical Review E, Vol. 53*, 1996, pp.2266-2381 8

[11] Helbing, D., Verkehrsdynamik, Springer-Verlag, Berlin, 1997 5, 6, 8, 11, 44, 83

[12] Helbing, D., and Treiber, M., Numerical Simulation of Macroscopic Traffic Equations, Computing in Science & Engineering, 1999 5, 6

[13] Hirsch, C., Numerical Computation of Internal and External Flows, Volume 1: Fundamentals of Numerical Discretization, New York: Wiley, 1988 22, 34, 55

[14] Hirsch, C., Numerical Computation of Internal and External Flows, Volume 2: Computational Methods for Inviscid and Viscous Flows, New York: Wiley, 1990 34, 35

[15] Hoogendoorn, S.P., and Bovy, P.H.L., Multiple user-class traffic flow modeling – derivation, analysis and numerical results., Research report VK2205.328. Delft University of Technology, 1998 v, vii, 5, 8

[16] Hoogendoorn, S.P., and Bovy, P.H.L., Continuum modelling of multi-lane heterogeneous traffic flow operations, Final report VK2205.330. Delft University of Technology, 1998 v, vii, 5, 8, 9

[17] Hoogendoorn, S.P., Multiclass continuum modelling of multilane traffic flow, Delft University Press, 1999 v, vii, 4, 5, 6, 8, 9, 52

[18] Hoogendoorn, S.P., and Bovy, P.H.L., Generic gas-kinetic traffic systems modeling with applications to vehicular flow, Elsevier Science Ltd, 1999 v, vii, 5

[19] Hoogendoorn, S.P., and Bovy, P.H.L., Gas-Kinetic Modelling and Simulation of Pedestrian Flows, Transportation Research Board, 2000 v, vii

[20] Hoogendoorn, S.P., and Bovy, P.H.L., Continuum modeling of multiclass traffic flow, Transportation Research B, 2000 v, vii, 5

[21] Hoogendoorn, S.P., Bovy, P.H.L., and Lint, van H., Short-term prediction of multiclass multilane traffic flow operations. *Proceedings of the 15th International Symposium on Traffic and Transportation Theory.*, Adelaide, Australia, 2002 v, vii, 5, 6, 9, 83

[22] Kerner, B.S., and Konhäuser, P., Cluster effect in initially homogeneous traffic flow, *Phys. Rev. E 48*, 1993, pp.2335-2338 v, vii, 6, 7, 49

[23] Kerner, B.S., Konhäuser, P., and Schilke, M., Formation of traffic jams caused by fluctuations and random processes in traffic flow, *7th WCTR Proceedings, Vol. 2*, Sydney, 1995 v, vii, 7, 44, 49

[24] Kerner, B.S., and Herrmann, M., Local cluster effect in different traffic flow models, *Physica A, Vol. 255*, 1999, pp. 163-188 v, vii, 7, 48, 49

[25] Hühne, R.D., Traffic Patterns in unstable Traffic Flow on Freeways. *Highway Capacity and Level of Service.* Brannolte (ed.), Rotterdam, 1991 6

[26] Kurganov, A., Noelle, S., and Petrova, G. Semi-Discrete Central-Upwind Schemes for Hyperbolic Conservation Laws and Hamilton-Jacobi Equations, University of Michigan, 2000 28

[27] Lighthill, M.H., and Whitham, G.B., On kinematic waves II: a theory of traffic flow on long, crowded roads, *Proceedings of the Royal Society of London series A*, 229, 317-345, 1955 2, 3

[28] Ludmann, J., Beeinflussung des Verkehrsablauf auf Straßen – Analyse mit dem fahrzeugorientierten Verkehrssimulationsprogramm PELOPS. *Schriftenreihe Automobieltechnik*, Institut für Kraftfahrwesen, Aachen, 1998 3

[29] Misener, J., SmartAHS and SHIFT Enhancements, Persistence and Query Interpretation, *California PATH Research Report UCB-ITS-PRR-2000-6*, CALIFORNIA PATH PROGRAM, March 2000 3

[30] Paveri-Fontana, S.L., On Boltzmann-like treatments for traffic flow: a critical review of the basic model and an alternative proposal for dilute traffic analysis, *Transportation Research B, Vol 9*, 1975 4

[31] Payne, H.J., Models of freeway traffic and control simulation, *Mathematical Models of Public Systems, Vol. 1*, 1971, pp.51-61 5

[32] Phillips, W.F., Kinetic Model for Traffic Flow, *Report DOT/RSPD/DPB/50-77/17*, National Technical Information Service, Springfield, 1977, VA 22161 7

[33] Shvetsov, V., and Hebling, D., Macroscopic dynamics of multi-lane traffic, *Physical Review E*, Vol. 59, 1999, pp. 6328-6339 9

[34] Smith, M., Duncan, G., and Druitt, S., PARAMICS: Microscopic Traffic Simulation For Congestion Management. Dynamic Control Of Strategic Inter-Urban Road Networks. Institution of Electrical Engineers, London, 1995 4

[35] Sod, G., A survey of several finite difference methods for systems of nonlinear hyperbolic conservation laws, *J. Comput. Phys. 27*, 1978 34

