COMPARISON OF GODUNOV'S AND RELAXATION SCHEMES APPROXIMATION OF SOLUTIONS TO THE TRAFFIC FLOW AND EULER EQUATIONS

SAMUEL KAUNDE MUTUA

MASTER OF SCIENCE

(Applied Mathematics)

JOMO KENYATTA UNIVERSITY OF AGRICULTURE AND TECHNOLOGY

2016

Comparison of Godunov’s and relaxation schemes approximation of solutions to the traffic flow and Euler equations

Samuel Kaunde Mutua

A thesis submitted in partial fulfilment for the degree of Master of Science in Applied Mathematics in the Jomo Kenyatta University of Agriculture and Technology

2016

DECLARATION

This research thesis is my original work and has not in part or whole been presented for any degree award in any university.

Signature .... Date ....

Samuel Kaunde Mutua

The research thesis has been submitted for examination with our approval as supervisors

Signature .... Date ....

Prof. Jackson Kwanza,

JKUAT, Kenya

Signature .... Date ....

Dr. Mark E.M. Kimathi,

TUK, Kenya

DEDICATION

I dedicate this work to my encouraging lady friend Rose, my dad John, mum Catherine and encouraging brothers Chris, Nick, Augustine and Steph. You are always my heroes.

ACKNOWLEDGEMENT

First and foremost am grateful to Abba Father for seeing me through the research and even taking me through my Master'sprogram.

Vividly grateful for the encouragement and unwavering support I received from my project supervisors, Prof Jackson Kwanzaand Dr. Kimathi you are such wonderful people in my life. Special thanks to Dr. Mark Eric Kimathi for the tireless support you offered in mentoring me in the field of research, I really enjoyed your tutorials, God bless you abundantly.

Overwhelming encouragement and financial support from my lady friend Rose, dad, mum and Brother Nicholis Mutua, I really appreciated. You are a family am always proud of.

All friends and colleagues, who in one way or the other assisted me in compiling my work, feel appreciated. Gods' blessings are upon you always.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF APPENDICES

NOMENCLATURE

SYMBOLS	MEANING
u_i	Velocity component, ms^{-1}
t	Time, s
$\mathbf{i}, \mathbf{j}, \mathbf{k}$	Unit vectors in the x, y and z directions respectively
$\frac{D}{Dt}$	Material derivative $\left(= \frac{\partial}{\partial t} + u\frac{\partial}{\partial x} + v\frac{\partial}{\partial y} + w\frac{\partial}{\partial z} \right)$
P	Pressure of the fluid, N/m^2
$\vec{\nabla}$	Gradient operator $\left(= \mathbf{i}\frac{\partial}{\partial x} + \mathbf{j}\frac{\partial}{\partial y} + \mathbf{k}\frac{\partial}{\partial z} \right)$
∇^2	Laplacian operator $\left(= \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2} \right)$
u	Velocity, ms^{-1}
µ	Viscosity, kg/(s.m)
υ	Kinematic viscosity, m^2/s
γ	Specific weight, N/m^3
e	Internal energy per unit mass of fluid
ρ	Fluid density kg/m^3
q	The velocity vector of the fluid with components u, v, w

ABBREVIATION

PDE	Partial Differential Equations
FDE	Finite Difference Equations
LWR	Lighthill-Whitham-Richards model
ODE	Ordinary Differential Equations
CFL	Courant-Friedrichs-Lewy

ABSTRACT

The study of Euler equations in gas dynamics is well known to give birth to the theory of hyperbolic conservation law, that is, Euler equations are essential in development, analysis and successful use of numerical methods for non-linear systems of conservation laws particularly in problems involving shock waves. This research deals with the study of Euler equations for isothermal gas as well as traffic flow model that is governed by two hyperbolic equations. The equations are analysed to obtain two real and distinct eigen values which enables one to determine the wave structure of the possible solution to the Riemann problem set up. The numerical solution to the Riemann problem set up is obtained using both the Godunov scheme and the relaxation scheme. Finally, comparison of the results obtained from these two schemes is done graphically and Relaxation scheme appears to be more promising and a good alternative scheme as compared to Godunov scheme because of its simplicity.

CHAPTER ONE

INTRODUCTION AND LITERATURE REVIEW

1.1Introduction

The study of Euler equations in gas dynamics is well known to give birth to the theory of hyperbolic conservation law, that is, Euler equations are essential in development, analysis and successful use of numerical methods for non-linear systems of conservation laws particularly in problems involving shock waves. In general, ahyperbolic system contains discontinuous processes involving contact, shocks and rarefactions. For example, if a gas in a tube initially prepared with a jump in the states (density and velocity) across some surface as the evolution proceeds in time these jumps will break up into a combination of shocks, rarefactions and contacts in addition to any bulk motion and sound waves that may exist or develop. In this chapter we seek to define some terminologies used in the thesis as well as literature review of the area of study.

1.2 Background Information

1.2.1 Isothermal gas

Isothermal gas is one in which the temperature throughout the gas remains constant and thus the ideal gas law is reduced to $p = a^2\rho$. That is pressure is a function of density alone.

1.2.2 Unsteady flow

An unsteady flow refers to the condition where the fluid properties at a point in the system change over time, otherwise the flow is steady.

1.2.3 Riemann problem

Riemann problem consist of equations together with the discontinuous initial data consisting of two constant states separated by a single discontinuity. It is very useful in understanding hyperbolic partial differential equations because all properties appear as characteristics in the solution. It is defined by: $u(x,t=0)=\begin{cases} u_L & x \le 0 \\ u_R & x > 0 \end{cases}$

1.2.4 Shock and Rarefaction waves

Consider the Euler equation for isothermal gas. The system consists of Euler equations and is strictly hyperbolic with two real and distinct eigenvalues, whereby, one is greater than the other.

Eigenvalues physically represent speeds of propagation of information. Depending on the initial data the eigenvalues may represent shock and rarefaction waves.

A shock wave commonly known as shock is a type of propagating disturbance that is characterized by an abrupt but nearly discontinuous change in characteristic of the medium. Across a shock there is always an extremely rapid rise in pressure, temperature and density of the flow.

1.2.5 Weak solution

Weak solution to an ordinary or partial differential equation is a function for which the derivatives may not all exist but which is nonetheless deemed to satisfy the equation in some precisely defined sense.

1.3 Literature Review

Jin and Xin (1995) presented a class of non-oscillatory numerical schemes for systems of conservation laws in several space dimensions. Using local relaxation approximation they constructed a linear hyperbolic system with a stiff lower order term that approximates the original system with a small dissipative correction. The main feature of this class of schemes was its simplicity and generality since it used neither Riemann solvers spatially nor non-linear systems of algebraic equations solvers temporally, yet it could achieve high order accuracy and picked up the right weak solutions.

LeVeque (1992) explains how Godunov proposed a way to make use of the characteristic information within the framework of a conservative method. Godunov suggested solving the Riemann problems forward in time and the solutions since they were easy to compute as well as gave substantial information about the characteristic structure and lead to conservation methods since they were themselves exact solutions of the conservation laws and hence conservative.

Borsche *et al* (2012) investigated numerically a special class of multi-phase traffic theories based on microscopic, kinematic and macroscopic traffic models. This was by deriving macroscopic traffic equations with multi-valued fundamental diagrams from different microscopic and kinetic models. Numerical experiments showed similarities and differences of the models. For all models, phase transitions appeared near bottlenecks depending on the local density and velocity of the flow.

Puppo and Semplice (2011), reviewed recent results on the role of entropy in the numerical integration of conservation laws. The entropy indicator preserved the correct sign of the entropy dissipation provided the numerical entropy flux is not only consistent but accurately tailored to the particular numerical flux used to update the solution.

Wen-Long*et al* (2011), applied eight difference schemes and five limiters to numerical computation of Riemann problem and compared the resolutions of discontinuities of each scheme produced. The results showed that the numerical dissipation of each scheme were vital to improve the schemes accuracy and stability.

Ksibi and Moussa(2008) studied the numerical simulation of a one dimensional shock tube problem at super critical fluid conditions, where it was found that the numerical calculations gave similar results for a perfect gas and at supercritical condition similar profiles of density pressure and velocity are given.

Mengel (2006) studied the numerical computation of the shock tube problem by means of wave digital principles and showed that the MD Kirchoffs network which represents the Euler equation can successfully be extended by taking viscosity into account to represent the Navier Stokes equation.

Zhang (2002) presented a non-equilibrium traffic model shown to be devoid of the gas like behavior that plagues other higher order models. The system of partial differential equations that described the model was hyperbolic and had two characteristic fields: one which was genuinely non-linear and the other were linearly degenerate. The non-equilibrium model did not only retain all capabilities of the Lighthill-Whitham-Richards(LWR) model, but also completely eradicated the gas-like behavior that plagues most other non-equilibrium models. Moreover, the model captured certain traffic phenomena that eluded the Lighthill-Whitham-Richards (LWR) model, such as, vehicle clustering and forward propagation of disturbances in dense traffic.

Zhang (2000) analyzed the structural properties of the shock and rarefaction wave solutions of a non-equilibrium theory of vehicular traffic flow. It was noted that the speed equations also comes with more complex traffic dynamics, that is, instead of one characteristic field in the equilibrium theory, non-equilibrium theories have two characteristic fields in which two kinds of shock and rarefaction waves develop.

Aw and Rascle (2000) introduced a new "second order" model of traffic flow by replacing the space derivative as used in previous "second order" models with a convective derivative. This very simple repair completely resolved the inconsistencies of the previous models as well as nicely predicted instabilities near the vacuum, that is, for very light traffic.

Zhang (1999) presented a mathematical theory for modeling the hysteresis phenomenon observed in traffic flow and proposed that acceleration, deceleration and equilibrium flow should be distinguished in obtaining speed-concentration and/or occupancy relationship such that the phase transitions from one phase to another could be correctly identified. He provided guidelines for proper data treatment in traffic flow analysis and clarified certain misconceptions in the study of traffic flow relationships.

Hoogendoorn (1999) established a new traffic flow theory, developed several models on various levels of aggregation, described suitable numerical approximation schemes analyzed MLMC traffic flow data and presented some interesting macroscopic simulation results. The theory provided a genuine extension and generalization of traditional macroscopic traffic flow theory while the model provided insight into both the interactions between the distinct user-classes and the interactions between the lanes of the motorway.

Zhang (1998) presented a new continuum traffic flow theory based on both empirical evidence of traffic flow behavior and basic assumptions on drivers reaction to stimuli. The new theory included the Lighthill-Whitham-Richards (LWR) theory as a special case and removed certain deficiencies of that theory without introducing the undesirable property of wrong way travel.

Samtaney (1997) studied the computational methods for self-similar solutions of the compressible Euler equations and it was evident that in the self-similar solution the resolution of the discontinuities is sharper than the corresponding initial value solution. This was evident from an examination of the self-similar solution of the Sod's shock tube problem and the one dimensional shock refraction problem, both of which exhibit a very sharp contact discontinuity.

In this study we consider the Euler equations for isothermal case as well as for traffic flow models. Analyzing the equations to determine the wave structure, which are then utilized in the numerical approximation of the solution to the Riemann problem set up. We implement both the Godunov and Relaxation schemes and compare their performance to determine which of the two schemes would be appropriate in obtaining reasonable shock resolution.

1.4 Problem Statement

In this study we compare the performance of both Godunov and relaxation schemes in resolving a combination of possible wave equations obtained from both Euler equations and traffic flow models.Relaxation scheme is known to avoid costly numerical treatment of the nonlinearity of hyperbolic equations and as well doesn’t require a particular space discretization as compared to Godunov scheme. By considering the Euler equations for isothermal gas we set up a Riemann problem which is then solved using the two numerical schemes. Moreover, using a suitable anticipation term derived from traffic flow theory we write a traffic flow model that is governed by two hyperbolic equations and that are equivalent to the Euler equations and then solve them numerically using the two schemes.

1.5 Justification of the Study

The Euler equations are essential in development, analysis and successful use of numerical methods for non-linear systems of conservation laws particularly in problems involving shock waves. Therefore reasonable understanding of the mathematical structure of these equations and their solutions is crucial. All the computing methods for the Euler equations in two and three space dimensions rely heavily on techniques developed for the one dimensional Euler equations. The traffic flow model equations have been considered so as to bring the application part of the schemes in simulating vehicular traffic.

1.6 Objectives

The main objective of the study is to compare the numerical approximation of both Godunov and Relaxation schemes to the solutions of both Euler and the Aw-Rascle traffic flow model equations.

The specific objectives of the study are:-

i. To compare the numerical solutions of Euler equations for isothermal gas using both Godunov and Relaxation schemes.

ii. To compare the numerical approximation of both Godunov and Relaxation schemes using traffic flow model.

iii. To compute the Riemann invariants which are utilized in computing the Riemann problem using the Godunov scheme

CHAPTER TWO

GOVERNING EQUATIONS

2.1 Introduction

In this chapter we outline the governing equations which are the Euler equations and the traffic dynamic equations. Non dimensionalization of the equations is done as well as analysis of both Euler and Traffic flow equations.

2.2 Euler and Traffic Flow Equation

2.2.1 Euler equations

Equation of continuity

This is expressed as

$(\rho)_t + (\rho u)_x = 0$ ....(2.1)

That is the equation of conservation of mass of a fluid flow in one dimensional flow. It states that as a fluid moves from one point to another, the amount of mass remains constant that is mass is neither created nor destroyed.

Equation of motion

Also known as the Euler's equation of motion and derived from Newton's second law of motion that can be written in the conserved form as:

$(\rho u)_t + (\rho u^2 + p)_x = 0$ Where p is a function of ρ given by $p = a^2\rho$ ....(2.2)

2.2.2 Traffic dynamic equations

LWR model

This was a first order non-linear partial differential equation of hyperbolic type (Zhang, 1998).

$$\frac{\partial\rho(x,t)}{\partial t} + q'_e(\rho)\frac{\partial\rho(x,t)}{\partial x} = 0 \qquad \text{where} \quad q'_e(\rho) = \frac{dq_e}{d\rho} \quad \text{and} \quad q_e = \rho v_e \text{ ....(2.3)}$$

Payne model

Since the LWR model had a number of deficiencies such as it lacked a mechanism for traffic to accelerate or decelerate at a finite speed when the concentration gradient is large and also its assumption that the equilibrium speed concentration relationship holds for non-equilibrium speed and concentration observation (Zhang, 1998), the Payne model was developed.

Payne model consist of two equations and are usually based on the analogy to one dimensional fluid flow.

$$\partial_t\rho + \partial_x(\rho v) = 0 \text{ ....(2.4)}$$

$$\partial_t v + v\partial_x v + \rho^{-1}c_0(\rho)\partial_x\rho = (\tau^{-1})(V(\rho) - v) \text{ ....(2.5)}$$

With p, traffic pressure, given by $p = p(\rho) = a_0\rho$ inspired from gas dynamics such as isothermal law (Hoogendoorn, 1999) while $p^{'}$ describes the decrease-rate in the equilibrium velocity with increasing density.

AW-Rascle model

The Payne model was reportedly said to have difficulties which may have been caused by numerical methods used to convert a nonlinear system of hyperbolic Partial Differential Equations (PDEs) to set of Finite Difference Equations (FDEs). These problems were attributed to a fundamental flaw in the model since it produces 'wrong way travel' that is negative travel speed under certain circumstances (AW, 2000).

$$\partial_t \rho + \partial_x (\rho v) = 0 \quad \text{(2.6)}$$

$$\partial_t (v + p(\rho)) + v\partial_x (v + p(\rho)) = 0 \quad \text{(2.7)}$$

2.3 Non-Dimensionalisation of Euler Equations

Using equation [2.1] and equation [2.2], that is:

$$\frac{\partial \rho}{\partial t} + \frac{\partial (\rho u)}{\partial x} = 0$$

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + \frac{1}{\rho}\frac{\partial p}{\partial x} = 0$$

To non-dimensionalize the above equations we use the following transformations. Let $\hat{u}, \hat{t}, \hat{\rho}, \hat{p}$ and $\hat{x}$ be the non-dimensional velocity, time, density, pressure and length respectively.

Then if u, t, $\boldsymbol{\rho}$, p and x are the characteristic values of velocity, time, density, pressure and length respectively; then we have:

$$\rho = \rho L\hat{\rho}\ ,\ p = pL\hat{p}\ ,\ \ t = \frac{L}{\sqrt{\frac{pL}{\rho L}}}\hat{t}\ ,$$
$$x = L\hat{x}\ ,\ \ u = \sqrt{\frac{pL}{\rho L}}\hat{u} \quad \text{.........}(2.8)$$

From equations 2.5 and 2.7, we have

$$\frac{\partial\rho}{\partial t} = \frac{\partial\rho}{\partial\hat{\rho}}\cdot\frac{\partial\hat{\rho}}{\partial\hat{t}}\cdot\frac{\partial\hat{t}}{\partial t} \Rightarrow \frac{\partial\rho}{\partial t} = \rho\sqrt{\frac{pL}{\rho L}}\frac{\partial\hat{\rho}}{\partial\hat{t}} = \rho L\frac{\sqrt{\frac{pL}{\rho L}}}{L}\frac{\partial\hat{\rho}}{\partial\hat{t}}$$

$$\frac{\partial(\rho u)}{\partial x} = \rho L\frac{\sqrt{\frac{pL}{\rho L}}}{L}\frac{\partial(\hat{\rho}\hat{u})}{\partial\hat{x}}$$

Therefore, the first equation becomes,

$$\rho L\frac{\sqrt{\frac{pL}{\rho L}}}{L}\frac{\partial\hat{\rho}}{\partial\hat{t}} = -\rho L\frac{\sqrt{\frac{pL}{\rho L}}}{L}\frac{\partial(\hat{\rho}\hat{u})}{\partial\hat{x}}$$

So, $$\frac{\partial\hat{\rho}}{\partial\hat{t}} = -\frac{\partial(\hat{\rho}\hat{u})}{\partial\hat{x}} \quad \text{.........}(2.9)$$

Similarly, the second equation becomes,

$$\frac{\partial(\rho u)}{\partial t} = -\frac{\partial(\rho u^2 + p)}{\partial x}$$

$$\frac{\sqrt{\frac{pL}{\rho L}}}{L}\rho L\sqrt{\frac{pL}{\rho L}}\frac{\partial(\hat{\rho}\hat{u})}{\partial\hat{t}} = -\frac{1}{L}\frac{\partial}{\partial x}\left(\rho L\hat{\rho}\frac{pL}{\rho L}\hat{u}^2 + pL\hat{p}\right)$$

$$\Rightarrow \frac{pL}{L}\frac{\partial(\hat{\rho}\hat{u})}{\partial\hat{t}} = -\frac{1}{L}\frac{\partial}{\partial x}(pL\hat{\rho}\hat{u}^2 + pL\hat{p})$$

Thus, $$\frac{\partial(\hat{\rho}\hat{u})}{\partial\hat{t}} = -\frac{\partial}{\partial x}(\hat{\rho}\hat{u}^2 + \hat{p}) \quad \text{.........}(2.10)$$

From the above, we have been able to show that using the above method of non-dimensionalising the equations remains unaltered, therefore in the calculation we shall use the non-dimensional values not the physical values.

2.4 Assumptions of the Study

In order to describe the phenomenon mathematically the following assumptions were made for fluid dynamics:-

i. The fluid flow is unsteady, so as to enable us study the propagation of wave that is, shock and rarefaction waves.
ii. The density of the fluid changes across the shock and thus the fluid is compressible.
iii. The flow considered is one-dimensional for simplicity.

The equivalent assumptions for traffic flow are:-

i. Drivers adapt to traffic changes in their reaction zones not instantaneously but after some time that is known as reaction time.
ii. The model remains in a non-equilibrium state over time that is $(\rho, v) \neq (\rho, v_e(\rho))$, this is to allow for sustained presence of certain wave phenomena that do not exist in the equilibrium model.
iii. The traffic flow is one dimensional.

The fundamental equations of fluid dynamics are based on the following universal laws of conservation i.e. conservation of mass, momentum and energy and are presented in three quantities: density $\boldsymbol{\rho}$, velocity u and a specific energy e.

CHAPTER THREE

RESULTS AND DISCUSSION

3.1 Introduction

In this chapter analysis of both Euler equations and traffic flow equations is done. The two numerical schemes used to solve the equations that is Godunov scheme and Relaxation scheme are analysed and later simulation and comparison of the two schemes done and visualized using graphs.

