Master Thesis

Support for Dymola in the Modeling and Simulation of Physical Systems with Distributed Parameters

ETH Zürich
Department of Computer Science
Institute of Computational Science

Farid Dshabarow

Adviser: Prof. François E. Cellier
Responsible: Prof. Walter Gander

June 18, 2007

Contents

1 Preface

The thesis title "Support for Dymola in the Modeling and Simulation of Physical Systems with Distributed Parameters" could appear a little bit confusing. In short, the goal of the master thesis is to provide Dymola with a Partial Differential Equations (PDE) Package, with which it will be possible to simulate physical systems for example. The PDE area is huge and it is therefore not possible to implement everything. For instance, the methods of solving PDEs are many, ranging from analytical to semianalytical to fully numerical. The methods implemented in PDE Package are Method of Lines (MOL) and Finite Volume Methods (FVM). Among many types of PDEs we considered only time-dependent PDEs, because Dymola was conceived mainly to simulate the quantities in time. Recently, however, a time independent problem, the Poisson equation, was also implemented and works well. The PDE Package provides necessary blocks for the implementation of PDEs, such as integrators and space derivatives blocks. Many examples were implemented with both MOL and FVM to show the use of PDE Package. Some examples are implemented together with the corresponding analytical solution so that the user can see the error of the approximation. To make everything transparent to the user the methods are implemented in such a way that user must not understand the details of the numerical methods that solve the PDEs. What is required from the user is that he knows the complete problem (PDE, initial condition, ...) that he wants to implement. The blocks necessary for the implementation of the problem are provided, and explanations of how to use them is described in the corresponding documentation. As already said, many PDEs, ranging from simple, like advection equation, to more complex, like Euler equations, are implemented to show the user the correct use of the package.

2 Partial Differential Equations

Partial differential equations are the main topic of this thesis. In the following a very short introduction is given with the purpose to show some important aspects of these huge field. In the next Chapters two numerical methods, the Method of Lines and the Finite Volume Methods for solving these equations will be discussed more deeply.

2.1 Introduction

Many phenomena in nature are described by PDEs. Some examples are [4]:

- **Navier-Stokes equations**

  The Navier-Stokes equations describe the fluid behavior by prescribing the relationships among its velocity, density, pressure and viscosity.

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho V) = 0 \tag{1}$$

$$\frac{\partial \rho v_x}{\partial t} + \nabla \cdot (\rho v_x V) = -\frac{\partial p}{\partial x} + \frac{\partial \tau_{xx}}{\partial x} + \frac{\partial \tau_{yx}}{\partial y} + \frac{\partial \tau_{zx}}{\partial z} + \rho f_x \tag{2}$$

$$\frac{\partial \rho v_y}{\partial t} + \nabla \cdot (\rho v_y V) = -\frac{\partial p}{\partial y} + \frac{\partial \tau_{xy}}{\partial x} + \frac{\partial \tau_{yy}}{\partial y} + \frac{\partial \tau_{zy}}{\partial z} + \rho f_y \tag{3}$$

$$\frac{\partial \rho v_z}{\partial t} + \nabla \cdot (\rho v_z V) = -\frac{\partial p}{\partial z} + \frac{\partial \tau_{xz}}{\partial x} + \frac{\partial \tau_{yz}}{\partial y} + \frac{\partial \tau_{zz}}{\partial z} + \rho f_z \tag{4}$$

$$\frac{\partial}{\partial t}(\rho(e + \frac{V^2}{2})) + \nabla \cdot (\rho(e + \frac{V^2}{2})V) = \tag{5}$$

$$\rho \dot{q} + \frac{\partial}{\partial x}(k\frac{\partial T}{\partial x}) + \frac{\partial}{\partial y}(k\frac{\partial T}{\partial y}) + \frac{\partial}{\partial z}(k\frac{\partial T}{\partial z}) -$$