[36] Stockie, J.M., Mackenzie, J.A., and Russell, R.D., A moving mesh method for one-dimensional hyperbolic conservation laws. *SIAM J. Sci. Comput.*, 22 (2001), pp. 1791-1813. 36

[37] Tampère, C.M.J., Modelling of traffic flow in congestion for the analysis of Automated Driver Assistance Systems: Literature Review, *TNO Inro report*, Inro-V+V/2001-59, Delft, The Netherlands, 2001 4, 12

[38] Tang, H.-Z., and Tang T., A moving mesh algorithm for one-dimensional nonlinear hyperbolic conservation laws., Proc. of The 5th China-Japan Semniar on Numerical Mathematics., 2000

[39] Tang, H.-Z., and Tang T., Moving Mesh Methods for One- and Two-dimensional Hyperbolic Conservation Laws., Preprint, 2001 17, 18, 36

[40] Wallentowitz, H., Neunzig, D., and Ludmann, J., Auswirkungen neuer Fahrzeug- und Verkehrstechnologien - Analyse von Verkehrsfluß, Kraftstoffverbrauch und Emissionen mit PELOPS, *Tagungsband zum Ergebnis-Workshop Verkehr und Mobilität des Nordrhein-Westfälischen Forschungsverbundes Verkehrssimulation und Umweltwirkungen, Schriftenreihe Stadt Region Land 66, Institut für Stadtbauwesen*, RWTH Aachen, 1998 3

[41] Zegeling, P.A., and Keppens, R., Adaptive Method of Lines for Magneto-Hydrodynamic PDE Models, *Ch. 4 in 'Adaptive Method of Lines' by A. Vande Wouwer et al.*, Chapman & Hall/CRC, 2000 25

Glossary

Advanced Driver Assistance (ADA) An AVG-system where the driver remains responsible, but is supported by technologies like lane keeping, automated speed adaption etc. [p. 3]

aggregate-lane _- model, a model for simulating roads with one or more roadway lanes, where all lanes are aggregated into one 'aggregate lane'. Overtaking is possible, in contrast to single lane models. [p. 3]

anisotropy Traffic flow is considered to be anisotrope: drivers react mainly to what happens downstream, not upstream. In fluid and gas flow all directions have the same influence. [p. 6]

anticipation The anticiption of the drivers to changing traffic conditions downstream. [p. 6]

Automated Vehicle Guidance (AVG) Concept of full or partial automation of various kinds of vehicles. It can improve accessibility, increase safety and reduce pollution of traffic. *See also* Advanced Driver Assistance (ADA) *and* Fully Automated Vehicle (FAV). [p. 3]

conservative _- macroscopic model, macro-model that considers the conservative variables density r, flow m and energy e. *See also* primitive. [p. 5]

convection Term that describes the dynamical (in time) change of a certain variable, i.e density or velocity, due to spatially changing variables. For example, due to changing in- and outflow of traffic at a certain location, the density changes at that location. [p. 4]

density _- of a flow, The local density ρ within a fluid or a gas. [p. 35]

downstream The direction of the traffic flow. Traffic always moves downstream. [p. 6]

internal energy _- of a flow, all the energy of the flow that is not kinetic, mostly caused by increasing pressure or increasing temperature. [p. 35]

kinetic energy _- of a flow, the 'energy of movement' that a flow has, by having a velocity larger than zero. [p. 35]

total energy _- of a flow, The sum of kinetic and internal energy.

$$E_{\text{int}} = E_{\text{tot}} - E_{\text{kin}}.$$

See also internal energy *and* kinetic energy. [p. 35]

continuity equation PDE that describes the evolution of density. It denotes that the total amount (of traffic, fluid or gas, etc.) is conserved. Forms the traffic flow PDEs together with the equation of motion. [p. 5]

equation of motion PDE that describes the evolution of velocity due to convection, relaxation and interaction. Forms the traffic flow PDEs together with the continuity equation. [p. 5]

Fully Automated Vehicle (FAV) An AVG-system where the driver function is fully automated. Main functions are speed- and distance adaptation along the way, positioning across the road, interacting with other vehicles and detecting obstacles. [p. 3]

fundamental diagram a 2D diagram that specifies the relation between two traffic flow quantities. Best-known relations are $m(r)$, $V(r)$ and $m(V)$. [p. 10]

gas-kinetic _- continuum models, a class of models that is derived through techniques, borrowed from the description of gas-kinetics. *See also* mesoscopic. [p. 4]

interaction _- between two vehicles occurs whenever they drive at various velocities or whenever they are too close to each other. The impeded vehicle will try to overtake the impeding vehicle if possible, otherwise it will lower its velocity. [p. 4]

local clusters congestion waves whose upstream and downstream front are smaller than its body, i.e. wide congestion clusters, usually of several kilometers. [p. 48]

macroscopic _- model, simulation model for traffic flow with low detail description (high level of aggregation). Traffic is often represented using aggregrate characteristics like flow-rate, density and velocity. Traffic flow is described using partial differential equations with these characteristics variables. Macroscopic models may be classified according to the number of PDEs and their order. [p. 3]

mass flow The amount of matter that passes a certain location per time-unit: $m = \rho v$. [p. 35]

matlab A very well-known mathematical software package and programming environment by MathWorks Inc. (http://www.mathworks.org) [p. 2]

mesoscopic _- model, simulation model for traffic flow with medium detail description, where traffic is reresented by (small) groups of traffic entities. These models describe the behaviour of individuals, for example in probabilistic terms. *See also* gas-kinetic. [p. 3]

microscopic _- model, simulation model for traffic flow with high detail description, where individual entities are distinguished and traced. [p. 3]