3.2 Analysis of the Equations

3.2.1 Euler Equations

Considering the one-dimensional time dependent Euler equation for the conservation laws given by $U_t + F(U)_x = 0$ ....(3.1)

Where $U = \begin{bmatrix} \rho \\ \rho u \end{bmatrix}$ $F = \begin{bmatrix} \rho u \\ \rho u^2 + a^2 \rho \end{bmatrix}$

With initial conditions $U(x,0) = \begin{cases} U_L & x \le 0 \\ U_R & x > 0 \end{cases}$ ....(3.2)

In solving the Riemann problem, we shall frequently make use of the vector $W = (\rho, u)$ of primitive variables rather than the vector U of conserved variables, where is ρ density and u is the particle velocity, (Toro, 2013)

Data consists of just two constant states which in terms of primitive variables are $W_L = (\rho_L, u_L)$ to the left of $x = 0$ and $W_R = (\rho_R, u_R)$ to the right of $x = 0$ and separated by a discontinuity at $x = 0$. For the case in which no vacuum is present the exact solution of the Riemann problem has two waves which are associated with the eigenvalues $\lambda_1 = u - a$ and $\lambda_2 = u + a$, (Mutua *et al*, 2013)

From the above eigenvalues and their corresponding eigenvectors then we can be able to determine the structure of the waves as follows.

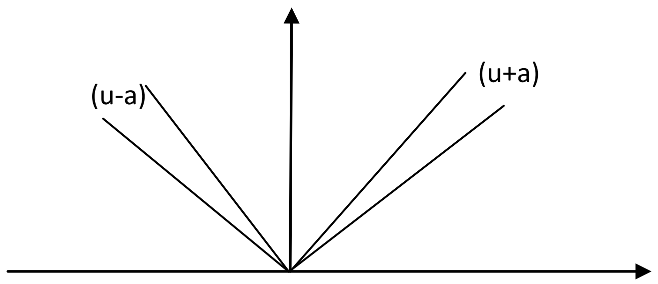


Fig 3.1.1: Structure of the solution on the x-t plane for the time-dependent Euler equation.

These two waves separate three constant states namely W_L (data on the left hand side), W_* and W_R (data on the right hand side). The solution to this problem depends on the relative values of W_L and W_R. That is, for $W_L < W_R$, a rarefaction wave is going to develop while for $W_L > W_R$, a shock wave is going to appear. That is,

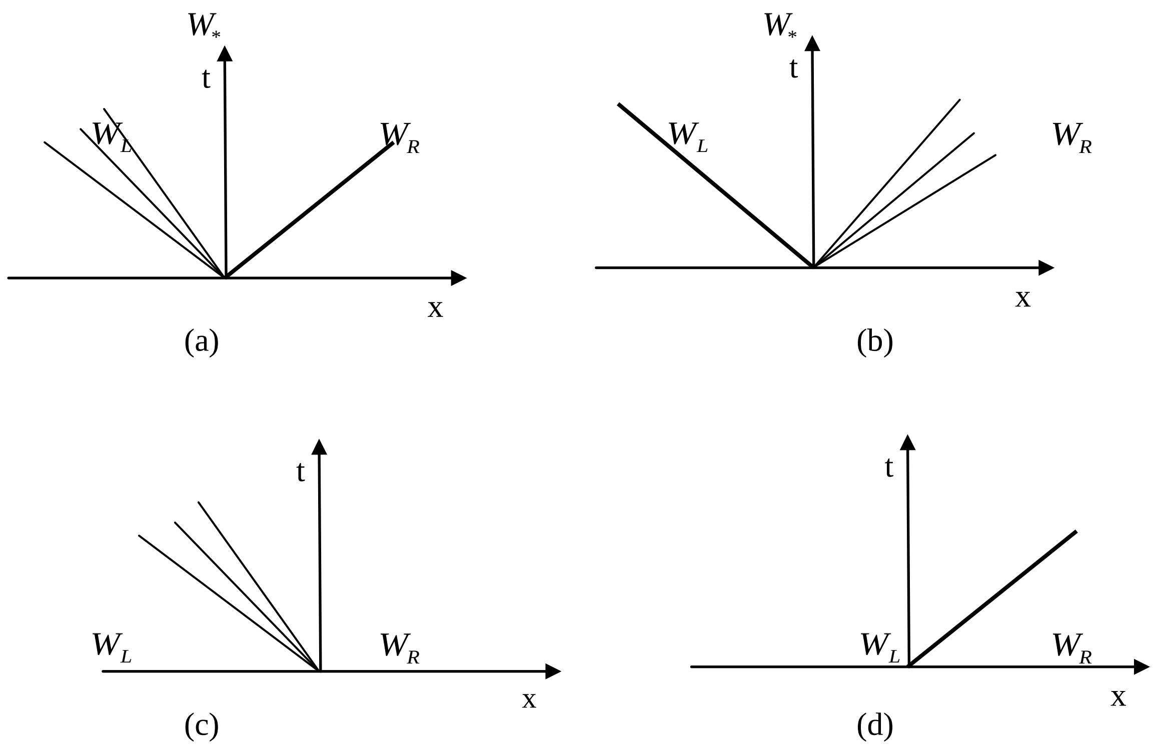


Figure3.1.2: Possible solutions to the Riemann's problem

All the two quantities, that is, density and particle velocity change across a shock wave. Considering a right facing shock wave travelling at the constant speed, S. In terms of the primitive variables we denote the state ahead of the shock by $W_R = (\rho_R, u_R)^T$ and the state behind the shock by $W_* = (\rho_*, u_*)^T$. Now we are interested in deriving relations across the shock wave between the various quantities involved. We accomplish this by finding the intermediate state such that they are connected by a discontinuity satisfying the Rankine-Hugoniot condition, given by:-

$$\rho_l u_l - \rho_* u_* = S(\rho_l - \rho_*)$$
$$(\rho_l u_l^2 - a^2 \rho_l) - (\rho_* u_*^2 - a^2 \rho_*) = S(\rho_l u_l - \rho_* u_*)$$

Now introducing a mass flux, Q, and using the above equations to do some manipulation one can be able to obtain the shock speed S, (Leveque, 1992), given as:

$$S = u_* \pm a\sqrt{\frac{\rho_l}{\rho_*}} \qquad \text{.......................}(3.3)$$

Similarly, inspection of the eigenvectors for the primitive variable formulation reveals that density and particle velocity change across a rarefaction wave. Now utilizing the i-th generalized Riemann invariants given by the (M-1) ODEs, we have

$$U = (\rho, \rho u)^T, \quad \lambda_1 = u - a, \quad \lambda_2 = u + a$$

$$K^{(1)} = \begin{bmatrix} 1 \\ u-a \end{bmatrix}, \quad K^{(2)} = \begin{bmatrix} 1 \\ u+a \end{bmatrix}$$

$$\therefore \quad \frac{du_1}{K_1^{(i)}} = \frac{du_2}{K_2^{(i)}}$$

$$\Rightarrow \frac{d\rho}{1} = \frac{d(\rho u)}{u-a} \quad \Rightarrow du + \frac{a}{\rho}d\rho = 0$$

$$\Rightarrow u + \int \frac{a}{\rho}d\rho = 0 \quad \Rightarrow u + a\ln\rho = C \quad 1st\ Riemann\ Invariant$$

$$\Rightarrow u - a\ln\rho = C \quad 2nd\ Riemann\ Invariant \qquad \text{.......................}(3.4)$$

Where C is a constant.

Now using the above equations and some manipulations we obtain ρ_* and u_* as shown below

$$u_* + a\ln\rho_* = u_L + a\ln\rho_L$$
$$u_* - a\ln\rho_* = u_R - a\ln\rho_R$$
$$\therefore 2u_* = u_L + u_R + a\ln\rho_L - a\ln\rho_R$$
$$2u_* = u_L + u_R + a(\ln\rho_L - \ln\rho_R)$$
$$\Rightarrow u_* = \frac{u_L + u_R + a(\ln\rho_L - \ln\rho_R)}{2} \qquad \text{.......................}(3.5)$$

Similarly,

$$2a\ln\rho_* = u_L - u_R + a\ln\rho_L + a\ln\rho_R$$

$$\ln\rho_* = \frac{u_L - u_R + a(\ln\rho_L + \ln\rho_R)}{2a}$$

$$\therefore \rho_* = e^{\frac{u_L - u_R + a(\ln\rho_L + \ln\rho_R)}{2a}} \quad \text{...}(3.6)$$

Suppose we wish to solve the Riemann problem with left and right states u_L and u_R. Just as in the linear case, we can accomplish this by finding an intermediate state u_* such that u_L and u_* are connected by a discontinuity satisfying the Rankine-Hugoniot condition and so are u_* and u_R (Leveque, 1992).

3.2.2 Traffic Flow Model Equation

Using the conservative form of the AW-Rascle model we set up the Riemann problem (3.2) with piecewise constant initial data

Where $U = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} \rho \\ y \end{bmatrix} \quad F(U) = \begin{bmatrix} F_1 \\ F_2 \end{bmatrix} = \begin{bmatrix} \rho u \\ yu \end{bmatrix}$ for $y = \rho u + \rho p(\rho)$

The general solution of the Riemann problem includes a 1-wave connecting to the left state U_L to an intermediate state U_* (to be defined) and a 2-wave connecting this intermediate state to the right state U_R. Since the 1-waves can either be shocks or rarefaction waves there will be the following types of solutions:

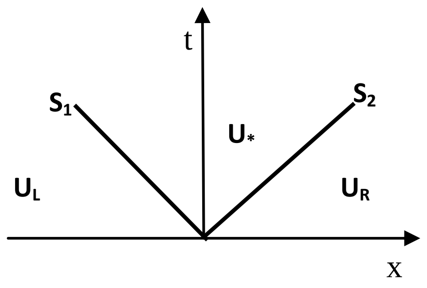


Fig 3.1.3: Possible solution

- 1-shock of speed S$_1$, connecting U_L to an intermediate state U_* followed by a 2-contact discontinuity connecting U_* to the right state U_R .

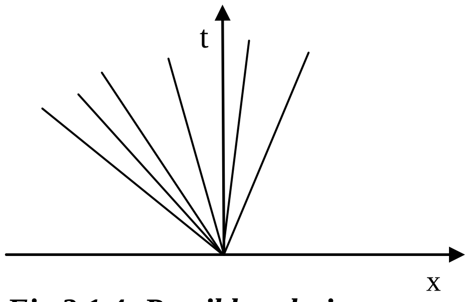


Fig 3.1.4: Possible solution

- 1-rarefaction wave, connecting U_L to an intermediate state U_* followed by a 2-contact discontinuity connecting U_* to the right state U_R .

The shock speed S$_1$ can either be less than zero or greater than zero. To determine the intermediate state we need to compute the Riemann invariants in the sense of Lax and use the *i*-Lax curves to represent the solution of our preferred eigen structure, for *i=1,2*(Kimathi, 2012). Thus to determine the *i*-Lax curves, we arbitrary choose the situation where a given left state U_L can be connected to an arbitrary state U_* on the right by a 1-shock of speed S$_1$. Thus for any discontinuity of speed S$_1$ to satisfy the Rankine-Hugoniot condition we write:

$$\begin{aligned} \rho_* u_* - \rho_L u_L &= S_1(\rho_* - \rho_L) \\ y_* u_* - y_L u_L &= S_1(y_* - y_L) \end{aligned} \quad \text{......}(3.7)$$

Where on eliminating S1 and simplifying we have:

$$\frac{y_*}{\rho_*} = \frac{y_L}{\rho_L} \quad \text{......}(3.8)$$

And since state U_* is arbitrary the *i*-Lax curve passing through U_L are obtained from equation (3.8) in terms of the primitive variables as (Kimathi, 2012).

$$L_1(\rho;\rho_L,u_L)=u_L+p(\rho_L)-p(\rho)$$

Similarly, the 2-Lax curves are obtained by considering the Riemann invariants

$$L_2(\rho;\rho_R,u_R)=u_R$$

Now stating the Riemann invariants W_1 and W_2 associated with the respective characteristics λ_1 and λ_2 we have:

$$W_1=u+p(\rho) \qquad W_2=u$$

3.3 Numerical Schemes

3.3.1 Relaxation scheme

Let's consider systems of conservation laws in one dimension (3.2).

Now introducing the relaxation system corresponding to the above equation, we have

$$\begin{aligned} &U_t+V_x=0 \\ &V_t+AU_x=-\frac{1}{\xi}(V-F(U)) \end{aligned} \quad \text{......................}(3.9)$$

Where $\xi>0$ and;

$$V=\begin{bmatrix} v_1 \\ v_2 \end{bmatrix},\ A=\begin{bmatrix} a_1 & 0 \\ 0 & a_2 \end{bmatrix}$$

The matrix A is a positive diagonal matrix to be chosen.

For $\boldsymbol{\xi}$ sufficiently small, it is expected that by solving (3.9) properly, one can obtain good approximations to the original conservation laws

The positive constant a need satisfy: (Jin, 1995)

$$-\sqrt{a} \le F'(U) \le \sqrt{a} \qquad \forall U$$

For the relaxation system (3.9), the initial data is:

$$U(x,0) = U_0(x)$$
$$V(x,0) = V_0(x) \equiv F(U_0(x))$$

Now introducing the spatial grid x_j with mesh width $h_j = x_{j+\frac{1}{2}} - x_{j-\frac{1}{2}}$ while the discrete

time level t_n with time step $k = t_{n+1} - t_n$ for n=0, 1, 2.

A spatial discretization to equation (3.9) in conservation form can be written as:

$$\frac{\partial}{\partial t}U_j + \frac{1}{h_j}(V_{j+\frac{1}{2}} - V_{j-\frac{1}{2}}) = 0$$
$$\frac{\partial}{\partial t}V_j + \frac{1}{h_j}A(U_{j+\frac{1}{2}} - U_{j-\frac{1}{2}}) = -\frac{1}{\xi}(V_j - F_j) \tag{3.10}$$

Where the averaged quantity F$_j$ is defined by

$$F_j = \frac{1}{h_j}\int_{x_{j-\frac{1}{2}}}^{x_{j+\frac{1}{2}}} F(U)dx \approx F(U_j)$$

The relaxation system 3.9 has two characteristic variables, see (Jin, 1995)

$$v_p \pm \sqrt{A}u_p, \quad p = 1, 2 \tag{3.11}$$

that travel with the frozen characteristic speeds $\pm\sqrt{A}$ respectively.

For better accuracy we use a second-order scheme that is the Van Leer's MUSCL scheme, see (Van, 1979).

Applying this scheme to the pth component that is equation 3.11, gives

$$(v_p + \sqrt{a_p} u_p)_{j+\frac{1}{2}} = (v_p + \sqrt{a_p} u_p)_j + \frac{1}{2} h_j \sigma_j^+$$
$$(v_p - \sqrt{a_p} u_p)_{j+\frac{1}{2}} = (v_p - \sqrt{a_p} u_p)_{j+1} - \frac{1}{2} h_j \sigma_{j+1}^- \quad \text{....................(3.12)}$$

Where σ_j is the slope of $v_p \pm \sqrt{a_p} u_p$ on the j-th cell which we define using Sweby's notation, see (Sweby, 1984).

$$\sigma_j^{\pm} = \frac{1}{h_j} \begin{bmatrix} (v_p)_{j+1} \pm \sqrt{a_p} (u_p)_{j+1} - (v_p)_j \\ \pm \sqrt{a_p} (u_p)_j \end{bmatrix} \phi(\theta_j^{\pm})$$

$$\theta_j^{\pm} = \frac{(v_p)_j \pm \sqrt{a_p} (u_p)_j - (v_p)_{j-1} \pm \sqrt{a_p} (u_p)_{j-1}}{(v_p)_{j+1} \pm \sqrt{a_p} (u_p)_{j+1} - (v_p)_j \pm \sqrt{a_p} (u_p)_j}$$

Where ϕ is a slope-limiter function given as, (Van, 1979).

$$\phi(\theta) = \frac{|\theta| + \theta}{1 + |\theta|}$$

Solving equations 3.12 for $u_{j+1/2}$ and $v_{j+1/2}$ gives

$$U_{j+\frac{1}{2}} = \frac{1}{2}(U_j + U_{j+1}) - \frac{1}{2}\sqrt{a_p}(V_{j+1} - V_j)$$
$$+ \frac{1}{4}\sqrt{a_p}(h_j \sigma_j^+ + h_{j+1}\sigma_{j+1}^-$$
$$V_{j+\frac{1}{2}} = \frac{1}{2}(V_j + V_{j+1}) - \frac{1}{2}\sqrt{a_p}(U_{j+1} - U_j) \quad \text{....................(3.13)}$$
$$+ \frac{1}{4}\sqrt{a_p}(h_j \sigma_j^+ + h_{j+1}\sigma_{j+1}^-$$

Applying 3.13 in 3.10 we have,

$$\frac{\partial}{\partial t}U + \frac{1}{2h_j}(V_{j+1} - V_{j-1}) - \frac{\sqrt{a_p}}{2h_j}(U_{j+1} - 2U_j + U_{j-1})$$
$$-\frac{1}{4h_j}(h_{j+1}\sigma^-_{j+1} - h_j(\sigma^+_j + \sigma^-_{j-1}) + h_{j-1}\sigma^+_{j-1} = 0$$
$$\partial\!\!\diagup\!\!\partial t\, U + \frac{1}{2h_j}(U_{j+1} - U_{j-1}) - \frac{\sqrt{a_p}}{2h_j}(V_{j+1} - 2V_j + V_{j-1}) \text{..}(3.14)$$
$$-\frac{1}{4h_j}(h_{j+1}\sigma^-_{j+1} + h_j(\sigma^+_j + \sigma^-_{j-1}) - h_{j-1}\sigma^+_{j-1}$$
$$= -\frac{1}{\xi}(V_j - F_{(p)}(U_j))$$

Where $F_{(p)}$ is the p-th component of the flux vector F

Since the one dimensional systems of equation 3.9 has two eigen values $u + p'(\rho),\quad and\ u$ for traffic flow equations and $u \pm a$ for Euler equations, we take

$$\begin{aligned} a_1 &= \sup|u + p'(\rho)|,\ a_2 = \sup|u| \\ a_1 &= \sup|u + a|,\ a_2 = \sup|u - a| \end{aligned},$$

Denoting $D^+W_j = \frac{1}{h_j}(W_{j+\frac{1}{2}} - W_{j-\frac{1}{2}})$

Where W_j^n is the approximate cell average of a quantity W in the cell $\left[x_{j-\frac{1}{2}}, x_{j+\frac{1}{2}}\right]$ at time t_n and by $W^n_{j+\frac{1}{2}}$ the approximate point value of W at $x = x_{j+\frac{1}{2}}$ and $t = t_n$.

Now, to obtain the time discretization for the relaxation scheme we use a second-order TVD Runge-Kutta splitting scheme which was introduced by (Jin, 1995).

Second-order TVD Runge-Kutta splitting scheme to the time derivative in 3.14, yields

$U^* = U^n$,

$V^{*} = V^{n} + \frac{k}{\xi}(V^{*} - F(U^{*}))$;

$U^{(1)} = U^{*} - kD^{+}V^{*}$,

$V^{(1)} = V^{*} - kAD^{+}U^{*}$;

$U^{**} = U^{(1)}$,

$V^{**} = V^{(1)} - \frac{k}{\xi}(V^{**} - F(U^{**})) - 2\frac{k}{\xi}(V^{*} - F(U^{*}))$ $U^{(2)} = U^{**} - kD^{+}V^{**}$,

$V^{(2)} = V^{**} - kAD^{+}U^{**}$;

$U^{n+1} = \frac{1}{2}(U^{n} + U^{(2)})$, $\quad$ $V^{n+1} = \frac{1}{2}(V^{n} + V^{(2)})$;

3.3.2 Godunov Method

Considering the Riemann problem, equation 3.2, with:

Initial condition: $U(x,0) = U_0(x) = \begin{cases} \widetilde{U}_i^n & \text{if } x < 0 \\ \widetilde{U}_{i+1}^n & x > 0 \end{cases}$

Boundary condition: $U(0,t) = U_l(t)$, $\quad$ $U(M,t) = U_r(t)$

Now we discretize the spatial domain into finite volumes $I_i = \left[x_{i-\frac{1}{2}}, x_{i+\frac{1}{2}} \right]$ of regular size.