$$\frac{\partial v_x p}{\partial x} - \frac{\partial v_y p}{\partial y} - \frac{\partial v_z p}{\partial z} + \frac{\partial v_x \tau_{xx}}{\partial x} + \frac{\partial v_x \tau_{yx}}{\partial y} + \frac{\partial v_x \tau_{zx}}{\partial z} +$$

$$\frac{\partial v_y \tau_{xy}}{\partial x} + \frac{\partial v_y \tau_{yy}}{\partial y} + \frac{\partial v_y \tau_{zy}}{\partial z} +$$

$$\frac{\partial v_z \tau_{xz}}{\partial x} + \frac{\partial v_z \tau_{yz}}{\partial y} + \frac{\partial v_z \tau_{zz}}{\partial z} + \rho f \cdot V$$

  Later we will see how these equations are derived and how to simulate them.

- **Schrödinger´s equation**

  This equation of quantum mechanics describes the wave function of a particle by prescribing the relationships among its mass, potential energy, and total energy. The time independent Schrödinger equation is

$$-\frac{\hbar^2}{2m}(\frac{\partial^2\psi}{\partial x^2}+\frac{\partial^2\psi}{\partial y^2}+\frac{\partial^2\psi}{\partial z^2})+V\psi=E\psi \tag{6}$$

  and the time dependent Schrödinger equation is

$$-\frac{\hbar^2}{2m}(\frac{\partial^2\psi}{\partial x^2}+\frac{\partial^2\psi}{\partial y^2}+\frac{\partial^2\psi}{\partial z^2})+V\psi=i\hbar\frac{\partial\psi}{\partial t} \tag{7}$$

  where $i=\sqrt{-1}$, m is the mass of the particle, E its energy, V the potential, ψ the wave function, $\hbar=\frac{h}{2\pi}$ and h is the Planck constant ($h=6.626076\cdot10^{-34}J\cdot s$).

- **Maxwell´s equations**

  Describe the behavior of an electromagnetic field by prescribing the relationships among the electric and magnetic field strengths, magnetic flux density, and electric charge and current densities.

$$\nabla\cdot E=\frac{\varrho}{\varepsilon_0} \tag{8}$$

$$\nabla\cdot B=0 \tag{9}$$

$$\nabla\times E=-\frac{\partial B}{\partial t} \tag{10}$$

$$\nabla\times B=\mu_0 j+\mu_0\varepsilon_0\frac{\partial E}{\partial t} \tag{11}$$

  where B is the magnetic field, E the electric field, ϱ the electric charge, j the current density, μ_0 magnetic permeability and ε_0 the dielectric constant.

In an ordinary differential equation the unknown function depends on only one variable. In PDE, the unknown function depends on several variables. Example: the one-dimensional heat equation:

$$q_t-q_{xx}=0 \tag{12}$$

Here we are searching the unknown function $q(x,t)$ that depends both on space and time.

Definition PDE [4]:

A partial differential equation is an equation involving partial derivatives of an unknown function with respect to more than one independent variable.

PDEs can be classified according to order, linearity, homogeneity, and so on. In Figure 1 (taken from Farlow) the main classifications of PDEs are illustrated. Another classifi-

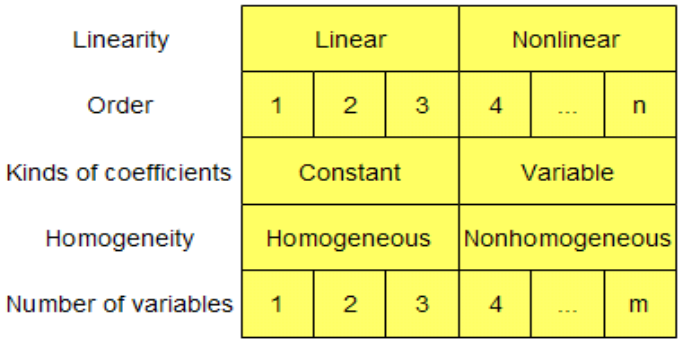

Figure 1: PDE classification

cation that is very important concerns the linear equations of the form

$$Aq_{xx} + Bq_{xy} + Cq_{yy} + Dq_x + Eq_y + Fq = G \tag{13}$$

where A, B, C, D, E, F, G are functions of x, y or just constants. This kind of equations can be classified according to the value of $B^2 - 4AC$:

$$\begin{cases} parabolic, & \text{if } B^2 - 4AC = 0 \\ hyperbolic, & \text{if } B^2 - 4AC > 0 \\ elliptic, & \text{if } B^2 - 4AC < 0 \end{cases} \tag{14}$$