MMFVPDES Moving Mesh Finite Volume Partial Differential Equation Solver. The developed moving mesh solver for general nonlinear systems of hyperbolic PDEs. Implemented in MATLAB . [p. 2]

Multi-Class _- model, simulation model for traffic flow in which each model variable is considered for a certain number of different user-classes. The non-convective terms may contain cross-terms in which different userclasses interact with each other. [p. 3]

Multi-Lane-Multi-Class _- model, an extended form of the MultiClass model, in which also various roadlanes are considered. The PDEs now also include terms describing lane-changing behaviour and possibly lanedrops. [p. 8]

phantom jam The appearance of a traffic jam, seemingly out of nothing. This happens in a certain critical density range, where small perturbations in density can eventually cause huge congestion waves. [p. 7]

platoon A cluster of dense traffic driving all more or less at the same speed. The traffic is said to be *platooning*. [p. 4]

primitive _- macroscopic model, macro-model that considers the primitive variables density r, velocity V. *See also* conservative. [p. 5]

Phase-Space-Density A probability-density of cars driving with velocity v and desired velocity v^0 at location x and time t: $\rho(x, t, v, v^0)$. The PSD is a generalization of the density ρ.[p. 5]

ramp metering controlling the number of cars that enters a road from an on-ramp. This is usually done by adapting the frequency of traffic lights at the on-ramp to the existing traffic flow at the road. [p. 43]

reduced phase-space density The probability density function of cars driving with velocity v at location x and time t, regardless of their desired velocity: $\varphi(x, v, t)$. The reduced PSD is a generalization of the density ρ. [p. 8]

relaxation _ of a vehicle, the process of adaptating the current speed to reach the equilibrium velocity. [p. 4]

stop-start waves congestion waves whose upstream and downstream front are bigger than its body, i.e. small congestion peaks. Between the peaks, traffic drives with normal speed, until the next wave is encountered: stop-start behaviour. [p. 7]

submicroscopic _ model, simulation model for traffic flow with very high detail description, where besides behaviour of individual entities, also the functioning of specific parts and processes of vehicles and driving tasks are considered. These include processes like gearing and braking, for which the various required operations and reaction times are described. [p. 3]

traffic flow (TF) _ model, simulation model describing the process of traffic flow. In this case only roadsegments are considered, with possible, no road networks. Classification of models is based on the level of detail. [p. 4]

TraFlow A commandline interface to the solver MMFVPDES , aimed at easy simulations of macroscopic traffic flow. Implemented in MATLAB . [p. 2]

trafic pressure _ is an additional improvement to the convection term in the equation of motion, usually set to $P \stackrel{\text{def}}{=} r\Theta$. [p. 5]

traffic viscosity Term to describe the tendency of drivers to accelerate along with drivers ahead of them, mathematically expressed by a second-order derivative $\partial_x^2 V$. [p. 6]

upstream The direction opposite to the direction of traffic flow. Congestion-structure tend to move upstream. [p. 6]

desired velocity userclass-specific velocity V^0 that drivers wish to accelerate to, in absence of interaction with other traffic. [p. 6]

equilibrium velocity Collective term for both acceleration towards desired velocity and deceleration due to interaction. In older models, this term was approximated by an analytical function fitted to empirical data. Helbing [11] and Hoogendoorn [21] approximate the interaction integral more sophisticated. [p. 6]

velocity _ of a flow, the average speed v at a certain location x and time t of a flowing fluid or gas. [p. 35]

velocity variance Statistical measure of the fluctuations of the velocity v around the average velocity V. $\Theta = \langle v^2 \rangle - \langle v \rangle^2$. [p. 5]

Variable Message Signs (VMS) signs above the road that displays speed limits 50, 70 or 90. VMS is used to improve the homogeneity of the traffic flow by adapting the flow to traffic conditions downstream. [p. 43]

Appendix A

Program Code

A.1 TraFlow$_{\text{PACK}}$: MMFVPDES and TraFlow

The developed solver MMFVPDES and simulator TRAFLOW have been cleaned up and made a bit more user-friendly. The main routines are accompanied with visualization tools and routines for importing and exporting simulation data. All in all, this results in several thousands of lines of MATLAB code.

It is for this reason, that none of the program codes is listed here. Instead, the solver and simulator have been placed in a software package TRAFLOW$_{\text{PACK}}$. More information on this package can be found at

http://www.inro.tno.nl/five/traflow/

This site also contains a link to a freely downloadable distribution of TRAFLOW$_{\text{PACK}}$.