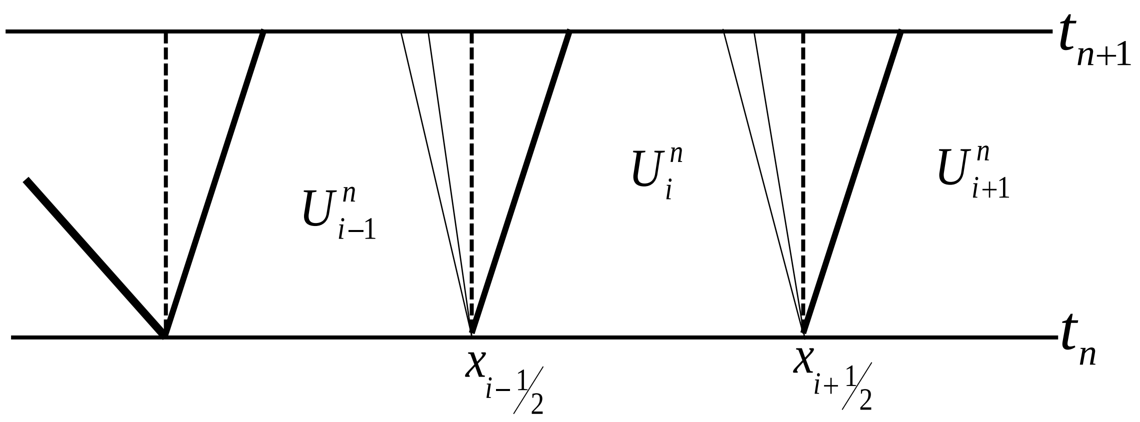


Fig 3.1.5: discretization of the spatial domain.

Now the data at time level n maybe seen as pairs of constant states $\left(U_i^n, U_{i+1}^n\right)$ separated by a discontinuity at the inter cell boundary $x_{i+\frac{1}{2}}$ .In order to admit discontinuous solution, we must use one of the integral forms of the conservation laws (Toro, 2013), that is,

$$\int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} \tilde{U}(x,\Delta t)dx = \int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} \tilde{U}(x,0)dx + \int_0^{\Delta t} F(\tilde{U}(x_{i-\frac{1}{2}},t))dt - \int_0^{\Delta t} F(\tilde{U}(x_{i+\frac{1}{2}},t))dt \qquad (3.15)$$

For the control volume $\left[x_{i-\frac{1}{2}}, x_{i+\frac{1}{2}}\right] \times [0,\Delta t]$ in the domain of interest.

Godunov first assumes a piecewise constant distribution of the data. This is by defining cell averages which produces the desired piecewise constant distribution, that is,

$$U(x,t^n) = U_i^n = \frac{1}{\Delta x}\int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} \tilde{U}(x,t^n)dx \qquad (3.16)$$

Now choosing a time step Δt that is sufficiently small to avoid wave interaction that is by imposing the restriction that

$$\Delta t \le \frac{0.75\Delta x}{S_{\max}^n} \qquad (3.17)$$

where $S_{\max}^n$ is the maximum velocity present throughout the domain at time t^n.

Equation (3.17) allows the interaction of waves from the neighboring Riemann problem during the time step provided the interaction is entirely contained within a mesh cell.

The integrand in (3.16) is an exact solution to the conservation laws and thus applying the integral form to the control volume we can show that $U_{i+\frac{1}{2}}(0)$ is the solution to the Riemann problem $\left(U_i^n, U_{i+1}^n\right)$ along the t-axis in the local domain. Similarly $U_{i-\frac{1}{2}}(0)$ is the solution to the Riemann problem $\left(U_{i-1}^n, U_i^n\right)$ along the t-axis, (Toro, 2013). Thus the Godunov method can be written in conservation form as

$$U_i^{n+1} = U_i^n + \frac{\Delta t}{\Delta x}\left[F_{i-\frac{1}{2}} - F_{i+\frac{1}{2}}\right] \qquad (3.18)$$

With the inter cell numerical flux given by

$$F_{i+\frac{1}{2}}^n = F(U_{i+\frac{1}{2}}(0^-; U_i^n, U_{i+1}^n)) \qquad (3.19)$$

3.4 Numerical Simulation

In order to ensure that shocks and other steep gradients are captured by any numerical scheme that is they move at the right speed even if they are unresolved, we must write the equations in a discrete conservation form as discussed in earlier sections.

3.4.1 Traffic flow Equations

3.4.1.1Relaxation scheme

In order to visualize the features of the numerical scheme presented in the previous section we simulate the AW-Rascle type traffic model equation 3.1 by choosing the following form of the speed adaptation coefficient, (Kimathi, 2012), $P(\rho) = C\ln(\frac{\rho}{1-\rho})$.

where $C = 0.9$

Now we investigate two traffic flow scenarios that are envisaged in two Riemann problems. Thisleads to two solutions of interest namely: a 1-shock wave followed by a 2-contact and a 1-rarefaction followed by a 2-contact.

Taking into consideration a scenario of traffic along the x-axis and beginning at $x=-30$ and ending at $x=30$ with traffic street lights at $x=0$ andletting the direction of flow of traffic be in the direction of increasing x along the axis.

Case 1:

When the traffic light red goes on, the vehicles approaching the street light decrease their velocity which as a result increases the density of the vehicles and thus causing shock wave.

Taking into consideration the left and right traffic states, then the following initial conditions are chosen.

$$\begin{array}{l} \rho_L = 0.6, u_L = 0.95 \\ \rho_R = 0.6, u_R = 0.1 \end{array} \quad \text{.........} (3.20)$$

This data gives rise to a 1-shock wave followed by a 2-contact discontinuity as shown by figure3.2. Note that there is an abrupt change in color in figure3.1 (c), check the color bar, due to the abrupt change in density as shown in figure3.1 (a). Also in figure 3.1 (b) we show the velocity profile at the same final computational time obtained using the initial data (3.20).

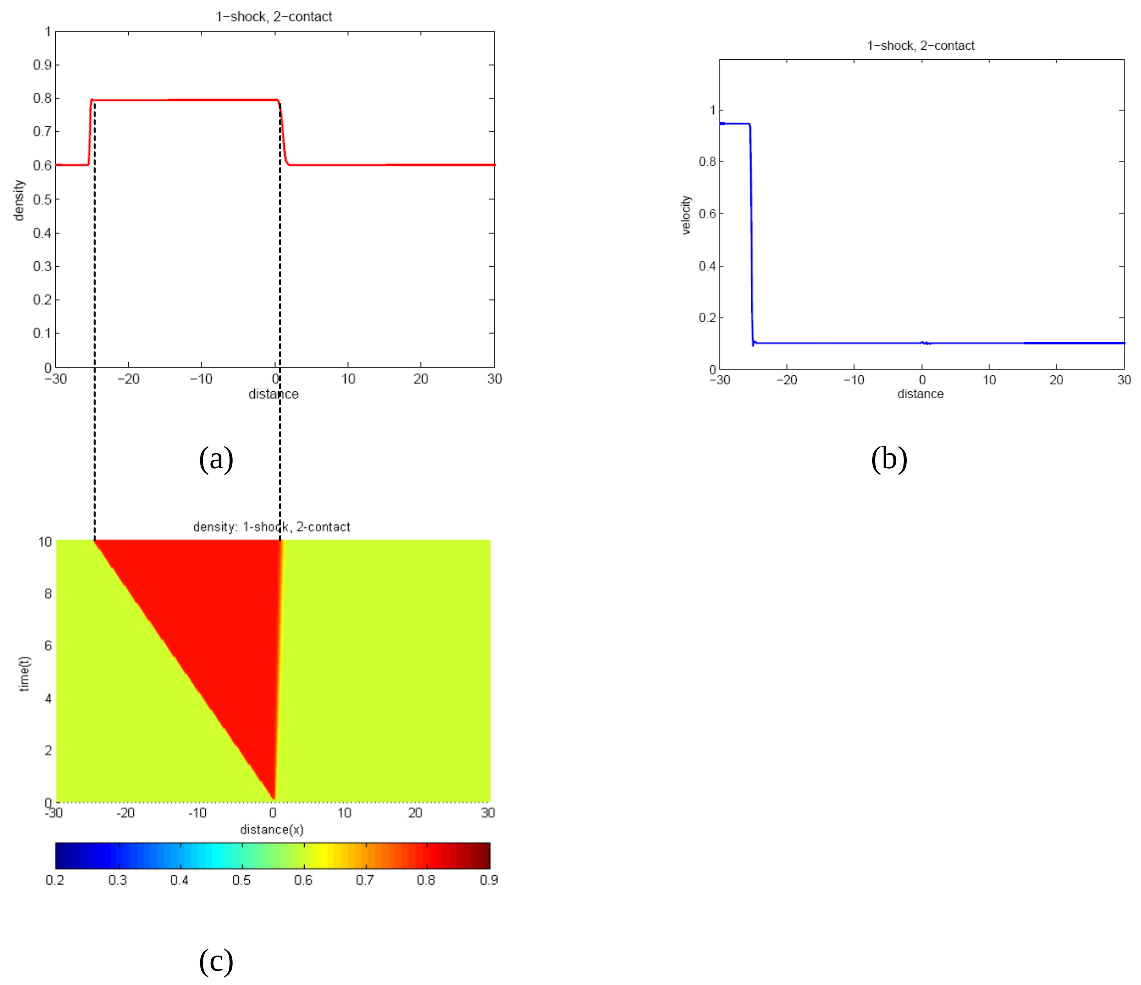


Figure 3.1: The Relaxation scheme solution to the Riemann problem; $\rho_L = 0.6, u_L = 0.95, \rho_R = 0.6, u_R = 0.1, x_0 = 0$

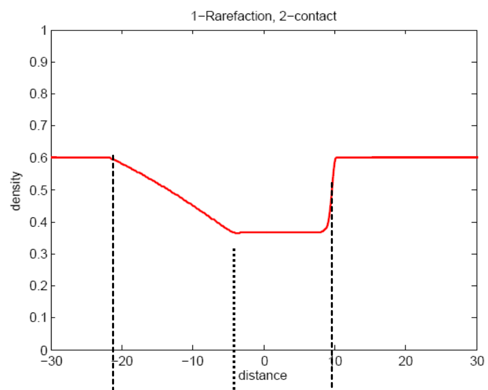
The density increases from point zero (0) abruptly and travels backwards due to the traffic congestion and queueing. Shockwave is the boundary between two traffic states characterized by different densities and is seen to travel towards the negative side of the graph as evident in the density profile in figure 3.1 (a). In traffic flow this is due to the increase in the number of vehicles as they approach the red traffic light.

Case 2:

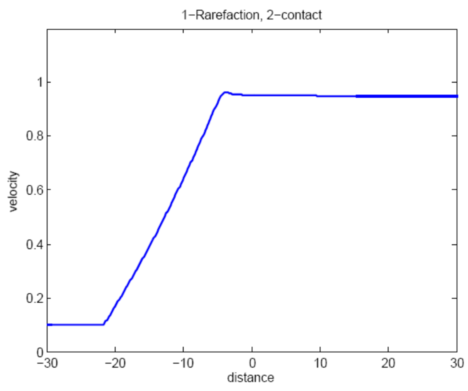
When green traffic light goes on, the first vehicle moves as well asthe subsequent vehicles. One will notice that due to the difference in the reaction time of the different drivers as well as the vehicle power, some vehicles will move faster as compared to others and thus forming a continuous wave known as rarefaction wave.

Taking into consideration the left and right traffic states the following initial conditions are chosen,

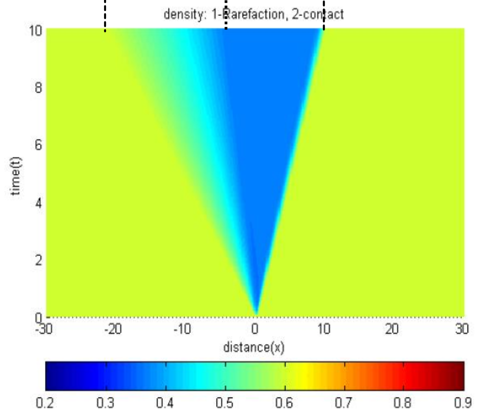
$$\begin{aligned} \rho_L &= 0.6, u_L = 0.1 \\ \rho_R &= 0.6, u_R = 0.95 \end{aligned} \qquad (3.21)$$



(a)



(b)



(c)

Figure3.2: Relaxation scheme solution to the Riemann problem;

$\rho_L = 0.6, u_L = 0.1, \quad \rho_R = 0.6, u_R = 0.95$

This data gives rise to a 1-Rarefaction wave followed by a contact discontinuity as shown by figure 3.2 at final time T=10 obtained using the initial data (3.21) above.

From the color bar it is noted that there is continuous change in color due to the formation of a rarefaction wave.

3.4.1.2 Godunov scheme

In the previous section traffic flow equations have been simulated for two case scenarios using the Relaxation scheme. Similarly, to visualize the features of the Godunov scheme we simulate the AW-Rascle type traffic model equations. In this case we consider the two different scenarios and simulation is done using the Godunov scheme. For case one the initial conditions stated in 3.20 are considered. In this case a 1-shock wave followed by a 2-contact discontinuity are visualized using the Godunov scheme as shown in the figure 3.3 that is for the density and velocity profiles respectively.

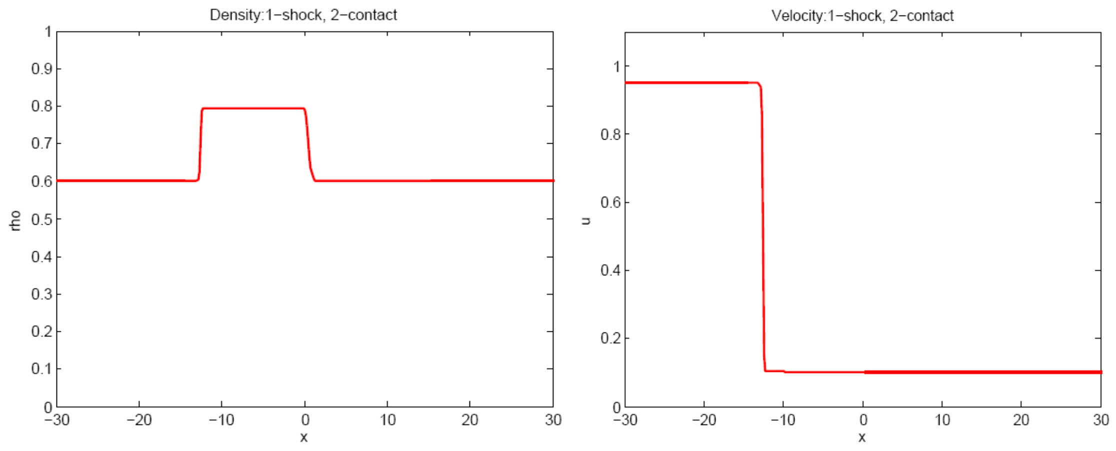


Figure 3.3: Godunov scheme solution to the Riemann problem;

In the second scenario we consider the initial conditions as stated in 3.21, which give rise to a 1-Rarefaction wave followed by a contact discontinuity as shown in figure 3.4 for the density and velocity profiles respectively.

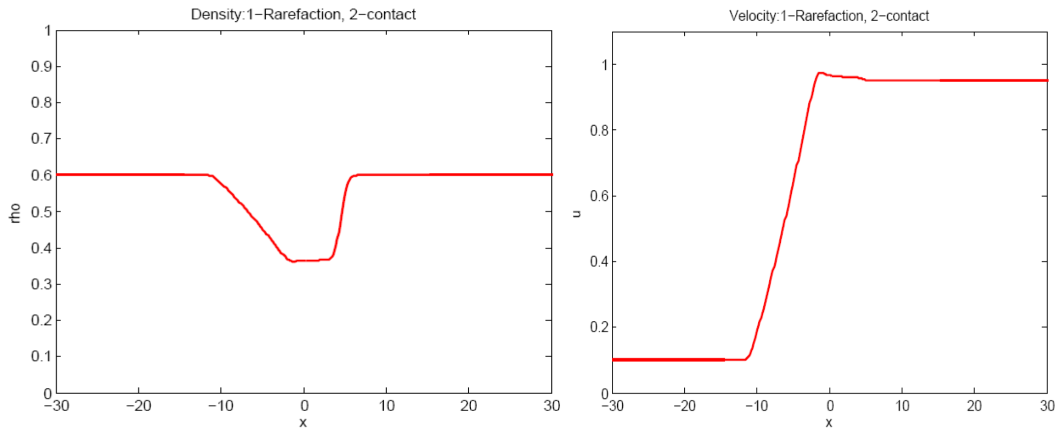


Figure 3.4: Godunov scheme solution to the Riemann problem;

Note that in both cases the computation is done using the conserved variables ρ & $y = \rho u + \rho p(\rho)$. This is essential for determination of the correct intermediate states.

3.4.2 Euler Equations

In general, a hyperbolic system contains discontinuous processes involving contact, shocks and rarefactions. For example, if a gas in a tube initially prepared with a jump in the states (density and velocity) across some surface as the evolution proceeds in time these jumps will break up into a combination of shocks, rarefactions and contacts in addition to any bulk motion and sound waves that may exist or develop.

3.4.2.1 Relaxation Scheme

We use the value of h as 0.005 but choose the time step, k, according to Courant-Friedrichs-Lewy (CFL) condition.

$\dfrac{k}{h|\lambda_{\max}|} \le \dfrac{1}{2}$ Where $\lambda_{\max}$ is the maximal (in absolute value) eigenvalue of the Jacobian matrix A.

Now investigating two Euler flow scenarios that lead to two solutions of interest namely 1-Rarefaction followed by 2-Shock wave and a 1-Shock wave followed by 2-Rarefaction.

In the first scenario we consider the initial data $\rho_L = 0.9, \rho_R = 0.2 \quad v_L = 0.1, v_R = 0.2$ .Figure 3.5 (a),(b), (c) shows the density, velocity and Space – time plot profiles respectively for 1-rarefaction followed by a 2-shock wave.

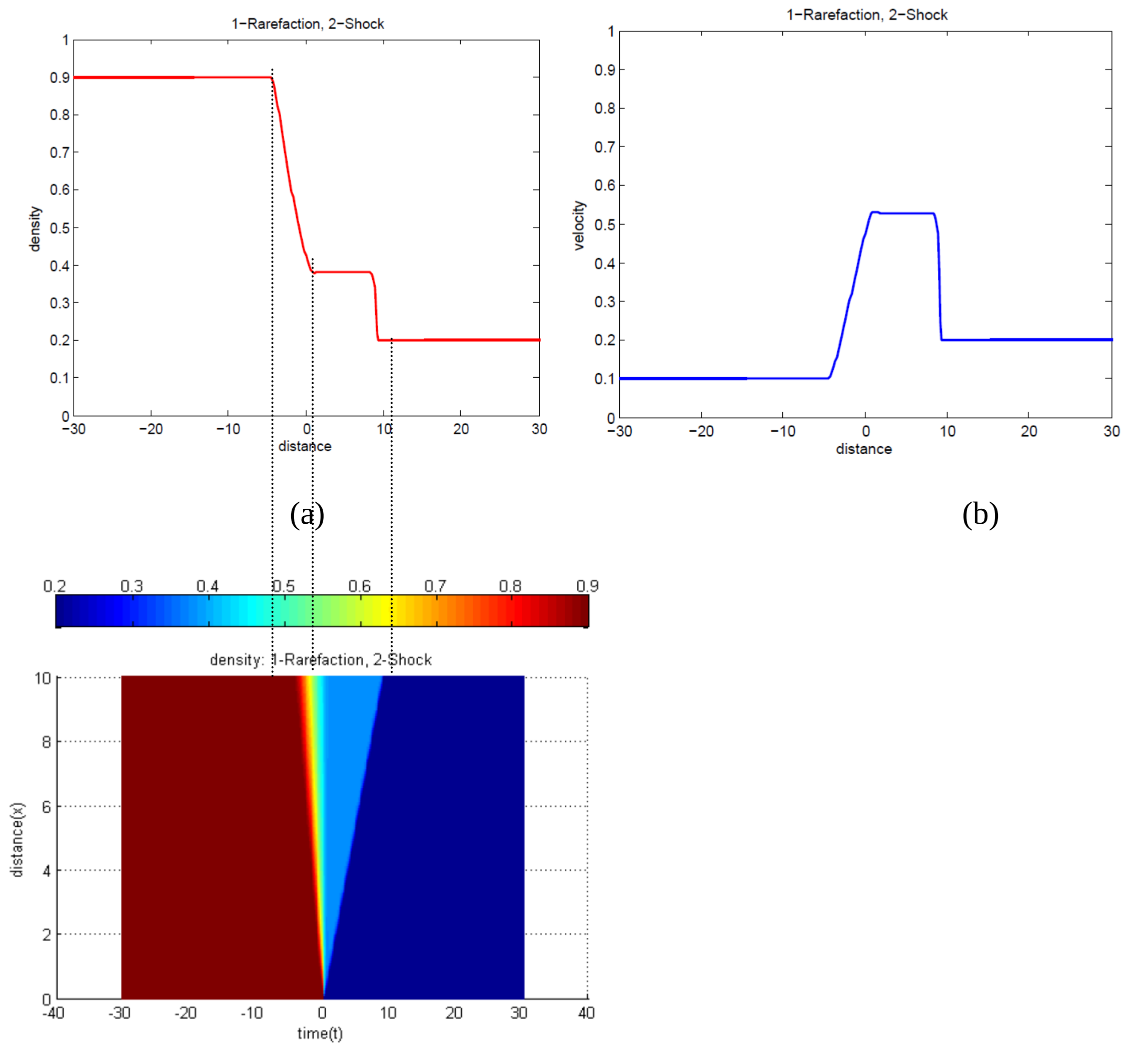


Figure 3.5: Relaxation scheme solution to Euler equation: *1-rarefaction followed by a 2-shock*

In the second scenario we consider the initial data $\rho_L = 0.2, \rho_R = 0.9, v_L = 0.9, v_R = 0.5$. Figure 3.6 (a), (b) and(c) shows the density, velocity profiles and Space – time plot respectively for a 1Shock wave followed by a 2-rarefaction.