The names come from the conic sections theory (Figure 2), where the general equation for a conic section is

$$ax^2 + bxy + cy^2 + dx + ey + f = 0 \tag{15}$$

and the conic is

$$\begin{cases} parabolic, & \text{if } b^2 - 4ac = 0 \\ hyperbolic, & \text{if } b^2 - 4ac > 0 \\ elliptic, & \text{if } b^2 - 4ac < 0 \end{cases} \tag{16}$$

Examples of such equations are:

parabolic PDE: $u_t = \alpha^2 u_{xx}$ (Diffusion equation)

hyperbolic PDE: $u_{tt} = c^2 u_{xx}$ (Wave equation)

elliptic PDE: $\nabla^2 u = 0$ (Laplace´s equation)

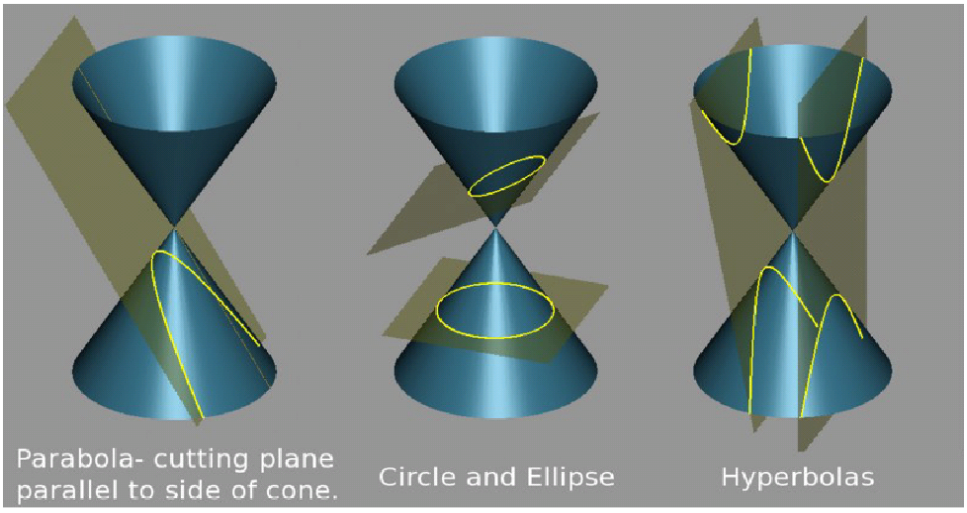

Figure 2: Conic sections (picture taken from Wikipedia

There are many methods to solve a partial differential equation. Some are analytical, some semi-analytical and some numerical. In practice, there are few PDEs that have an analytical solution, many interesting PDEs that arise in science don´t have an analytical solution. In the following some of the methods to solve PDEs are listed:

Separation of Variables

This method transforms a PDE in n variables into n ODEs, that can be then solved with ODE solvers.

Integral equations

Transforms a PDE to an integral equation which is then solved by various techniques.

Calculus of Variation Methods

The PDE equation is reformulated as a minimization problem. The minimum of the obtained expression is equal to the solution to the PDE.

Numerical Methods

There are many numerical methods to solve PDEs. Finite Difference Methods, Finite Element Methods and Finite Volume Methods are only some examples. We will later see a more detailed view of some of the most used numerical methods, in particular the Finite Volume Methods and the Method of Lines.

2.2 Constructing PDEs with Blocks

One of the goal of this master thesis is to construct basic blocks with which it is possible to construct almost any partial differential equation. As analogy let us take the example of the Lego game (Figure 3). In this game we have many building blocks of different

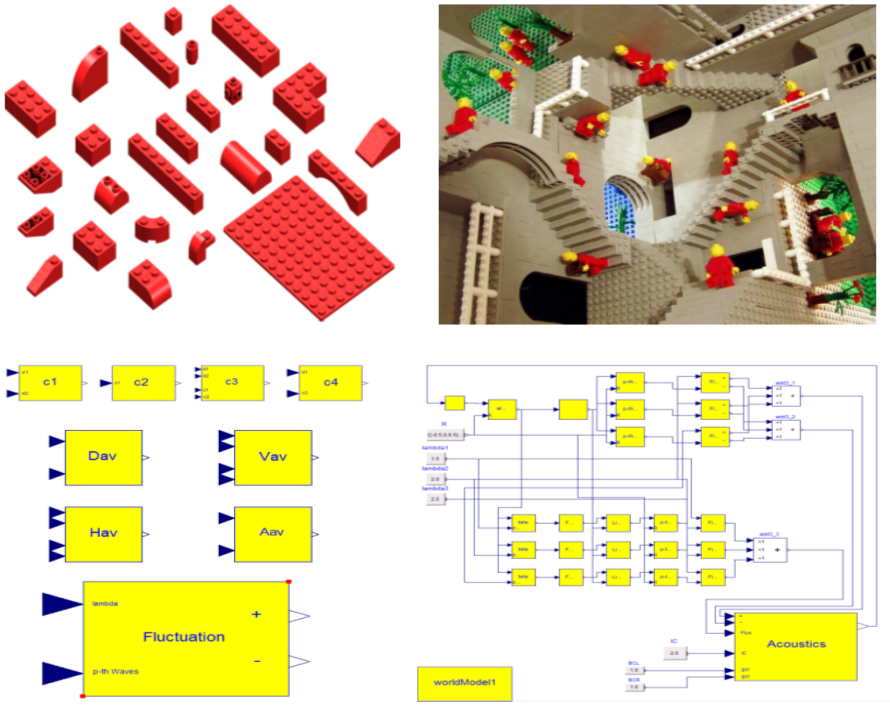

Figure 3: Lego analogy

types with which it is possible to construct complex objects, such as castles, planes, and so on. In the case of the partial differential equation the situation is more complicated, but the idea is the same: we have blocks like $\frac{\partial}{\partial x}$, $(\frac{\partial}{\partial x})^2$, integrators and so on and we want construct the PDE.

3 Computational Fluid Dynamics

3.1 Introduction

When studying fluids (liquids and gases), we are interested in how the properties of the fluid, such as velocity or pressure, change in space and time. To find a solution means to find the space-time change of these properties. When trying to describe the behavior of the gas in a tube for example, we must think of what ingredients are necessary to describe such a system. And here comes physics into play. We can start by saying that mass must be conserved for sure. This is right, but not enough. What are the other actors in this system? Well, it can maybe be surprising, but the fundamental equations to describe such a system are based on conservation laws:

1. Conservation of mass
2. Conservation of momentum
3. Conservation of energy

Fluids are governed by these conservation laws. The system of equations (PDEs), that result from these conservation laws, gives the "Euler equations" for gas dynamics. Because there is no analytical solution to this problem, it must be solved numerically. This is the task of computational fluid dynamics. As said in [8]: *Computational fluid dynamics (CFD) is the art of replacing such PDE systems by a set of algebraic equations that can be solved using digital computers.* For what is the CFD useful? For aerodynamics