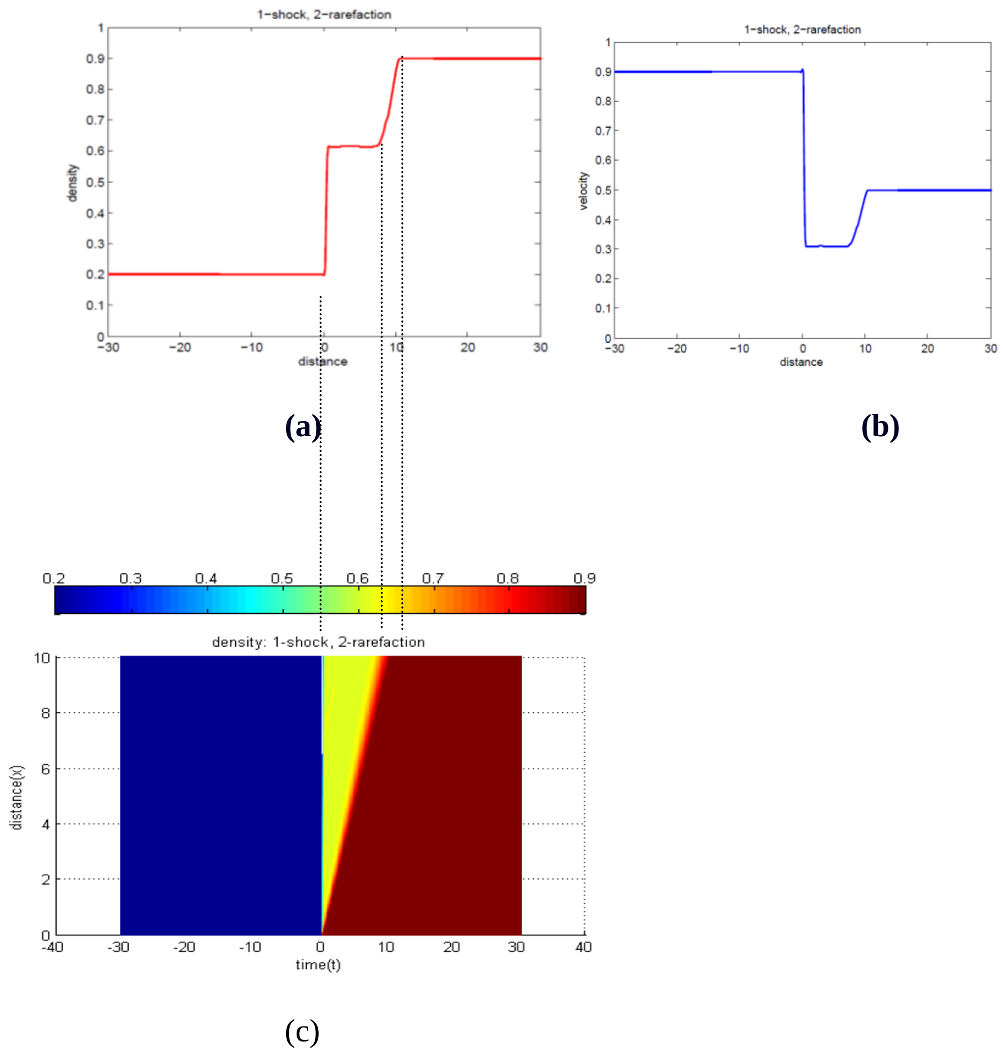


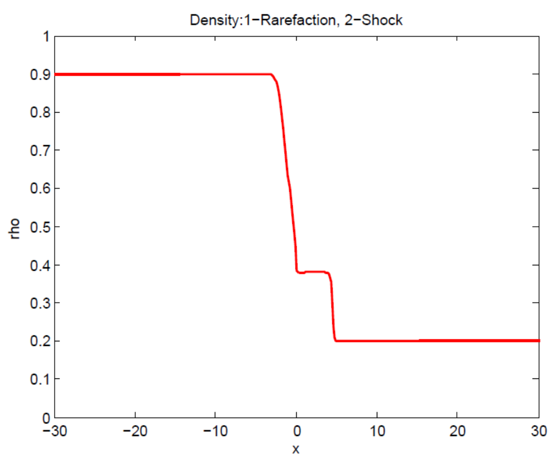
Figure3.6: Relaxation scheme solution to Euler equations: *1-shock followed by a 2-rarefaction*

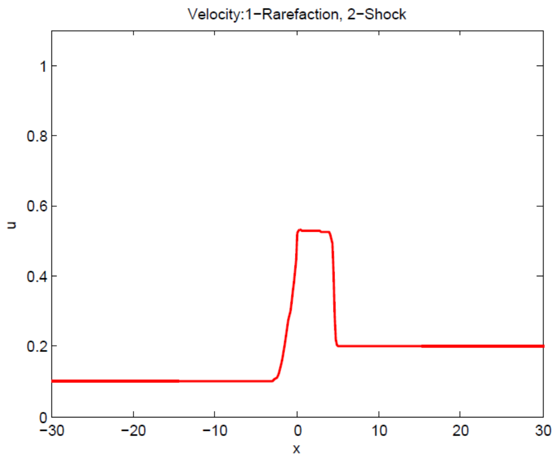
3.4.2.2 Godunov Scheme

We also use the value of h as 0.005 but choose the time step, k, according to Courant-Friedrichs-Lewy (CFL) condition.

Similarly, we also investigate two Euler flow scenarios that lead to two solutions of interest using the Godunov scheme. In the first scenario we consider the initial data

$\rho_L = 0.9, \rho_R = 0.2$ $v_L = 0.1, v_R = 0.2$ .Figure 3.7(a) and (b) shows the density and velocity profiles for the wave solutions respectively.

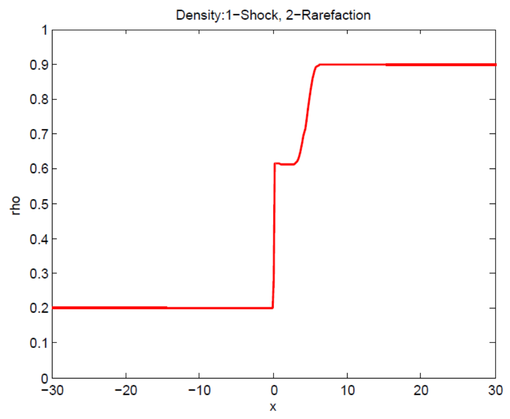




(a) **(b)**

Figure3.7: ***Godunov scheme solution for 1-rarefaction followed by 2-shock***

In the second scenario we consider the initial data $\rho_L = 0.2, \rho_R = 0.9, v_L = 0.9, v_R = 0.5$ to yield a 1-Shock followed by 2-Rarefaction wave as shown below for both density and velocity profiles respectively.



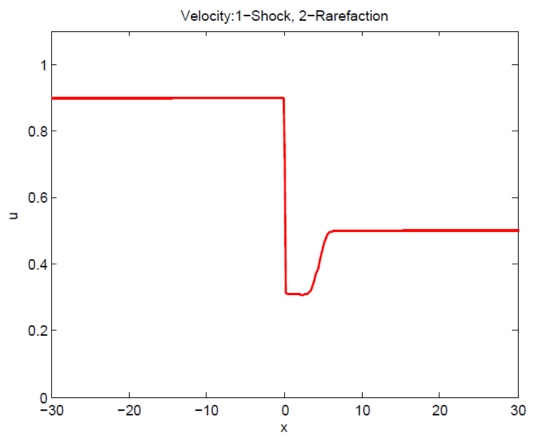


Figure 3.8: ***Godunov scheme solution for 1-shock followed by a 2-rarefaction.***

Under the predetermined conditions as shown in case one, the wave propagates under low pressure section and thus increase the pressure and thus induces a flow in the direction of the shock wave but at a lower velocity than the lead wave forming the rarefaction wave.

3.5 Numerical Comparison

3.5.1 Traffic flow equations

Having considered the velocity and density profiles for the two schemes using the initial conditions as discussed in section 3.3.1, we now present a comparison for the two schemes as shown in the figure3.9 below for 1-Shock followed by 2-contact scenario and figure 3.10 for 1-Shock followed by 2-contact scenario.

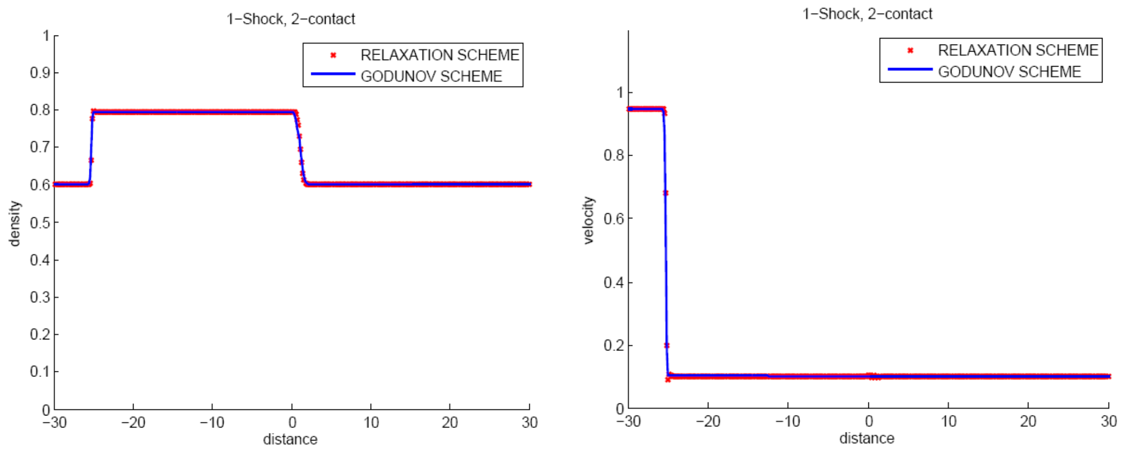


Figure3.9: Density and Velocity profile for 1-Shock followed by 2-Contact respectively

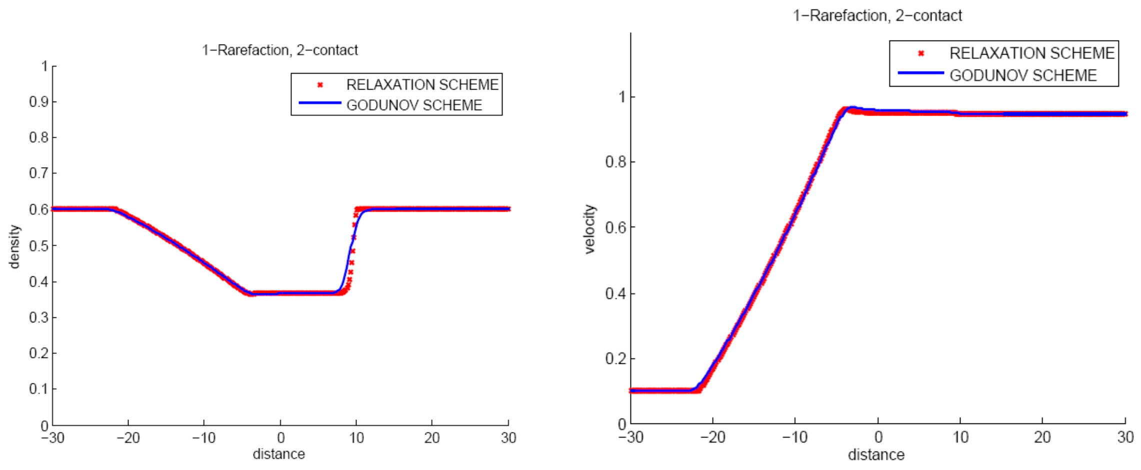


Figure3.10: Density and Velocity profile for 1-Rarefaction followed by 2-Contact respectively

3.5.2 Euler equations

Having considered both the density and velocity profiles for the Euler equations using both the Relaxation and Godunov's schemes as earlier discussed iin section 3.3.2, we now present a comparison for the two schemes as shown in the figures below.

Figure 3.11 shows the density profile as well as velocity profile comparison for both schemes applied to Euler equations for a 1-Rarefaction wave followed by 2- shock wave solution.

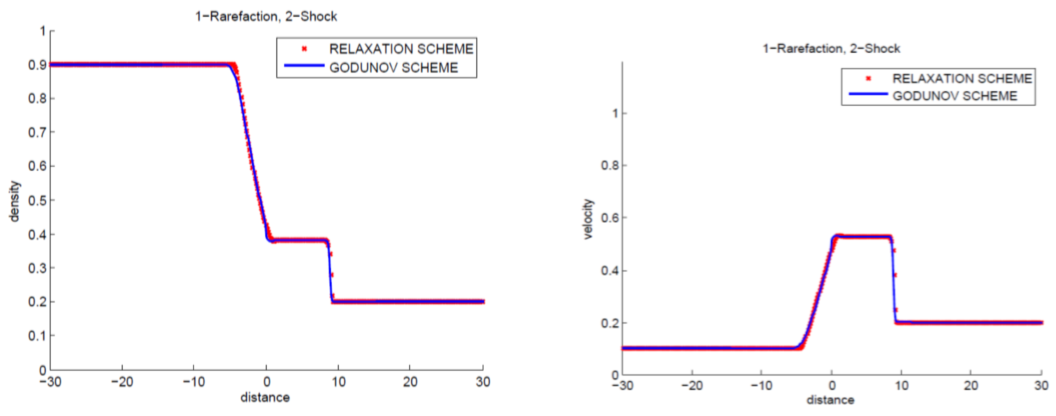


Figure 3.11: Density and Velocity profile for 1-Rarefaction followed by 2-Shock respectively

Figure 3.12 shows the density profile as well as Velocity profile comparison for both schemes applied to Euler equations for a 1-Shock wave followed by 2- Rarefaction wave solution.

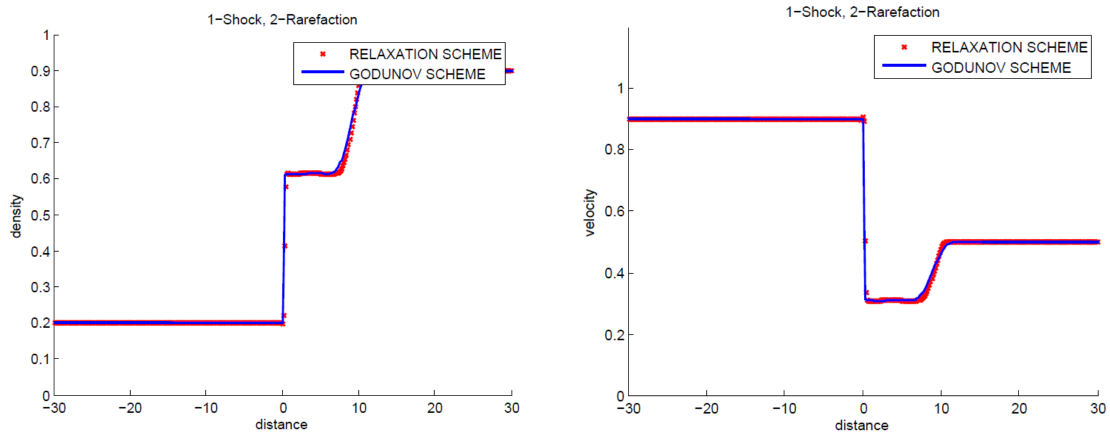


Figure3.12 Density and Velocity profile for 1-Shock followed by 2-Rarefaction respectively

From the above figures we have been able to simulate both relaxation and Godunov schemes using both Euler equations for isothermal gas and AW-Rascle traffic flow model. From the comparison we have been able to show both schemes having corresponding curves and able to simulate the equations the same way. However, relaxation scheme is seen to avoid costly numerical treatment of the non linearities and as well doesn’t require a particular space discretization as compared to Godunov scheme and thus is seen to be advantageous from the point of view of computation time and therefore seems to be very promising as compared to Godunov scheme.

CHAPTER FOUR

CONCLUSION AND RECOMMENDATIONS

4.1 Conclusion

In this study Euler equations for isothermal gas has been analysed as well as a traffic flow model equations. As for the Euler equation, a one dimensional time dependent equation has been considered. The Riemann problem set up gives rise to two real and distinct eigenvalues which together with their corresponding eigenvectors give rise to the different structure waves. Two scenarios namely: 1-shock followed by a 2-rarefaction and 1-rarefaction followed by a 2-shock are simulated depending on the initial conditions set for the Riemann problem. Similarly, the traffic model equations which are of the AW-Rascle type are derived and the Riemann problem is set up. Two different initial data are chosen to give rise to two different scenarios namely: 1-shock followed by 2-contact and 1-rarefaction followed by 2-contact. Later the two scenarios for the equations are simulated using both Godunov and Relaxation schemes and the results are compared graphically. Relaxation scheme has been shown that it avoids a costly numerical treatment of the nonlinearities and as well doesn't require a particular space discretization as compared to Godunov scheme. Moreover from the comparison, Relaxation scheme is noted to perform equally better as the Godunov scheme as shown in the graphs in the previous chapter. Therefore, Relaxation scheme appears to be more promising and a good alternative to the Godunov scheme because of its simplicity.

4.2 Recommendations

Due to insufficient time the author of this thesis considered both numerical schemes that is, Godunov scheme and relaxation schemes and thus the author recommends for comparison of the schemes with the exact solution to check their accuracy so as to show which scheme is more accurate than the other in terms of approximating the exact solutions.

References

Aw, A., & Rascle, M. (2000). Resurrection of" second order" models of traffic flow. *SIAM journal on applied mathematics*, *60*(3), 916-938.

Borsche, R., Kimathi, M., & Klar, A. (2012). A class of multi-phase traffic theories for microscopic, kinetic and continuum traffic models. *Computers & Mathematics with Applications*, *64*(9), 2939-2953.

Evett, J. B., & Liu, C. (1987). *Fundamentals of fluid mechanics*. . New York: McGraw-Hill College.

Hoogendoorn, S. P. (1999). *Multiclass continuum modelling of multilane traffic flow*. TU Delft: Delft University of Technology.

Jin, S., & Xin, Z. (1995). The relaxation schemes for systems of conservation laws in arbitrary space dimensions. *Communications on pure and applied mathematics*, *48*(3), 235-276.

Kaushik, K. N. (2012). *A Low Dissipative Relaxation Scheme For Hyperbolic Consevation Laws.* Bangalore: Indian institute of science.

Kimathi, M. E. M., & Illner, R. (2012). Mathematical models for 3-phase traffic flow theory. *A PhD thesis, University of Kaiserslautern Germany*.

Ksibi, H., & Moussa, A. B. (2008). Numerical simulation of a one-dimensional shock tube problem at supercritical fluid conditions. *International Journal of Physical Sciences, 3(12), 314-320.*

LeVeque, R. J., & Le Veque, R. J. (1992). *Numerical methods for conservation laws* (Vol. 132). Basel: Birkhäuser.

Mengel, A. (2006). Numerical computation of the Shock Tube Problem by means of wave digital principles. *Advances in Radio Science, 4*(7), 161-164.

Mutua, S. K., Kimathi, M. E., Kiogora, P. R., & Mutua, N. M. (2013). A Study of Solutions to Euler Equations for a One Dimensional Unsteady Flow. *American Journal of Computational and Applied Mathematics*, *3*(4), 233-237.

Puppo, G., & Semplice, M. (2011). Entropy and the numerical integration of conservation laws. In *submitted to Proceedings of the 2nd Annual Meeting of the Lebanese Society for Mathematical Sciences*.

Samtaney, R. (1997). Computational methods for self-similar solutions of the compressible Euler equations. *Journal of Computational Physics*, *132*(2), 327-345.

Sweby, P. K. (1984). High resolution schemes using flux limiters for hyperbolic conservation laws. *SIAM journal on numerical analysis*, *21*(5), 995-1011.

Toro, E. F. (2013). *Riemann solvers and numerical methods for fluid dynamics: a practical introduction*. New York: Springer Science & Business Media.

Van Leer, B. (1979). Towards the ultimate conservative difference scheme. V. A second-order sequel to Godunov's method. *Journal of computational Physics*, *32*(1), 101-136.

Wen-Long, W., Hua, L., & Sha, P. (2011). Performance Comparison and Analysis of Different Schemes and Limiters. *World Academy of Science, Engineering and Technology, International Journal of Mathematical Computational Physical, Electrical and Computer Engineering,* 5(7), 974-979

Zhang, H. M. (2002). A non-equilibrium traffic model devoid of gas-like behavior. *Transportation Research Part B: Methodological*, *36*(3), 275-290.

Zhang, H. M. (2000). Structural properties of solutions arising from a nonequilibrium traffic flow theory. *Transportation Research Part B: Methodological*, *34*(7), 583-603.

Zhang, H. M. (1999). A mathematical theory of traffic hysteresis. *Transportation Research Part B: Methodological*, *33*(1), 1-23.

Zhang, H. M. (1998). A theory of nonequilibrium traffic flow. *Transportation Research Part B: Methodological*, *32*(7), 485-498.

Appendix I: Godunov Euler

```
function Godunov_Euler()
  clear all;
  clc;
  tend=10.0;

  plotting=1;   plotFD=1;   plotFD1=0;   plotFD2=5;   plotFD3=-20;   pausing=0;
storeFD1=1;

   lanes1=1;    lanes2=2;    Trelax=5;   rhofree=0.3;    rhojam=0.9;    rhosyn=0.5;
Usyn=0.28;
   boundaryL=0;%0=Neumann,1=Dirichlet
   boundaryR=0;    boundaryRflux=0.01;    boundaryLflux=0.25;    contraction_x=0;
contraction_flux=inf;
   contraction_x_smooth=1.0;   lanefactor=1.5;   clf;

   solver=2;Ao=1;Vmax=0;                Euler={};                Euler.name='Euler';
Euler.FundamentalDiagram=0;
   Euler.plotmarker='k';         Euler.pausing=pausing;         Euler.Solver=solver;   %
Euler.Pressure=pressure;
   Euler.nu=2;%     Euler.C=.7;     Euler.C1=.50;     Euler.Ao=Ao;Euler.Vmax=Vmax;
Euler.h=1;
   Euler.Lanes1=lanes1;
   Euler.Plot=plotting;   Euler.plotvelocity=1;
   Euler.rhofree=rhofree;            Euler.rhojam=rhojam;            Euler.rhosyn=rhosyn;
Euler.Usyn=Usyn;
   Euler.boundaryR=boundaryR;   Euler.boundaryL=boundaryL;
   Euler.contraction_x=contraction_x;
   Euler.lanefactor=lanefactor;      Euler.plotFD=plotFD;      Euler.plotFD1=plotFD1;
Euler.plotFD2=plotFD2;
   Euler.plotFD3=plotFD3;   Euler.storeFD1=storeFD1;
   KIN=Euler;         KIN.name='Euler   model';         KIN.FundamentalDiagram=1;
KIN.plotmarker='m';
   [xKIN,t,uKIN,istoreUKIN,pp,UsurfKIN,tKIN]=Solve_Euler(KIN,tend);
         xpos=KIN.plotFD3;                         [dummy,ipos]=min(abs(xKIN-xpos));
rpos=uKIN(ipos,1);
         vel=uKIN(:,2)./uKIN(:,1);         vpos=vel(ipos);
   [r V1 Rline Jline]=plotFundamentalDiagram(KIN,xpos,rpos,vpos);

  save('KIN.mat','xKIN','tKIN','uKIN','pp','UsurfKIN','tend','KIN');
   if(plotting==1)
       print -depsc Kinetic
```

```
    end

    plotFD=0;
  KIN.plotFD=plotFD;%  KINnew.plotFD=plotFD;
    clf;
%     print -depsc Compare
   instant=min(istoreUKIN);
    c1=floor(.5*instant);
   figure(2)
  %subplot(3,2,1)
  B=1.5;
   RE= plot(xKIN,UsurfKIN(instant-c1,:,3)','r');
  set(RE, 'LineWidth', B);  title('Velocity:');  xlabel('x')  ylabel('u')  axis([-30 30 0.0
1.1])

  figure(3)
  %subplot(3,2,1)
   RE=    plot(xKIN,UsurfKIN(instant-c1,:,1)','r');      set(RE,    'LineWidth',    B);
title('Density:');  xlabel('x')  ylabel('rho')
  axis([-30 30 0.0 1.0])
end

function [x,t,u,istoreU,FD1stored,Usurf,tvector]=Solve_Euler(Problem,tend)
    display(Problem.name)
    nbc=2;  inx=400;  nx=inx+2*nbc;  nu=2;    xlow=-30.0;  xup=30.0;  dx=(xup-
xlow)/(inx-1);
    x=xlow-nbc*dx:dx:xup+nbc*dx;
       % Inital Conditions
    Problem.Lanes=ones(nx,1)*Problem.Lanes1;   u=zeros(nx,nu);
    %RP in the middle of the tube
    icont=find(x>Problem.contraction_x);                          r=0.2*ones(nx,1);
r(icont)=.2*ones(length(icont),1);
    v=.8*ones(nx,1);  v(icont)=.2*ones(length(icont),1);   y=r.*v;  u(:,1)=r;  u(:,2)=y;
    if(Problem.storeFD1==1)
        [dummy,iFD1]=min(abs(x-Problem.plotFD3));                         istoreFD1=1;
FD1stored(istoreFD1,1)=u(iFD1,1);
        FD1stored(istoreFD1,2)=u(iFD1,2);
%FD1stored(istoreFD1,2)=u(iFD1,2)./u(iFD1,1)-press(Problem,u(iFD1,1),iFD1);
istoreFD1=istoreFD1+1;     istoreU=1;     Usurf(istoreU,1:nx,1)=u(1:nx,1);
        Usurf(istoreU,1:nx,3)=u(1:nx,2)./u(1:nx,1);     istoreU=istoreU+1;
    end

    t=0.0;     itvector=1;     tvector(itvector)=t;     itvector=itvector+1;     iteration=0;
maxiteration=10000;
```

```
  dt=1.e-8;    cfl=0.99;    desiredcfl=0.9;
   while ((t<tend)&&(iteration<maxiteration))
      iteration=iteration+1;            dt=min(dt,tend-t);            ustore=u;            %compute
conservation law   %LF
      [u,speed]=Lanes_LF_Step(Problem,nx,nbc,nu,dt,dx,u);             %update boundary
      u=Lanes_boundary(Problem,nx,nbc,u);
      %timestep control
      if(speed<=0)%set dt=0.1 if no speed available
         speed=dx/0.1*desiredcfl;
      end
      if((dt<(dx/speed*cfl)))%accepd timestep
         t=t+dt;                       tvector(itvector)=t;                        itvector=itvector+1;
if(Problem.storeFD1==1)
            FD1stored(istoreFD1,1)=u(iFD1,1);
FD1stored(istoreFD1,2)=u(iFD1,2);
            %FD1stored(istoreFD1,2)=u(iFD1,2)./u(iFD1,1)-
press(Problem,u(iFD1,1),iFD1);
            istoreFD1=istoreFD1+1;
            Usurf(istoreU,1:nx,1)=u(1:nx,1);
Usurf(istoreU,1:nx,3)=(u(1:nx,2)./u(1:nx,1));
            istoreU=istoreU+1;
         end
      else%restore
         u=ustore;
      end
      %set new timestep
      dt=dx/speed*desiredcfl;
          if(Problem.Plot==1)
          plotTraffic(Problem,x,u,t,0,Problem.plotmarker);                          drawnow;
if(Problem.pausing==1)
            pause();
         end
      end
   end
   plotTraffic(Problem,x,u,t,1,Problem.plotmarker);
   if(iteration>=maxiteration)
      display('out of iterations')       pause;
   end
end

function [unew,speed]=Lanes_LF_Step(Problem,nx,nbc,nu,dt,dx,u)
      %generate the random numbers
            nrand=100;%=log2(#choices)
```

```
        ran=randi([0 1],nx-1,nrand);        k=1:nrand;        an=ran*2.^(-k)';        v=(u(1:nx-
    1,2)./u(1:nx-1,1));
        ian=find(an<dt/dx*v);       % solution with only the 2 wave u12
        ustar=zeros(size(u));       ustar(2:nx,:)=starstate(Problem,u(1:nx-1,:),u(2:nx,:),1:nx-
    1,2:nx);
        u12=zeros(size(u));        u12(2:nx,:)=u(2:nx,:);        u12(ian,:)=ustar(ian,:);        %
    adding the 1 wave
      if(Problem.Solver==1)
        fluxm=zeros(size(u));        fluxp=zeros(size(u));                fluxp2=zeros(size(u));
    fluxp1=zeros(size(u));
        [fluxm(2:nx-1,:),s1]=Godunov_flux(Problem,u12(2:nx-1,:),u(3:nx,:),2:nx-1,3:nx);
        [fluxp1(2:nx-1,:),s21]=Godunov_flux(Problem,u(1:nx-2,:),u12(2:nx-1,:),1:nx-
    2,2:nx-1);
        [fluxp2(2:nx-1,:),s22]=Euler_flux(Problem,u12(2:nx-1,:),2:nx-1);
        s2=max(s21,s22);                        ieq=uequal(ustar(2:nx-1,:),u12(2:nx-1,:),1.e-12);
    fluxp(2:nx-1,:)=fluxp1(2:nx-1,:);
        fluxp(ieq,:)=fluxp2(ieq,:);       fluxp(1:nx-2,:)=fluxp(2:nx-1,:);
        fluxm(2:nx-1,:)=fluxm(2:nx-1,:)+dx/dt*(u(2:nx-1,:)-u12(2:nx-1,:));
        % cfl<1/2
        speed=2*max(max(max(abs(s1))),max(max(abs(s2))));

      unew=zeros(size(u));    xin=nbc+1:nx-nbc;    one=ones(size(xin));
      unew(xin,:)=u(xin,:)-dt/dx*(fluxm(xin,:)-fluxp(xin-one,:));
      elseif(Problem.Solver==2)

        [flux(1:nx-1,:),s1]=Godunov_flux(Problem,u(1:nx-1,:),u(2:nx,:),1:nx-1,2:nx);

        speed=max(max(abs(s1)));

      unew=zeros(size(u));    xin=nbc+1:nx-nbc;    one=ones(size(xin));
      unew(xin,:)=u(xin,:)-dt/dx*(flux(xin,:)-flux(xin-one,:));
      else
        disp('No Solver specified');
        pause;
      end
    end
    function unew=Lanes_boundary(Problem,nx,nbc,u)
      unew=u;
      %left
      bcL=Problem.boundaryL;    if(bcL==1)       %Dirichlet
        ubc=u(nbc+1,:);       flux=Problem.boundaryLflux;
        ubc(1,2)=flux+ubc(1,1)*press(Problem,ubc(1,1),1);
        for ibc=1:nbc           unew(ibc,:)=ubc;
        end
```

```
   else
      %zero Neumann at the left end
      ubc=u(nbc+1,:);        for ibc=1:nbc            unew(ibc,:)=ubc;
      end
   end
   %right
   bcR=Problem.boundaryR;
   %zero Neumann at the right end
   if(bcR==1)
      %Dirichlet
      ubc=u(nx-nbc,:);        flux=Problem.boundaryRflux;
      ubc(1,2)=flux+ubc(1,1)*press(Problem,ubc(1,1),1);
      for ibc=nx-nbc+1:nx            unew(ibc,:)=ubc;
      end
   else
      ubc=u(nx-nbc,:);
      for ibc=nx-nbc+1:nx
         unew(ibc,:)=ubc;
      end
   end
end

function plotTraffic(Problem,x,u,t,holdon,color)

   if(Problem.plotFD==1)
      ncol=3;
   else
      ncol=2;
   end
   nline=3;    icol=1;    iline=1;
   subplot(nline,ncol,[(iline-1)*ncol+icol (iline-1)*ncol+icol+1])
   icol=icol+2;
   if(holdon==1)
      hold on;
   end
   plot(x,u(:,1),color)    ylabel('rho');    ax=axis;    axis([-30 30 0.0 1.0])
   if(holdon==1)
      hold off;
   end

   if(Problem.plotFD==1)
      subplot(nline,ncol,(iline-1)*ncol+icol)

      xpos=Problem.plotFD1;
      [dummy,ipos]=min(abs(x-xpos));
      rpos=u(ipos,1);

      vel=(u(:,2)./u(:,1));

      vpos=vel(ipos);
      plotFundamentalDiagram(Problem,xpos,rpos,vpos);
   end

   iline=iline+1;        icol=1;        subplot(nline,ncol,[(iline-1)*ncol+icol   (iline-
1)*ncol+icol+1])    icol=icol+2;
   if(holdon==1)
      hold on;
   end

   if(Problem.plotvelocity==1)
         vel=(u(:,2)./u(:,1));         plot(x,vel,color)        ylabel('v')        ax=axis;
axis([-30 30 0.0 1.0])
   else
      plot(x,u(:,2),color)      ylabel('y')
   end
   title(['time= ' num2str(t)])
   if(holdon==1)
      hold off;
   end
   if(Problem.plotFD==1)
      subplot(nline,ncol,(iline-1)*ncol+icol)
      xpos=Problem.plotFD2;      [dummy,ipos]=min(abs(x-xpos));      rpos=u(ipos,1);
          vel=(u(:,2)./u(:,1));
          vpos=vel(ipos);
      plotFundamentalDiagram(Problem,xpos,rpos,vpos);
   end

   if(nline>2)
      iline=iline+1;        icol=1;        subplot(nline,ncol,[(iline-1)*ncol+icol  (iline-
1)*ncol+icol+1])      icol=icol+2;
      if(holdon==1)
         hold on;
      end
      if(Problem.plotvelocity==1)
         vel=(u(:,2)./u(:,1));
         plot(x,vel.*u(:,1),color)        ylabel('Q')        ax=axis;        axis([-30 30 0.0
0.5])
      end
```

```
        if(holdon==1)
            hold off;
        end

        if(Problem.plotFD==1)
            subplot(nline,ncol,(iline-1)*ncol+icol)
            xpos=Problem.plotFD3;                                    [dummy,ipos]=min(abs(x-xpos));
rpos=u(ipos,1);
             vel=(u(:,2)./u(:,1));             vpos=vel(ipos);
            plotFundamentalDiagram(Problem,xpos,rpos,vpos);
        end

    end
    drawnow;
end

function [r V1  Rline Jline]=plotFundamentalDiagram(Prob,xpos,rpos,vpos)
    nr=500;    r=0:1/(nr-1):1;
    rhofree=Prob.rhofree;    rhojam=Prob.rhojam;    rhosyn=Prob.rhosyn;
    irfree=find(r<rhofree);                                    irsync=find((r>=rhofree)&(r<=rhosyn));
irsync2=find((r>rhosyn)&(r<=rhojam));
    irjam=find(r>rhojam);    rsync=r(irsync);

    % V1
    V0=0.85;c=2.9;    V1func=@(rho)V0*tanh((.45./rho-0.02)/(c*V0));    V1=V1func(r);

    %Jamline
    V0s=0.5;cs=2.9;                    Jlinefunc=@(rho)V0s*tanh((.45*(1./rho-1.1))/(cs*V0s));
Jline=Jlinefunc(r);

    Jline(irfree)=V1(irfree);     V1(irsync2)=Jline(irsync2);
    Jline(irjam)=max(zeros(size(irjam)),Jline(irjam));
V1(irjam)=max(zeros(size(irjam)),Jline(irjam));

    %Rline
    Rline=zeros(size(r));    slope=(V1func(rhosyn)-Jlinefunc(rhofree))/(rhosyn-rhofree);
    Rline(irsync)=(r(irsync)-
rhofree*ones(size(r(irsync))))*slope+Jlinefunc(rhofree)*ones(size(r(irsync)));
    Rline(irsync2)=ones(size(irsync2));    Rline(irjam)=ones(size(irjam));

    if(Prob.contraction_x<xpos)
        lanefactor=Prob.lanefactor;        r=r/lanefactor;
    end
end
```

```
function ieq=uequal(u1,u2,diff)
   ieq=1:length(u1(:,1));        for  iu=1:length(u1(1,:))                ieq1=find(abs(u1(:,iu)-
u2(:,iu))<diff);        ieq=intersect(ieq,ieq1);
   end
end

function ivq=vequal(v1,v2,diff)
   ivq=1:length(v1(:,1));                                  ivq1=find(abs(v1(:,1)-v2(:,1))<diff);
ivq=intersect(ivq,ivq1);
end

function [fG,s]=Godunov_flux(Problem,ul,ur,intervall,intervalr)
   %find the new state
   speed=zeros(length(ul),3);%the first wave has two entries for the head and tail of a
rarefaction
   ustar=starstate(Problem,ul,ur,intervall,intervalr);
   intervalm=intervall;
   rr=ur(:,1);    rl=ul(:,1);    rm=ustar(:,1);    yr=ur(:,2);    yl=ul(:,2);    ym=ustar(:,2);
vl=yl./rl;
   vr=yr./rr;    vm=ym./rm;
   izerol=find(rl<=0);          izeror=find(rr<=0);          vl(izerol)=zeros(length(izerol),1);
vr(izeror)=zeros(length(izeror),1);
   izerom=find(rm<=0);                               rm(izerom)=zeros(length(izerom),1);
vm(izerom)=vl(izerom);
  C1=Problem.C1;
   speed(:,3)=vr+C1*ones(size(speed(:,3)));
   itype1=find((vl-vm)>0);
   itype2=setdiff((1:length(ul)),itype1);
   speed(itype1,1)=(vl(itype1).*rl(itype1)-vm(itype1).*rm(itype1))./(rl(itype1)-
rm(itype1));
   speed(itype1,2)=(vl(itype1).*rl(itype1)-vm(itype1).*rm(itype1))./(rl(itype1)-
rm(itype1));
   speed(itype2,1)=vl(itype2)-C1*ones(size(speed(itype2,1)));
   speed(itype2,2)=vm(itype2)-C1*ones(size(speed(itype2,2)));
   %find correct 'Godunov'-states according to the speeds
   il=find(speed(:,1)>0);                                          istar=find(speed(:,2)<=0);
irare=setdiff(setdiff(1:length(ul),il),istar);
      uG(il,:)=ul(il,:);    uG(istar,:)=ustar(istar,:);       rhs=[0 0];
   options=optimset('Display','off');
   for i=irare
      qinit=[rl(i) vl(i)];        rhs(1)=vl(i)+pressNL(Problem,rl(i),i);
       rare=fsolve(@(q)rarefactionshape(Problem,q,rhs,i),qinit,options);
      uG(i,1)=rare(1);       uG(i,2)=rare(1)*rare(2);
```

```
    end
    [fG,s]=Euler_flux(Problem,uG,intervall);
end
function res=rarefactionshape(Problem,q,rhs,interval)
     %riemann invariant
     C1=Problem.C1; res(1)=q(2)+pressNL(Problem,q(1),interval)-rhs(1);
    %slope of the characteristic = first eigenvalue
     res(2)=q(2)-C1-rhs(2);
end
function [f,s]=Euler_flux(Problem,u,interval)
   C1=Problem.C1;    f=zeros(size(u));    r=u(:,1);    y=u(:,2);    p=C1*C1*r;    v=y./r;
f(:,1)=y;    f(:,2)=y.*v+p;
    s=zeros(size(u));    s(:,1)=v-C1*ones(size(u(:,1)));    s(:,2)=v+C1*ones(size(u(:,1)));
end
function plog=pressNL(Problem,rho,interval)
    C1=Problem.C1;    plog=C1*log(rho);
end
function p=press(Problem,rho,interval)
    C1=Problem.C1;    p=C1*C1*rho;
end
function rho=invpress(Problem,p,interval)
    C1=Problem.C1;   rho=p./(C1*C1);
  end
function p=pressprime(Problem,rho,interval)
    C1=Problem.C1;       p=C1*C1;
  end
function ustar=starstate(Problem,uL,uR,intervalL,intervalR)
    rL=uL(:,1);        yL=uL(:,2);     rR=uR(:,1);        yR=uR(:,2);     val=yL./rL-(yR./rR);
rstar=invpress(Problem,val,intervalL);          ystar=rstar./rL.*yL;          ustar(:,1)=rstar;
ustar(:,2)=ystar;
end
```