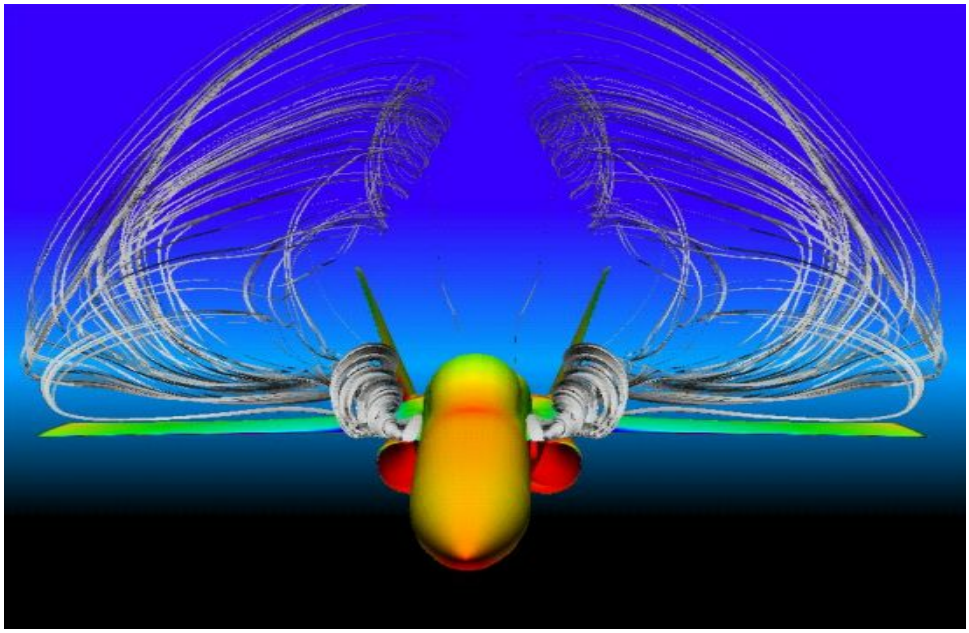

Figure 4: CFD visualization of vortices created by leading edge extensions on the F-18

purpose for example (Figure 4). By knowing the behavior of the flow around the airplane we can study and perfectionate the aerodynamics of the plane.
Sometimes we can perform simulations in a laboratory, by doing experiments, but there are many simulations that are not possible to do, or that are too expensive to perform in a laboratory. Say for example that an architect wants to know how, in the case of an accident, the fire will propagate through the building. This knowledge would allow the architect to decide for example where to position the windows and fire extinguishers. The distribution of the temperature will allow to differentiate regions with high temperature and influence maybe the choice of materials for the construction of those regions. Many other issues come into play. This kind of simulation cannot of course be done in reality. Nobody will position temperature sensors in the whole building and then burn it down.
CFD allows us to simulate situations that would otherwise be impossible to do, and give us a qualitative and quantitative description of the simulation. Of course, we must face problems that we have not in reality, such as the boundary conditions. By constructing the building in the virtual world by using for example the commercial CAD (Computer Aided Design) programs, we must specify the boundary conditions. This is a very difficult task, because there are no guidelines for this kind of problem, and the variety of materials and other factors are crucial for the simulation. In reality we would not have such problems, because the nature "knows" the boundary conditions whereas computers do not.
Before starting to derive the equations that describe the conservation laws, we must apply these physical principles to some model. There are two different ways of studying the problems in mechanics: Eulerian and Lagrange approach.

Eulerian Description

The Eulerian description tells us how the property changes at some place at some time. Say for example that we are interested in the behavior of the velocity field V at the point (x_p, y_p, z_p). Then the Eulerian description will describe the behaviour of the velocity field at this particular point over time: $V(x_p, y_p, z_p, t)$ (Figure 5).

Lagrangian description

If instead of the velocity at the particular point we are interested in the behavior of the velocity of the particle as it moves through the flow, then we are talking about the Lagrangian description (Figure 6).