Appendix II Godunov Traffic

```
% solve the Aw Rascle model
function Godunov()
  clear all;
  clc;
  tend=10.0;

  plotting=1;   plotFD=1;   plotFD1=0;   plotFD2=5;   plotFD3=-20;   pausing=0;
   storeFD1=1;

   lanes1=1;    lanes2=2;    Trelax=5;    rhofree=0.3;    rhojam=0.9;    rhosyn=0.5;
Usyn=0.28;
   boundaryL=0;%0=Neumann,1=Dirichlet
   boundaryR=0;   boundaryRflux=0.01;   boundaryLflux=0.25;   contraction_x=0;
   contraction_flux=inf;   contraction_x_smooth=1.0;   lanefactor=1.5;
   clf;

   solver=2;Ao=1;Vmax=0;          AwRascle={};          AwRascle.name='AwRascle';
AwRascle.FundamentalDiagram=0;
   AwRascle.plotmarker='k';   AwRascle.pausing=pausing;   AwRascle.Solver=solver;
%   AwRascle.Pressure=pressure;
   AwRascle.nu=2;                    AwRascle.C=.7;                    AwRascle.C1=.9;
AwRascle.Ao=Ao;AwRascle.Vmax=Vmax;
   AwRascle.h=1;              AwRascle.Vref=.45;              AwRascle.Lanes1=lanes1;
AwRascle.Lanes2=lanes2;
   AwRascle.Plot=plotting;      AwRascle.plotvelocity=1;      AwRascle.Trelax=Trelax;
AwRascle.rhofree=rhofree;
   AwRascle.rhojam=rhojam;      AwRascle.rhosyn=rhosyn;      AwRascle.Usyn=Usyn;
AwRascle.boundaryR=boundaryR;   AwRascle.boundaryL=boundaryL;
   AwRascle.boundaryRflux=boundaryRflux;
AwRascle.boundaryLflux=boundaryLflux;
   AwRascle.contraction_flux=contraction_flux;
AwRascle.contraction_x=contraction_x;
   AwRascle.contraction_x_smooth=contraction_x_smooth;
AwRascle.lanefactor=lanefactor;
   AwRascle.plotFD=plotFD;                              AwRascle.plotFD1=plotFD1;
AwRascle.plotFD2=plotFD2;
   AwRascle.plotFD3=plotFD3;   AwRascle.storeFD1=storeFD1;
  %    KIN={};
   KIN=AwRascle;       KIN.name='kinetic  model';       KIN.FundamentalDiagram=1;
KIN.plotmarker='m';
   [xKIN,t,uKIN,istoreUKIN,pp,UsurfKIN,tKIN]=Solve_Lanes(KIN,tend);
```

```
          xpos=KIN.plotFD3;                              [dummy,ipos]=min(abs(xKIN-xpos));
rpos=uKIN(ipos,1);
          vel=uKIN(:,2)./uKIN(:,1)-pressNL(KIN,uKIN(:,1));          vpos=vel(ipos);
   [r V1 Rline Jline]=plotFundamentalDiagram(KIN,xpos,rpos,vpos);
   save('KIN.mat','xKIN','tKIN','uKIN','pp','UsurfKIN','tend','KIN');
   if(plotting==1)
      print -depsc Kinetic
   end
plotFD=0;       KIN.plotFD=plotFD;
   clf;
instant=min(istoreUKIN);
   c1=floor(.5*instant);
  figure(2)
B=1.5;
  RE= plot(xKIN,UsurfKIN(instant-c1,:,3)','r');       set(RE, 'LineWidth', B);
 title('Velocity:1-shock, 2-contact');   xlabel('x')   ylabel('u')  axis([-30 30 0.0 1.1])

 figure(3)
 %subplot(3,2,1)
  RE= plot(xKIN,UsurfKIN(instant-c1,:,1)','r');       set(RE, 'LineWidth', B);
  title('Density:1-shock, 2-contact');   xlabel('x')   ylabel('rho')  axis([-30 30 0.0 1.0])
end
function [x,t,u,istoreU,FD1stored,Usurf,tvector]=Solve_Lanes(Problem,tend)
   display(Problem.name)
   nbc=2;    inx=400;    nx=inx+2*nbc;    nu=2;
   xlow=-30.0;    xup=30.0;    dx=(xup-xlow)/(inx-1);
   x=xlow-nbc*dx:dx:xup+nbc*dx;
     % Inital Conditions
   Problem.Lanes=ones(nx,1)*Problem.Lanes1;    u=zeros(nx,nu);
   %RP in the middle of the tube
   icont=find(x>Problem.contraction_x);
   r=0.6*ones(nx,1);
   r(icont)=.6*ones(length(icont),1);                                   v=.1*ones(nx,1);
v(icont)=.95*ones(length(icont),1);
   y=r.*v+r.*press(Problem,r,1:nx);
   u(:,1)=r;    u(:,2)=y;
   if(Problem.storeFD1==1)
      [dummy,iFD1]=min(abs(x-Problem.plotFD3));
      istoreFD1=1;        FD1stored(istoreFD1,1)=u(iFD1,1);
      FD1stored(istoreFD1,2)=u(iFD1,2);
      %FD1stored(istoreFD1,2)=u(iFD1,2)./u(iFD1,1)-press(Problem,u(iFD1,1),iFD1);
      istoreFD1=istoreFD1+1;       istoreU=1;       Usurf(istoreU,1:nx,1)=u(1:nx,1);
      Usurf(istoreU,1:nx,3)=(u(1:nx,2)./u(1:nx,1)-press(Problem,u(1:nx,1),1:nx));
      istoreU=istoreU+1;
   end

   t=0.0;   itvector=1;   tvector(itvector)=t;   itvector=itvector+1;   iteration=0;
   maxiteration=10000;  dt=1.e-8;   cfl=0.99;   desiredcfl=0.9;

   while ((t<tend)&&(iteration<maxiteration))
      iteration=iteration+1;      dt=min(dt,tend-t);      ustore=u;
      %compute conservation law   %LF
      [u,speed]=Lanes_LF_Step(Problem,nx,nbc,nu,dt,dx,u);
      %update boundary
      u=Lanes_boundary(Problem,nx,nbc,u);
      %timestep control
      if(speed<=0)%set dt=0.1 if no speed available
          speed=dx/0.1*desiredcfl;
      end
      if((dt<(dx/speed*cfl)))%accepd timestep
          t=t+dt;
          tvector(itvector)=t;          itvector=itvector+1;
          if(Problem.storeFD1==1)
            FD1stored(istoreFD1,1)=u(iFD1,1);
FD1stored(istoreFD1,2)=u(iFD1,2);
            istoreFD1=istoreFD1+1;              Usurf(istoreU,1:nx,1)=u(1:nx,1);
          Usurf(istoreU,1:nx,3)=(u(1:nx,2)./u(1:nx,1)-press(Problem,u(1:nx,1),1:nx));
            istoreU=istoreU+1;
          end
      else %restore
          u=ustore;
      end
      %set new timestep
      dt=dx/speed*desiredcfl;
      if(Problem.Plot==1)
          plotTraffic(Problem,x,u,t,0,Problem.plotmarker);          drawnow;
          if(Problem.pausing==1)
            pause();
          end
      end
   dt
   end
  plotTraffic(Problem,x,u,t,1,Problem.plotmarker);
   if(iteration>=maxiteration)
      display('out of iterations')       pause;
   end
end
```

```
function [unew,speed]=Lanes_LF_Step(Problem,nx,nbc,nu,dt,dx,u)
      %Glimm
       %generate the random numbers
       nrand=100;%=log2(#choices)
       ran=randi([0 1],nx-1,nrand);      k=1:nrand;      an=ran*2.^(-k)';
       v=(u(1:nx-1,2)./u(1:nx-1,1)-press(Problem,u(1:nx-1,1),1:nx-1));
       ian=find(an<dt/dx*v);
       % solution with only the 2 wave u12
       ustar=zeros(size(u));       ustar(2:nx,:)=starstate(Problem,u(1:nx-1,:),u(2:nx,:),1:nx-
1,2:nx);
       u12=zeros(size(u));      u12(2:nx,:)=u(2:nx,:);      u12(ian,:)=ustar(ian,:);      %
adding the 1 wave
     %  solver=solverchoice(v,nx);
    if(Problem.Solver==1)
       fluxm=zeros(size(u));          fluxp=zeros(size(u));          fluxp2=zeros(size(u));
fluxp1=zeros(size(u));
       [fluxm(2:nx-1,:),s1]=Godunov_flux(Problem,u12(2:nx-1,:),u(3:nx,:),2:nx-1,3:nx);
       [fluxp1(2:nx-1,:),s21]=Godunov_flux(Problem,u(1:nx-2,:),u12(2:nx-1,:),1:nx-
2,2:nx-1);
       [fluxp2(2:nx-1,:),s22]=AwRascle_flux(Problem,u12(2:nx-1,:),2:nx-1);
       s2=max(s21,s22);
       ieq=uequal(ustar(2:nx-1,:),u12(2:nx-1,:),1.e-12);      fluxp(2:nx-1,:)=fluxp1(2:nx-
1,:);
       fluxp(ieq,:)=fluxp2(ieq,:);      fluxp(1:nx-2,:)=fluxp(2:nx-1,:);
       fluxm(2:nx-1,:)=fluxm(2:nx-1,:)+dx/dt*(u(2:nx-1,:)-u12(2:nx-1,:));
       % cfl<1/2
       speed=2*max(max(max(abs(s1))),max(max(abs(s2))));
    unew=zeros(size(u));   xin=nbc+1:nx-nbc;   one=ones(size(xin));
    unew(xin,:)=u(xin,:)-dt/dx*(fluxm(xin,:)-fluxp(xin-one,:));
    elseif(Problem.Solver==2)
       [flux(1:nx-1,:),s1]=Godunov_flux(Problem,u(1:nx-1,:),u(2:nx,:),1:nx-1,2:nx);
       speed=max(max(abs(s1)));   unew=zeros(size(u));   xin=nbc+1:nx-nbc;
    one=ones(size(xin));   unew(xin,:)=u(xin,:)-dt/dx*(flux(xin,:)-flux(xin-one,:));
    else
       disp('No Solver specified');
       pause;
    end
end
function unew=Lanes_boundary(Problem,nx,nbc,u)
    unew=u;
    %left
    bcL=Problem.boundaryL;
    if(bcL==1)
       %Dirichlet
       ubc=u(nbc+1,:);       flux=Problem.boundaryLflux;
       ubc(1,2)=flux+ubc(1,1)*press(Problem,ubc(1,1),1);
       for ibc=1:nbc
          unew(ibc,:)=ubc;
       end
    else
       %zero Neumann at the left end
       ubc=u(nbc+1,:);
       for ibc=1:nbc
          unew(ibc,:)=ubc;
       end
    end
    %right
    bcR=Problem.boundaryR;
    %zero Neumann at the right end
    if(bcR==1)
       %Dirichlet
       ubc=u(nx-nbc,:);       flux=Problem.boundaryRflux;
       ubc(1,2)=flux+ubc(1,1)*press(Problem,ubc(1,1),1);
       for ibc=nx-nbc+1:nx
          unew(ibc,:)=ubc;
       end
    else
       ubc=u(nx-nbc,:);
       for ibc=nx-nbc+1:nx
          unew(ibc,:)=ubc;
       end
    end
end
function plotTraffic(Problem,x,u,t,holdon,color)
    if(Problem.plotFD==1)
       ncol=3;
    else
       ncol=2;
    end
    nline=3;       icol=1;       iline=1;       subplot(nline,ncol,[(iline-1)*ncol+icol  (iline-
1)*ncol+icol+1])    icol=icol+2;
    if(holdon==1)
       hold on;
    end
    plot(x,u(:,1),color)    ylabel('rho');    ax=axis;    axis([-30 30 0.0 1.0])
    if(holdon==1)
       hold off;
    end
    if(Problem.plotFD==1)
       subplot(nline,ncol,(iline-1)*ncol+icol)
       xpos=Problem.plotFD1;      [dummy,ipos]=min(abs(x-xpos));      rpos=u(ipos,1);
       vel=(u(:,2)./u(:,1)-pressNL(Problem,u(:,1)));      vpos=vel(ipos);
       plotFundamentalDiagram(Problem,xpos,rpos,vpos);
    end
    iline=iline+1;            icol=1;            subplot(nline,ncol,[(iline-1)*ncol+icol    (iline-
1)*ncol+icol+1])    icol=icol+2;
    if(holdon==1)
       hold on;
    end
    if(Problem.plotvelocity==1)
       vel=(u(:,2)./u(:,1)-pressNL(Problem,u(:,1)));
       plot(x,vel,color)      ylabel('v')      ax=axis;      axis([-30 30 0.0 1.0])
    else
       plot(x,u(:,2),color)      ylabel('y')
    end
    title(['time= ' num2str(t)])
    if(holdon==1)
       hold off;
    end
    if(Problem.plotFD==1)
       subplot(nline,ncol,(iline-1)*ncol+icol)
       xpos=Problem.plotFD2;      [dummy,ipos]=min(abs(x-xpos));      rpos=u(ipos,1);
       vel=(u(:,2)./u(:,1)-pressNL(Problem,u(:,1)));      vpos=vel(ipos);
       plotFundamentalDiagram(Problem,xpos,rpos,vpos);
    end
    if(nline>2)
       iline=iline+1;            icol=1;            subplot(nline,ncol,[(iline-1)*ncol+icol  (iline-
1)*ncol+icol+1])      icol=icol+2;
       if(holdon==1)
          hold on;
       end
       if(Problem.plotvelocity==1)
      vel=(u(:,2)./u(:,1)-pressNL(Problem,u(:,1)));                  plot(x,vel.*u(:,1),color)
ylabel('Q')          ax=axis;
          axis([-30 30 0.0 0.5])
       end
       if(holdon==1)
          hold off;
       end
             if(Problem.plotFD==1)
          subplot(nline,ncol,(iline-1)*ncol+icol)
```

```
          xpos=Problem.plotFD3;                              [dummy,ipos]=min(abs(x-xpos));
rpos=u(ipos,1);
       vel=(u(:,2)./u(:,1)-pressNL(Problem,u(:,1)));
          vpos=vel(ipos);           plotFundamentalDiagram(Problem,xpos,rpos,vpos);
       end
   end
   drawnow;
end
function [r V1  Rline Jline]=plotFundamentalDiagram(Prob,xpos,rpos,vpos)
   nr=500;
   r=0:1/(nr-1):1;                   rhofree=Prob.rhofree;                   rhojam=Prob.rhojam;
rhosyn=Prob.rhosyn;
   irfree=find(r<rhofree);                            irsync=find((r>=rhofree)&(r<=rhosyn));
irsync2=find((r>rhosyn)&(r<=rhojam));
   irjam=find(r>rhojam);    rsync=r(irsync);
   % V1
   V0=0.85;c=2.9;    V1func=@(rho)V0*tanh((.45./rho-0.02)/(c*V0));    V1=V1func(r);
   %Jamline
   V0s=0.5;cs=2.9;                   Jlinefunc=@(rho)V0s*tanh((.45*(1./rho-1.1))/(cs*V0s));
Jline=Jlinefunc(r);
   Jline(irfree)=V1(irfree);                                        V1(irsync2)=Jline(irsync2);
Jline(irjam)=max(zeros(size(irjam)),Jline(irjam));
   V1(irjam)=max(zeros(size(irjam)),Jline(irjam));
   %Rline
   Rline=zeros(size(r));    slope=(V1func(rhosyn)-Jlinefunc(rhofree))/(rhosyn-rhofree);
   Rline(irsync)=(r(irsync)-
rhofree*ones(size(r(irsync))))*slope+Jlinefunc(rhofree)*ones(size(r(irsync)));
   Rline(irsync2)=ones(size(irsync2));    Rline(irjam)=ones(size(irjam));
   if(Prob.contraction_x<xpos)
      lanefactor=Prob.lanefactor;       r=r/lanefactor;
   end
end
function ieq=uequal(u1,u2,diff)
   ieq=1:length(u1(:,1));
   for iu=1:length(u1(1,:))
      ieq1=find(abs(u1(:,iu)-u2(:,iu))<diff);       ieq=intersect(ieq,ieq1);
   end
end
function ivq=vequal(v1,v2,diff)
   ivq=1:length(v1(:,1));                             ivq1=find(abs(v1(:,1)-v2(:,1))<diff);
ivq=intersect(ivq,ivq1);
end
function [fG,s]=Godunov_flux(Problem,ul,ur,intervall,intervalr)
```

```
   speed=zeros(length(ul),3);%the first wave has two entries for the head and tail of a
rarefaction
                 % in ontcase of a shock use only speed(1)
   ustar=starstate(Problem,ul,ur,intervall,intervalr);    intervalm=intervall;
   rr=ur(:,1);    rl=ul(:,1);    rm=ustar(:,1);    yr=ur(:,2);    yl=ul(:,2);    ym=ustar(:,2);
       vl=yl./rl-press(Problem,rl,intervall);    vr=yr./rr-press(Problem,rr,intervalr);
   vm=ym./rm-press(Problem,rm,intervalm);
   izerol=find(rl<=0);    izeror=find(rr<=0);    vl(izerol)=zeros(length(izerol),1);
   vr(izeror)=zeros(length(izeror),1);
   izerom=find(rm<=0);    rm(izerom)=zeros(length(izerom),1);
   vm(izerom)=vl(izerom)+press(Problem,rl(izerom),izerom);
   speed(:,3)=vr;
   itype1=find((vl-vm-(rl.*pressprime(Problem,rl,intervall)-
rm.*pressprime(Problem,rm,intervalm)))>0);
   itype2=setdiff((1:length(ul)),itype1);

   speed(itype1,1)=(vl(itype1).*rl(itype1)-vm(itype1).*rm(itype1))./(rl(itype1)-
rm(itype1));
   speed(itype1,2)=(vl(itype1).*rl(itype1)-vm(itype1).*rm(itype1))./(rl(itype1)-
rm(itype1));
   speed(itype2,1)=vl(itype2)-rl(itype2).*pressprime(Problem,rl(itype2),itype2);
   speed(itype2,2)=vm(itype2)-rm(itype2).*pressprime(Problem,rm(itype2),itype2);

   %find correct 'Godunov'-states according to the speeds
   il=find(speed(:,1)>0);                                            istar=find(speed(:,2)<=0);
irare=setdiff(setdiff(1:length(ul),il),istar);
   uG(il,:)=ul(il,:);    uG(istar,:)=ustar(istar,:);
   rhs=[0 0];    options=optimset('Display','off');
   for i=irare
      qinit=[rl(i) vl(i)];        rhs(1)=vl(i)+press(Problem,rl(i),i);
      rare=fsolve(@(q)rarefactionshape(Problem,q,rhs,i),qinit,options);
      uG(i,1)=rare(1);        uG(i,2)=rare(1)*rare(2)+rare(1)*press(Problem,rare(1),i);
   end
   [fG,s]=AwRascle_flux(Problem,uG,intervall);
end
function res=rarefactionshape(Problem,q,rhs,interval)
    %riemann invariant
   res(1)=q(2)+press(Problem,q(1),interval)-rhs(1);
   %slope of the characteristic = first eigenvalue
   res(2)=q(2)-q(1)*pressprime(Problem,q(1),interval)-rhs(2);
end
function [f,s]=AwRascle_flux(Problem,u,interval)
   f=zeros(size(u));           r=u(:,1);           y=u(:,2);              p=press(Problem,r,interval);
pp=pressprime(Problem,r,interval);
```

```
   v=y./r-p;    f(:,1)=v.*r;    f(:,2)=v.*y;    s=zeros(size(u));    s(:,1)=v-r.*pp;    s(:,2)=v;
end
function pp=pressNL(Problem,rho)
   C1=Problem.C1;    pp=-C1*log(rho./(ones(size(rho))-rho));
end
function p=press(Problem,rho,interval)
   C1=Problem.C1;                                   lanes=Problem.Lanes(interval);
p=C1*log((rho./lanes)./(ones(size(rho))-rho./lanes));
end
function rho=invpress(Problem,p,interval)
   C1=Problem.C1;                                   lanes=Problem.Lanes(interval);
rho=(lanes.*exp(p/C1))./(ones(size(p))+exp(p/C1));
end
function p=pressprime(Problem,rho,interval)
   C1=Problem.C1;    lanes=Problem.Lanes(interval);
   p=C1*ones(size(rho))./((rho./lanes).*(ones(size(rho))-rho./lanes));
end
function ustar=starstate(Problem,uL,uR,intervalL,intervalR)
   rL=uL(:,1);       yL=uL(:,2);     rR=uR(:,1);       yR=uR(:,2);     val=yL./rL-(yR./rR-
press(Problem,rR,intervalR));
   rstar=invpress(Problem,val,intervalL);          ystar=rstar./rL.*yL;          ustar(:,1)=rstar;
ustar(:,2)=ystar;
end
```

Appendix III: Relaxation Euler

```
function Relaxation_Euler()
clear all;
clc;
[x,u,Usurf,t]=Solve_Euler();    save('Euler.mat','x','u','Usurf','t');
   figure(2)
   B=1.5;      RE=plot(x,u(:,1),'r')      set(RE,  'LineWidth',  B);        xlabel('distance')
ylabel('density')
    title(' 1-Rarefaction, 2-Shock')    axis([-30 30 0 1])
   figure(3)
  RE=plot(x,u(:,2),'k')
   set(RE,  'LineWidth',  B);      xlabel('distance')       ylabel('flow  rate')       title('  1-
Rarefaction, 2-Shock')
    axis([-30 30 0 0.5])
   figure(4)
   RE=plot(x,u(:,2)./u(:,1),'b')          set(RE,   'LineWidth',   B);         xlabel('distance')
ylabel('velocity')
  title(' 1-Rarefaction, 2-Shock')    axis([-30 30 0 1])
   figure(5)
    surf(x,t,Usurf(:,:,1))      xlabel('time(t)')      ylabel('distance(x)')      title('density:  1-
Rarefaction, 2-Shock')
   cmin=0.2;            cmax=0.9;            caxis([cmin    cmax]);            colorbar;
colorbar('location','NorthOutside')    view(2)
   shading flat
end
function [x,u,Usurf,tvector]=Solve_Euler()%time update function
xlower=-30.0;xupper=30.0;
inx=400+1;% number of cells
nbc=2;% number of boundary cells
nx=inx+2*nbc;%total number of cells
dx=abs(xupper-xlower)/(inx-1)
x=xlower-nbc*dx:dx:xupper+nbc*dx;
boundary_type=0;
% boundary_type=2;
Problem={};     Problem.nx=nx;      Problem.nbc=nbc;      Problem.dx=dx;      nu=2;
Problem.nu=nu;
   Problem.C=0.50;    Problem.s0=0.0;
clear u
u=zeros(nx,nu);      clear v        v=zeros(nx,nu);
%information for initial conditions:Riemann data
initalRP={};
     initalRP.xloc=(xlower+xupper)*0.5;
```

```
%       % 1-rarefaction followed by 2-shock
      initalRP.uL=[0.9 0.1];       initalRP.uR=[0.2 0.2];
%       % 1-shock followed by 2-rarefaction
%       initalRP.uL=[0.2 0.9];%       initalRP.uR=[0.9 0.5];
       icont=find(x>0);                                              r=initalRP.uL(1)*ones(nx,1);
r(icont)=initalRP.uR(1)*ones(length(icont),1);
       vel=initalRP.uL(2)*ones(nx,1);
vel(icont)=initalRP.uR(2)*ones(length(icont),1);
       C=Problem.C;
       p=C*C*r;       Q=r.*vel;       v1=r.*vel;       v2=Q.*vel+p;
       u(:,1)=r;                u(:,2)=Q;                v(:,1)=v1;                v(:,2)=v2;        %%
=================================
       istoreU=1; %istore ==to enable storing of computed u after each timestep
       Usurf(istoreU,1:nx,1)=u(1:nx,1);       istoreU=istoreU+1;
     %%
tend=10.0;      tshow=10.4;     t=0.0;       iteration=0;
%%
itvector=1;      tvector(itvector)=t;       itvector=itvector+1;
%%
dt=0.5;       cfl=0.49;
tshow=[tshow inf];        ishow=1;
store_only_special_times=1;
clear s_evolution t_evolution u_evolution
if(store_only_special_times==0)
   s_evolution(1)=0;   t_evolution(1)=t;   u_evolution(:,:,1)=u;   v_evolution(:,:,1)=v;
else
   tstore=tshow;                 tstore=[0     tstore     inf];                ntstore=length(tstore);
s_evolution=zeros(ntstore,1);
   t_evolution=zeros(ntstore,1);                                             t_evolution(1,1)=t;
u_evolution=zeros(nx,nu,ntstore);   u_evolution(:,:,1)=u;
   v_evolution=zeros(nx,nu,ntstore);   v_evolution(:,:,1)=v;       storestep=2;
end
while(t<tend)
   iteration=iteration+1;
   %time control
   dt=min(dt,tend-t);
   %%
    ustore=u;
    %%
   if((t<tshow(ishow))&&(t+dt>tshow(ishow)))
      dt=min(dt,tshow(ishow)-t);       ishow=ishow+1;
   end
   %update
```

```
    [unew,vnew,aC]=RungeKutta_solver(Problem,u,v,nx,dx,dt);
s=max(max(aC.*aC));
    if(dt<=dx/s*cfl)%timestep accepted
        u=unew;        v=vnew;        t=t+dt;
        %%
        tvector(itvector)=t;                                        itvector=itvector+1;
Usurf(istoreU,1:nx,1)=u(1:nx,1);
             istoreU=istoreU+1;
        %%
        dt=dx/s*cfl;
    else%rejected
        dt=dx/s*cfl*0.99;
        %%
        u=ustore;
        %%
    end
    %boundary update
    if(boundary_type==0)%zero Neumann
        u(1:nbc,:)=ones(nbc,1)*u(nbc+1,:);        u(nx-nbc+1:nx,:)=ones(nbc,1)*u(nx-nbc,:);
        v(1:nbc,:)=ones(nbc,1)*v(nbc+1,:);        v(nx-nbc+1:nx,:)=ones(nbc,1)*v(nx-nbc,:);
    elseif(boundary_type==2)%periodic
        u(1:nbc,:)=u(nx-2*nbc+1:nx-nbc,:);        u(nx-nbc+1:nx,:)=u(nbc+1:2*nbc,:);
        v(1:nbc,:)=v(nx-2*nbc+1:nx-nbc,:);        v(nx-nbc+1:nx,:)=v(nbc+1:2*nbc,:);
    end
    if(store_only_special_times==0)
        s_evolution(iteration)=s;                                   t_evolution(iteration)=t;
u_evolution(:,:,iteration)=u;
        v_evolution(:,:,iteration)=v;
    else
        if(tstore(storestep)<=t)
            s_evolution(storestep,1)=s;                         t_evolution(storestep,1)=t;
u_evolution(:,:,storestep)=u;
            v_evolution(:,:,storestep)=v;
            storestep=storestep+1;
        end
    end
   C=Problem.C;
   p(nbc+1:nx-nbc,1)=C*C*u(nbc+1:nx-nbc,1);
   vel(nbc+1:nx-nbc,1)=u(nbc+1:nx-nbc,2)./u(nbc+1:nx-nbc,1);
       %for iu=1:nu
           subplot(3,1,1)
           plot(x(1,nbc+1:nx-nbc),u(nbc+1:nx-nbc,1),'r')
           ylabel('density')
           subplot(3,1,2)
```

```
          plot(x(1,nbc+1:nx-nbc),u(nbc+1:nx-nbc,2),'b')
          ylabel('flow')
          subplot(3,1,3)
          plot(x(1,nbc+1:nx-nbc),vel(nbc+1:nx-nbc,1),'k')
          ylabel('velocity')
      title(t)
       % end
       pause(0.005);
end
end

function[unew,vnew,aC]=RungeKutta_solver(Problem,u,v,nx,dx,dt)
    e=1*10^(-8);%epsilon of the relaxation scheme
 %RUNGE                              KUTTA                              STEP
1============================================
   ustar=u;
   [fstar,sstar,velstar]=Euler_flux(Problem,ustar);
   F=fstar;
   vstar=(e/(e-dt))*(v-(dt/e)*F);
   ustarLL=u(1:nx-4,:);ustarL=u(2:nx-3,:);ustarC=u(3:nx-2,:);ustarR=u(4:nx-
1,:);ustarRR=u(5:nx,:);
   vstarLL=vstar(1:nx-4,:);vstarL=vstar(2:nx-3,:);vstarC=vstar(3:nx-
2,:);vstarR=vstar(4:nx-1,:);vstarRR=vstar(5:nx,:);
    %sigma_j-1

[phipLS,phimLS,thetapLS,thetamLS]=Sweby_notation(Problem,ustarLL,ustarL,ustarC,
vstarLL,vstarL,vstarC);

[sigmapLS,sigmamLS,aL]=Sweby_notation2(Problem,ustarL,ustarC,vstarL,vstarC,phip
LS,phimLS,dx);
   %sigma_j

[phipCS,phimCS,thetapCS,thetamCS]=Sweby_notation(Problem,ustarL,ustarC,ustarR,v
starL,vstarC,vstarR);

[sigmapCS,sigmamCS,aC]=Sweby_notation2(Problem,ustarC,ustarR,vstarC,vstarR,phip
CS,phimCS,dx);
   %sigma_j+1

[phipRS,phimRS,thetapRS,thetamRS]=Sweby_notation(Problem,ustarC,ustarR,ustarRR,
vstarC,vstarR,vstarRR);

[sigmapRS,sigmamRS,aR]=Sweby_notation2(Problem,ustarR,ustarRR,vstarR,vstarRR,p
hipRS,phimRS,dx);
```

```
[DplusUS,DplusVS]=Dplus_operator(Problem,ustarL,ustarC,ustarR,vstarL,vstarC,vstar
R,sigmapLS,sigmapCS,sigmamCS,sigmamRS,dx,aC);
   ADpUstar=zeros(size(u));                ADpUstar(3:nx-2,:)=(aC.*aC).*DplusUS;
DpVstar=zeros(size(u));
   DpVstar(3:nx-2,:)=DplusVS;
   ustep1=ustar-dt*DpVstar;    vstep1=vstar-dt*ADpUstar;
  %=============================================
   uSS=ustep1;
   [fSS,sSS,velSS]=Euler_flux(Problem,uSS);    size(velSS);    FSS=fSS;
   vSS=(e/(e+dt))*(vstep1+(dt/e)*FSS-(2*dt/e)*(vstar-F));
   %RUNGE                              KUTTA                              STEP
2%=============================================
   ustar2LL=uSS(1:nx-4,:);ustar2L=uSS(2:nx-3,:);ustar2C=uSS(3:nx-
2,:);ustar2R=uSS(4:nx-1,:);ustar2RR=uSS(5:nx,:);
   vstar2LL=vSS(1:nx-4,:);vstar2L=vSS(2:nx-3,:);vstar2C=vSS(3:nx-
2,:);vstar2R=vSS(4:nx-1,:);vstar2RR=vSS(5:nx,:);
   %sigma_j-1

[phipLSS,phimLSS,thetapLSS,thetamLSS]=Sweby_notation(Problem,ustar2LL,ustar2L,
ustar2C,vstar2LL,vstar2L,vstar2C);

[sigmapLSS,sigmamLSS,aLS]=Sweby_notation2(Problem,ustar2L,ustar2C,vstar2L,vsta
r2C,phipLSS,phimLSS,dx);
   %sigma_j
[phipCSS,phimCSS,thetapCSS,thetamCSS]=Sweby_notation(Problem,ustar2L,ustar2C,
ustar2R,vstar2L,vstar2C,vstar2R);
[sigmapCSS,sigmamCSS,aCS]=Sweby_notation2(Problem,ustar2C,ustar2R,vstar2C,vst
ar2R,phipCSS,phimCSS,dx);
   %sigma_j+1
[phipRSS,phimRSS,thetapRSS,thetamRSS]=Sweby_notation(Problem,ustar2C,ustar2R,
ustar2RR,vstar2C,vstar2R,vstar2RR);
[sigmapRSS,sigmamRSS,aRS]=Sweby_notation2(Problem,ustar2R,ustar2RR,vstar2R,vs
tar2RR,phipRSS,phimRSS,dx);
[DplusUSS,DplusVSS]=Dplus_operator(Problem,ustar2L,ustar2C,ustar2R,vstar2L,vstar
2C,vstar2R,sigmapLSS,sigmapCSS,sigmamCSS,sigmamRSS,dx,aCS);
   ADpUstasta=zeros(size(u));
   ADpUstasta(3:nx-2,:)=(aCS.*aCS).*DplusUSS;
   DpVstasta=zeros(size(u));
   DpVstasta(3:nx-2,:)=DplusVSS;
   ustep2=uSS-dt*DpVstasta;
   vstep2=vSS-dt*ADpUstasta;

%===================================================
==========
   unew=0.5*(u+ustep2);
   vnew=0.5*(v+vstep2);
end
function
[DplusU,DplusV]=Dplus_operator(Problem,UL,UC,UR,VL,VC,VR,SIGMApL,SIGMA
pC,SIGMAmC,SIGMAmR,dx,aC)
 %computation of U_i+half
   U_sum=UC+UR;         V_diff=VR-VC;    SIG_sum=SIGMApC+SIGMAmR;
   U_phalf=0.5*U_sum-(0.5./(eps+aC)).*V_diff+ dx*0.25*(SIG_sum./(eps+aC));
%computation of U_i-half
   U_sum2=UL+UC;       V_diff2=VC-VL;        SIG_sum2=SIGMApL+SIGMAmC;
   U_mhalf=0.5*U_sum2-(0.5./(eps+aC)).*V_diff2+ dx*0.25*(SIG_sum2./(eps+aC));
   DplusU=(U_phalf-U_mhalf)/dx;
   %computation of V_i+half
   V_sum=VC+VR;
   U_diff=UR-UC;
   SIG_diff=SIGMApC-SIGMAmR;
   V_phalf=0.5*V_sum-(0.5*aC).*U_diff+ dx*0.25*SIG_diff;
   %computation of V_i-half
   V_sum2=VL+VC;
   U_diff2=UC-UL;
   SIG_diff2=SIGMApL-SIGMAmC;
   V_mhalf=0.5*V_sum2-(0.5*aC).*U_diff2+ dx*0.25*SIG_diff2;
   DplusV=(V_phalf-V_mhalf)/dx;
end
function [phip,phim,thetap,thetam]=Sweby_notation(Problem,ul,uc,ur,vl,vc,vr)
s0=Problem.s0;
%========= computation of a_p in the relax scheme
[fl,sl,vell]=Euler_flux(Problem,ul);
asl(1)=s0+max(abs(sl(:,1)));                              asl(2)=s0+max(abs(sl(:,2)));
al(:,1)=sqrt(asl(1)).*ones(size(ul(:,1)));
al(:,2)=sqrt(asl(2)).*ones(size(ul(:,2)));
 [fc,sc,velc]=Euler_flux(Problem,uc);                     asc(1)=s0+max(abs(sc(:,1)));
         asc(2)=s0+max(abs(sc(:,2)));
ac(:,1)=sqrt(asc(1)).*ones(size(uc(:,1)));       ac(:,2)=sqrt(asc(2)).*ones(size(uc(:,2)));
 [fr,sr,velr]=Euler_flux(Problem,ur);  asr(1)=s0+max(abs(sr(:,1)));
         asr(2)=s0+max(abs(sr(:,2)));
ar(:,1)=sqrt(asr(1)).*ones(size(ur(:,1)));        ar(:,2)=sqrt(asr(2)).*ones(size(ur(:,2)));
%========= computation of a_p in the relax scheme
tetap1=(vc+ac.*uc-vl-al.*ul); tetap2=(vr+ar.*ur-vc-ac.*uc);thetap=tetap1./(tetap2);
tetam1=(vc-ac.*uc-vl+al.*ul);           tetam2=(vr-ar.*ur-vc+ac.*uc);
         thetam=tetam1./(tetam2);
p1=0;p2=5;
ipbelow=find(thetap(:,1)<=p1);
```

```
ipmiddle=find((thetap(:,1)>p1)&(thetap(:,1)<=p2));
ipabove=find(thetap(:,1)>p2);
ipbelow2=find(thetap(:,2)<=p1);
ipmiddle2=find((thetap(:,2)>p1)&(thetap(:,2)<=p2));
ipabove2=find(thetap(:,2)>p2);

phip=zeros(size(uc));
phip(ipbelow,1)=zeros(size(thetap(ipbelow,1)));
phip(ipmiddle,1)=slope_limiter(Problem,thetap(ipmiddle,1));
phip(ipabove,1)=2*ones(size(thetap(ipabove,1)));

phip(ipbelow2,2)=zeros(size(thetap(ipbelow2,2)));
phip(ipmiddle2,2)=slope_limiter(Problem,thetap(ipmiddle2,2));
phip(ipabove2,2)=2*ones(size(thetap(ipabove2,2)));

imbelow=find(thetam(:,1)<=p1);
immiddle=find((thetam(:,1)>p1)&(thetam(:,1)<=p2));
imabove=find(thetam(:,1)>p2);
imbelow2=find(thetam(:,2)<=p1);
immiddle2=find((thetam(:,2)>p1)&(thetam(:,2)<=p2));
imabove2=find(thetam(:,2)>p2);

phim=zeros(size(uc));
phim(imbelow,1)=zeros(size(thetam(imbelow,1)));
phim(immiddle,1)=slope_limiter(Problem,thetam(immiddle,1));
phim(imabove,1)=2*ones(size(thetam(imabove,1)));

phim(imbelow2,2)=zeros(size(thetam(imbelow2,2)));
phim(immiddle2,2)=slope_limiter(Problem,thetam(immiddle2,2));
phim(imabove2,2)=2*ones(size(thetam(imabove2,2)));
size(phim);
end
%computes sigma+, sigma- and constants
function [sigmap,sigmam,ac]=Sweby_notation2(Problem,uc,ur,vc,vr,fip,fim,dx)
s0=Problem.s0;
%========= computation of a_p in the relax scheme
[fc,sc,velc]=Euler_flux(Problem,uc);
asc(1)=s0+max(abs(sc(:,1)));          asc(2)=s0+max(abs(sc(:,2)));
        ac(:,1)=sqrt(asc(1)).*ones(size(uc(:,1)));
ac(:,2)=sqrt(asc(2)).*ones(size(uc(:,2)));
[fr,sr,velr]=Euler_flux(Problem,ur);
asr(1)=s0+max(abs(sr(:,1)));  asr(2)=s0+max(abs(sr(:,2)));
        ar(:,1)=sqrt(asr(1)).*ones(size(ur(:,1)));
ar(:,2)=sqrt(asr(2)).*ones(size(ur(:,2)));        %=========computation of a_p in the
relax scheme
sigmap=(1/dx)*(vr+ar.*ur-vc-ac.*uc).*fip;   sigmam=(1/dx)*((vr-ar.*ur)-
vc+ac.*uc).*fim;
end
function phi=slope_limiter(Problem,teta) %slope-limiter fxn
phi=(abs(teta)+teta)./(ones(size(teta))+abs(teta));
end
function [f,s,vel]=Euler_flux(Problem,u) %gives the flux and speed
   C=Problem.C;   f=zeros(size(u));   r=u(:,1);   Q=u(:,2);   p=C*C*r;   %C=a of
Euler Eqns
   vel=Q./r;   f(:,1)=Q;   f(:,2)=Q.*vel+p;   s=zeros(size(u));
   s(:,1)=vel-C*ones(size(u(:,1)));  %speed (the eigen values)
   s(:,2)=vel+C*ones(size(u(:,1)));
end
```

Appendix IV: Relaxation Traffic

```
function TrafficModel2()
clear all;
clc;
[x,p,u,Usurf,t]=Solve_Euler();
   save('Euler.mat','x','u','Usurf','t','p');
   figure(2)
   B=1.5;      RE=plot(x,u(:,1),'r')      set(RE,  'LineWidth',  B);        xlabel('distance')
ylabel('density')
%    title('1-shock, 2-contact')
    title('1-Rarefaction, 2-contact')    axis([-30 30 0 1])
   figure(3)
  RE=plot(x,u(:,2),'k')    set(RE, 'LineWidth', B);    xlabel('distance')    ylabel('flow rate')
%    title('1-shock, 2-contact')
    title('1-Rarefaction, 2-contact')    axis([-30 30 0 1])
   figure(4)
   RE=plot(x,u(:,2)./u(:,1)-p,'b')         set(RE,   'LineWidth',   B);        xlabel('distance')
ylabel('velocity')
%  title('1-shock, 2-contact')
  title('1-Rarefaction, 2-contact')   axis([-30 30 0 1.2])
   figure(5)
    surf(x,t,Usurf(:,:,1))   ylabel('time(t)')   xlabel('distance(x)')
%   title('density: 1-shock, 2-contact')
   title('density:  1-Rarefaction,  2-contact')     cmin=0.2;     cmax=0.9;     caxis([cmin
cmax]);
   colorbar;      colorbar('location','SouthOutside')      axis([-30  30  0  10])      view(2)
shading flat
end
function [x,p,u,Usurf,tvector]=Solve_Euler()%time update function
xlower=-30;   xupper=30;
inx=400+1;% number of cells
nbc=2;% number of boundary cells
nx=inx+2*nbc;%total number of cells
dx=abs(xupper-xlower)/(inx-1);       x=xlower-nbc*dx:dx:xupper+nbc*dx;
          boundary_type=0;
% boundary_type=2;
Problem={};  Problem.nx=nx;       Problem.nbc=nbc;     Problem.dx=dx;       nu=2;
   Problem.nu=nu;   Problem.C=0.9;   Problem.s0=0.0;
clear u
u=zeros(nx,nu);
clear v
v=zeros(nx,nu);
%information for initial conditions:Riemann data
```

```
initalRP={};
       initalRP.xloc=(xlower+xupper)*0.5;
       % 1-rarefaction followed by 2-contact
        initalRP.uL=[0.6 0.1];       initalRP.uR=[0.6 0.95];
%        % 1-shock followed by 2-contact
%       initalRP.uL=[0.6 0.95];
%       initalRP.uR=[0.6 0.1];
        icont=find(x>0);
        r=initalRP.uL(1)*ones(nx,1);       r(icont)=initalRP.uR(1)*ones(length(icont),1);
        vel=initalRP.uL(2)*ones(nx,1);
vel(icont)=initalRP.uR(2)*ones(length(icont),1);
        C=Problem.C;       p=C*log(r./(1-r));
        Q=r.*(vel+p);       v1=r.*vel;       v2=Q.*vel;
        u(:,1)=r;       u(:,2)=Q;       v(:,1)=v1;       v(:,2)=v2;
    %% ==================================
        istoreU=1; %istore ==to enable storing of computed u after each timestep
        Usurf(istoreU,1:nx,1)=u(1:nx,1);       istoreU=istoreU+1;
tend=10;                  tshow=10.4;    t=0.0;  iteration=0;     %%
itvector=1;      tvector(itvector)=t;     itvector=itvector+1;    %%
dt=.5;              cfl=0.49;         tshow=[tshow inf];     ishow=1;
store_only_special_times=1;
clear s_evolution t_evolution u_evolution
if(store_only_special_times==0)
    s_evolution(1)=0;    t_evolution(1)=t;    u_evolution(:,:,1)=u;    v_evolution(:,:,1)=v;
else
    tstore=tshow;                  tstore=[0     tstore     inf];                  ntstore=length(tstore);
s_evolution=zeros(ntstore,1);
    t_evolution=zeros(ntstore,1);                                                  t_evolution(1,1)=t;
u_evolution=zeros(nx,nu,ntstore);    u_evolution(:,:,1)=u;
    v_evolution=zeros(nx,nu,ntstore);    v_evolution(:,:,1)=v;
    storestep=2;
end
while(t<tend)
    iteration=iteration+1;
   %time control
    dt=dt;%=min(dt,tend-t);
    %%
     ustore=u;
     %%
    if((t<tshow(ishow))&&(t+dt>tshow(ishow)))
        dt=dt;%=min(dt,tshow(ishow)-t);       ishow=ishow+1;
    end
    %update
    [unew,vnew,aC]=RungeKutta_solver(Problem,u,v,nx,dx,dt);
s=max(max(aC.*aC))
    if(dt<=dx/s*cfl)%timestep accepted
        u=unew;        v=vnew;        t=t+dt;        %%
        tvector(itvector)=t;                                                    itvector=itvector+1;
Usurf(istoreU,1:nx,1)=u(1:nx,1);
            istoreU=istoreU+1;        %%
        dt=dx/s*cfl;
    else%rejected
        dt=dx/s*cfl*0.99;
        %%
        u=ustore;
        %%
    end
   dt
%   boundary update
    if(boundary_type==0)%zero Neumann
        u(1:nbc,:)=ones(nbc,1)*u(nbc+1,:);                u(nx-nbc+1:nx,:)=ones(nbc,1)*u(nx-
nbc,:);
        v(1:nbc,:)=ones(nbc,1)*v(nbc+1,:);        v(nx-nbc+1:nx,:)=ones(nbc,1)*v(nx-nbc,:);
    elseif(boundary_type==2)%periodic
        u(1:nbc,:)=u(nx-2*nbc+1:nx-nbc,:);        u(nx-nbc+1:nx,:)=u(nbc+1:2*nbc,:);
        v(1:nbc,:)=v(nx-2*nbc+1:nx-nbc,:);        v(nx-nbc+1:nx,:)=v(nbc+1:2*nbc,:);
    end
    if(store_only_special_times==0)
        s_evolution(iteration)=s;        t_evolution(iteration)=t;
        u_evolution(:,:,iteration)=u;        v_evolution(:,:,iteration)=v;

    else
        if(tstore(storestep)<=t)
            s_evolution(storestep,1)=s;            t_evolution(storestep,1)=t;
            u_evolution(:,:,storestep)=u;            v_evolution(:,:,storestep)=v;
           storestep=storestep+1;
        end
    end
   C=Problem.C;
   p(nbc+1:nx-nbc,1)=C*log(u(nbc+1:nx-nbc,1)./(ones(size(u(nbc+1:nx-nbc,1)))-
u(nbc+1:nx-nbc,1)));
   vel(nbc+1:nx-nbc,1)=(u(nbc+1:nx-nbc,2)./u(nbc+1:nx-nbc,1))-p(nbc+1:nx-nbc,1);
       %for iu=1:nu
           subplot(3,1,1)
           plot(x(1,nbc+1:nx-nbc),u(nbc+1:nx-nbc,1),'r')
           ylabel('density')
           subplot(3,1,2)
```

```
          plot(x(1,nbc+1:nx-nbc),u(nbc+1:nx-nbc,2),'b')
          ylabel('flow')
          subplot(3,1,3)
          plot(x(1,nbc+1:nx-nbc),vel(nbc+1:nx-nbc,1),'k')
          ylabel('velocity')
      title(t)
       % end
       pause(0.005);
end
end

function[unew,vnew,aC]=RungeKutta_solver(Problem,u,v,nx,dx,dt)
    e=1*10^(-8);%epsilon of the relaxation scheme
 %RUNGE                                KUTTA                                STEP
1==========================================
   ustar=u;
   [fstar,sstar,velstar]=Traffic_model(Problem,ustar);      F=fstar;      vstar=(e/(e-dt))*(v-
(dt/e)*F);
   ustarLL=u(1:nx-4,:);ustarL=u(2:nx-3,:);ustarC=u(3:nx-2,:);ustarR=u(4:nx-
1,:);ustarRR=u(5:nx,:);
   vstarLL=vstar(1:nx-4,:);vstarL=vstar(2:nx-3,:);vstarC=vstar(3:nx-
2,:);vstarR=vstar(4:nx-1,:);vstarRR=vstar(5:nx,:);
    %sigma_j-1

[phipLS,phimLS,thetapLS,thetamLS]=Sweby_notation(Problem,ustarLL,ustarL,ustarC,
vstarLL,vstarL,vstarC);

[sigmapLS,sigmamLS,aL]=Sweby_notation2(Problem,ustarL,ustarC,vstarL,vstarC,phip
LS,phimLS,dx);
   %sigma_j

[phipCS,phimCS,thetapCS,thetamCS]=Sweby_notation(Problem,ustarL,ustarC,ustarR,v
starL,vstarC,vstarR);

[sigmapCS,sigmamCS,aC]=Sweby_notation2(Problem,ustarC,ustarR,vstarC,vstarR,phip
CS,phimCS,dx);

   %sigma_j+1

[phipRS,phimRS,thetapRS,thetamRS]=Sweby_notation(Problem,ustarC,ustarR,ustarRR,
vstarC,vstarR,vstarRR);

[sigmapRS,sigmamRS,aR]=Sweby_notation2(Problem,ustarR,ustarRR,vstarR,vstarRR,p
hipRS,phimRS,dx);
```

```
[DplusUS,DplusVS]=Dplus_operator(Problem,ustarL,ustarC,ustarR,vstarL,vstarC,vstar
R,sigmapLS,sigmapCS,sigmamCS,sigmamRS,dx,aC);
   ADpUstar=zeros(size(u));                    ADpUstar(3:nx-2,:)=(aC.*aC).*DplusUS;
DpVstar=zeros(size(u));
   DpVstar(3:nx-2,:)=DplusVS;          ustep1=ustar-dt*DpVstar;          vstep1=vstar-
dt*ADpUstar;
  %=================================================
   uSS=ustep1;
   [fSS,sSS,velSS]=Traffic_model(Problem,uSS);
   size(velSS);
   FSS=fSS;
   vSS=(e/(e+dt))*(vstep1+(dt/e)*FSS-(2*dt/e)*(vstar-F));
   %RUNGE                                    KUTTA                                    STEP
2%=================================================
   ustar2LL=uSS(1:nx-4,:);ustar2L=uSS(2:nx-3,:);ustar2C=uSS(3:nx-
2,:);ustar2R=uSS(4:nx-1,:);ustar2RR=uSS(5:nx,:);
   vstar2LL=vSS(1:nx-4,:);vstar2L=vSS(2:nx-3,:);vstar2C=vSS(3:nx-
2,:);vstar2R=vSS(4:nx-1,:);vstar2RR=vSS(5:nx,:);
   %sigma_j-1
[phipLSS,phimLSS,thetapLSS,thetamLSS]=Sweby_notation(Problem,ustar2LL,ustar2L,
ustar2C,vstar2LL,vstar2L,vstar2C);

[sigmapLSS,sigmamLSS,aLS]=Sweby_notation2(Problem,ustar2L,ustar2C,vstar2L,vsta
r2C,phipLSS,phimLSS,dx);
   %sigma_j
[phipCSS,phimCSS,thetapCSS,thetamCSS]=Sweby_notation(Problem,ustar2L,ustar2C,
ustar2R,vstar2L,vstar2C,vstar2R);
[sigmapCSS,sigmamCSS,aCS]=Sweby_notation2(Problem,ustar2C,ustar2R,vstar2C,vst
ar2R,phipCSS,phimCSS,dx);

   %sigma_j+1

[phipRSS,phimRSS,thetapRSS,thetamRSS]=Sweby_notation(Problem,ustar2C,ustar2R,
ustar2RR,vstar2C,vstar2R,vstar2RR);

[sigmapRSS,sigmamRSS,aRS]=Sweby_notation2(Problem,ustar2R,ustar2RR,vstar2R,vs
tar2RR,phipRSS,phimRSS,dx);

[DplusUSS,DplusVSS]=Dplus_operator(Problem,ustar2L,ustar2C,ustar2R,vstar2L,vstar
2C,vstar2R,sigmapLSS,sigmapCSS,sigmamCSS,sigmamRSS,dx,aCS);

   ADpUstasta=zeros(size(u));
   ADpUstasta(3:nx-2,:)=(aCS.*aCS).*DplusUSS;
   DpVstasta=zeros(size(u));
   DpVstasta(3:nx-2,:)=DplusVSS;

   ustep2=uSS-dt*DpVstasta;
   vstep2=vSS-dt*ADpUstasta;

%=============================================================
==========

   unew=0.5*(u+ustep2);
   vnew=0.5*(v+vstep2);

end
function
[DplusU,DplusV]=Dplus_operator(Problem,UL,UC,UR,VL,VC,VR,SIGMApL,SIGMA
pC,SIGMAmC,SIGMAmR,dx,aC)
 %computation of U_i+half
   U_sum=UC+UR;
   V_diff=VR-VC;
   SIG_sum=SIGMApC+SIGMAmR;
   U_phalf=0.5*U_sum-(0.5./(eps+aC)).*V_diff+ dx*0.25*(SIG_sum./(eps+aC));

 %computation of U_i-half
   U_sum2=UL+UC;
   V_diff2=VC-VL;
   SIG_sum2=SIGMApL+SIGMAmC;
   U_mhalf=0.5*U_sum2-(0.5./(eps+aC)).*V_diff2+ dx*0.25*(SIG_sum2./(eps+aC));

   DplusU=(U_phalf-U_mhalf)/dx;

   %computation of V_i+half
   V_sum=VC+VR;
   U_diff=UR-UC;
   SIG_diff=SIGMApC-SIGMAmR;
   V_phalf=0.5*V_sum-(0.5*aC).*U_diff+ dx*0.25*SIG_diff;

   %computation of V_i-half
   V_sum2=VL+VC;
   U_diff2=UC-UL;
   SIG_diff2=SIGMApL-SIGMAmC;
   V_mhalf=0.5*V_sum2-(0.5*aC).*U_diff2+ dx*0.25*SIG_diff2;

   DplusV=(V_phalf-V_mhalf)/dx;

end
```

```
function [phip,phim,thetap,thetam]=Sweby_notation(Problem,ul,uc,ur,vl,vc,vr)

s0=Problem.s0;
%========= computation of a_p in the relax scheme
[fl,sl,vell]=Traffic_model(Problem,ul);
asl(1)=s0+max(abs(sl(:,1)));
asl(2)=s0+max(abs(sl(:,2)));
al(:,1)=sqrt(asl(1)).*ones(size(ul(:,1)));
al(:,2)=sqrt(asl(2)).*ones(size(ul(:,2)));

[fc,sc,velc]=Traffic_model(Problem,uc);
asc(1)=s0+max(abs(sc(:,1)));
asc(2)=s0+max(abs(sc(:,2)));
ac(:,1)=sqrt(asc(1)).*ones(size(uc(:,1)));
ac(:,2)=sqrt(asc(2)).*ones(size(uc(:,2)));

[fr,sr,velr]=Traffic_model(Problem,ur);
asr(1)=s0+max(abs(sr(:,1)));
asr(2)=s0+max(abs(sr(:,2)));
ar(:,1)=sqrt(asr(1)).*ones(size(ur(:,1)));
ar(:,2)=sqrt(asr(2)).*ones(size(ur(:,2)));
%========= computation of a_p in the relax scheme

tetap1=(vc+ac.*uc-vl-al.*ul);
tetap2=(vr+ar.*ur-vc-ac.*uc);
thetap=tetap1./(tetap2);

tetam1=(vc-ac.*uc-vl+al.*ul);
tetam2=(vr-ar.*ur-vc+ac.*uc);
thetam=tetam1./(tetam2);

p1=0;p2=5;

ipbelow=find(thetap(:,1)<=p1);
ipmiddle=find((thetap(:,1)>p1)&(thetap(:,1)<=p2));
ipabove=find(thetap(:,1)>p2);
ipbelow2=find(thetap(:,2)<=p1);
ipmiddle2=find((thetap(:,2)>p1)&(thetap(:,2)<=p2));
ipabove2=find(thetap(:,2)>p2);

phip=zeros(size(uc));
phip(ipbelow,1)=zeros(size(thetap(ipbelow,1)));
phip(ipmiddle,1)=slope_limiter(Problem,thetap(ipmiddle,1));
phip(ipabove,1)=2*ones(size(thetap(ipabove,1)));

phip(ipbelow2,2)=zeros(size(thetap(ipbelow2,2)));
phip(ipmiddle2,2)=slope_limiter(Problem,thetap(ipmiddle2,2));
phip(ipabove2,2)=2*ones(size(thetap(ipabove2,2)));

imbelow=find(thetam(:,1)<=p1);
immiddle=find((thetam(:,1)>p1)&(thetam(:,1)<=p2));
imabove=find(thetam(:,1)>p2);
imbelow2=find(thetam(:,2)<=p1);
immiddle2=find((thetam(:,2)>p1)&(thetam(:,2)<=p2));
imabove2=find(thetam(:,2)>p2);

phim=zeros(size(uc));
phim(imbelow,1)=zeros(size(thetam(imbelow,1)));
phim(immiddle,1)=slope_limiter(Problem,thetam(immiddle,1));
phim(imabove,1)=2*ones(size(thetam(imabove,1)));

phim(imbelow2,2)=zeros(size(thetam(imbelow2,2)));
phim(immiddle2,2)=slope_limiter(Problem,thetam(immiddle2,2));
phim(imabove2,2)=2*ones(size(thetam(imabove2,2)));
size(phim);
end

%computes sigma+, sigma- and constants
function [sigmap,sigmam,ac]=Sweby_notation2(Problem,uc,ur,vc,vr,fip,fim,dx)
s0=Problem.s0;
%========= computation of a_p in the relax scheme
[fc,sc,velc]=Traffic_model(Problem,uc);
asc(1)=s0+max(abs(sc(:,1)));          asc(2)=s0+max(abs(sc(:,2)));
ac(:,1)=sqrt(asc(1)).*ones(size(uc(:,1)));     ac(:,2)=sqrt(asc(2)).*ones(size(uc(:,2)));

[fr,sr,velr]=Traffic_model(Problem,ur);      asr(1)=s0+max(abs(sr(:,1)));
asr(2)=s0+max(abs(sr(:,2)));          ar(:,1)=sqrt(asr(1)).*ones(size(ur(:,1)));
ar(:,2)=sqrt(asr(2)).*ones(size(ur(:,2)));
%=========computation of a_p in the relax scheme

sigmap=(1/dx)*(vr+ar.*ur-vc-ac.*uc).*fip;          sigmam=(1/dx)*((vr-ar.*ur)-
vc+ac.*uc).*fim;
end

function phi=slope_limiter(Problem,teta) %slope-limiter fxn
phi=(abs(teta)+teta)./(ones(size(teta))+abs(teta));
end
```

```
function [f,s,vel]=Traffic_model(Problem,u) %gives the flux and speed
   C=Problem.C;
   f=zeros(size(u));    r=u(:,1);    Q=u(:,2);    pp=C./(1-r);    p=C*log(r./(1-r));
   vel=(Q./r)-p;    f(:,1)=r.*vel;    f(:,2)=Q.*vel;    s=zeros(size(u));
   s(:,1)=vel-r.*pp;  %speed (the eigen values)
   s(:,2)=vel;
end
```

Appendix V: Comparison Code

```
load('Euler.mat','x','u','Usurf','t','p');
xs=x*1;
load ('KIN.mat','xKIN','tKIN','uKIN','pp','UsurfKIN','tend','KIN');
figure(1)
B=1.5;
hold on
RE= plot(xs,u(:,1),'xr');
set(RE, 'LineWidth', B);
RE= plot(xKIN,uKIN(:,1)','b');
set(RE, 'LineWidth', B);
legend('RELAXATION SCHEME','GODUNOV SCHEME');
hold off
    xlabel('distance')
    ylabel('density')
    title('1-Rarefaction, 2-Shock')
    axis([-30 30 0.0 1.0])

 figure(2)
 D=1.5;
 hold on
%  C=0.5;
%  p=u(:,2)*C*C;
 RE= plot(xs,u(:,2)./u(:,1),'xr');
 set(RE, 'LineWidth', B);
C=.9;
 pp=C*log((uKIN./(ones(size(uKIN))-uKIN)));
  RE= plot(xKIN,uKIN(:,2)./uKIN(:,1),'b');
  legend('RELAXATION SCHEME','GODUNOV SCHEME');
   hold off
  set(RE, 'LineWidth', B);

   xlabel('distance')
   ylabel('velocity')
   title('1-Rarefaction, 2-Shock')
   axis([-30 30 0.0 1.2])
```

Appendix VI: Publication Acceptance