In describing the Eulerian and Lagrangian approach we have considered the velocity field. This field is very important, because its knowledge allow us to determine many other properties of the fluid. In the following we list some of these properties that can be derived using the velocity, V:

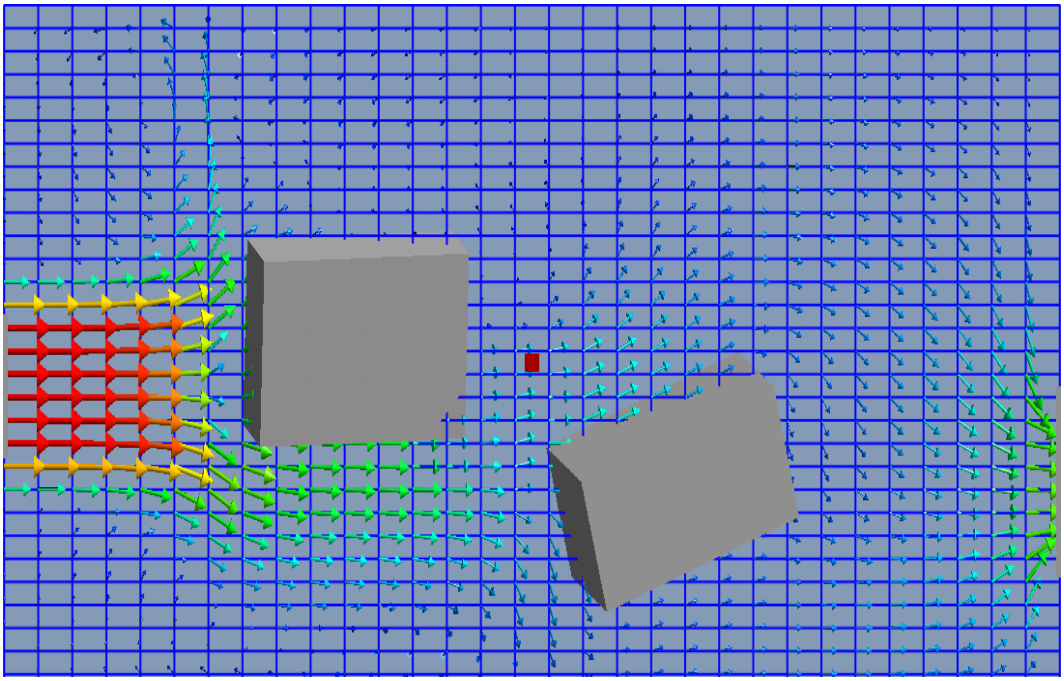

Figure 5: Eulerian approach: Velocity field

Displacement vector

$$x = \int V dt \tag{17}$$

Acceleration

$$a = \frac{dV}{dt} \tag{18}$$

Angular velocity

$$\omega = \frac{1}{2} \nabla \times V \tag{19}$$

where ∇ is the nabla operator.
Until now, we have considered only the density, pressure and velocity of the fluid. Internal energy, enthalpy, entropy and specific heat are other properties that become important when dealing with energy, work and heat. Additionally, the coefficient of viscosity

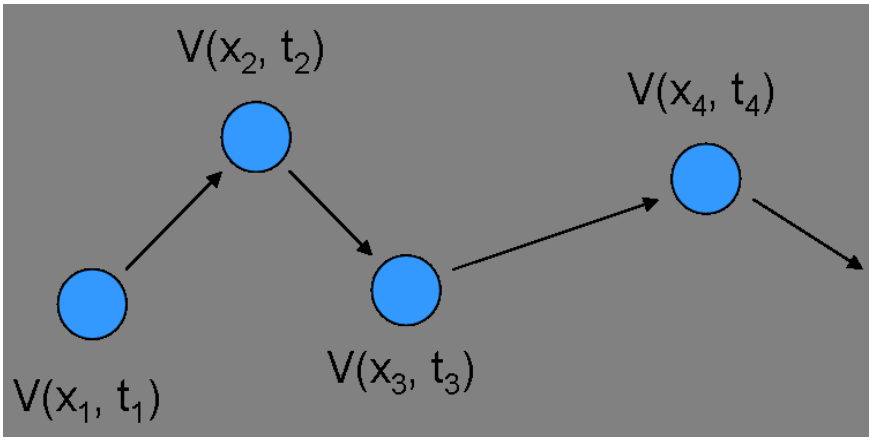

Figure 6: Lagrangian approach: Velocity field

and the thermal conductivity govern the friction and heat conduction. These nine thermodynamic properties are listed below:

1. Density
2. Pressure
3. Temperature
4. Internal energy
5. Entropy
6. Enthalpy
7. Specific heat
8. Viscosity
9. Thermal Conductivity

These quantities can be determined by the thermodynamic state of the fluid. For example for some substances, knowing the pressure and temperature is enough to determine the value of all the other quantities. We shall now concentrate on deriving the equations of conservations laws. As a model we choose an infinitesimally small fluid element that is fixed in space.

3.2 Euler Equations

3.2.1 The Continuity Equation

Consider the flow of the fluid through an infinitesimal small fluid element that is fixed in space (Figure 7). Let ρ be the fluid density and v its velocity. By using the Taylor

expansion (and neglecting higher order terms) we can derive the mass outflow for each space component:

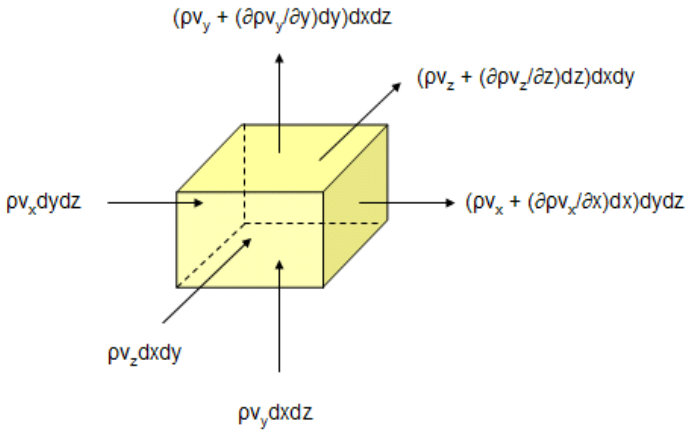

Figure 7: Mass outflow out of infinitesimal small fluid element

$$(\rho v_x + \frac{\partial \rho v_x}{\partial x} dx)dydz - \rho v_x dydz = \frac{\partial \rho v_x}{\partial x} dxdydz \tag{20}$$

$$(\rho v_y + \frac{\partial \rho v_y}{\partial y} dy)dxdz - \rho v_y dxdz = \frac{\partial \rho v_y}{\partial y} dxdydz \tag{21}$$

$$(\rho v_z + \frac{\partial \rho v_z}{\partial z} dz)dxdy - \rho v_z dxdy = \frac{\partial \rho v_z}{\partial z} dxdydz \tag{22}$$

Hence the total mass outflow is given by

$$(\frac{\partial \rho v_x}{\partial x} + \frac{\partial \rho v_y}{\partial y} + \frac{\partial \rho v_z}{\partial z})dxdydz \tag{23}$$

Since the mass of the infinitesimally small fluid element is $\rho dxdydz$, the change of mass over time inside the fluid element is

$$\frac{\partial \rho}{\partial t} dxdydz \tag{24}$$

The mass conservation law, applied to the infinitesimal small fluid element fixed in space,

states that the total mass outflow of the element must be equal to decrease of mass in time inside the element. By using the expressions we derived above, this could be written as:

$$(\frac{\partial \rho v_x}{\partial x} + \frac{\partial \rho v_y}{\partial y} + \frac{\partial \rho v_z}{\partial z})dxdydz = -\frac{\partial \rho}{\partial t}dxdydz \tag{25}$$

or

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho v) = 0 \tag{26}$$

This is the partial differential equation of the mass conservation law, called the continuity equation.

3.2.2 The Momentum Equation

We all know Newton´s second law

$$F = ma \tag{27}$$

which in the case of the fluid element states that the net force applied to the fluid element equals the mass of the element times its acceleration. Because we consider only one dimension, we write

$$F_x = ma_x \tag{28}$$

What kind of forces can act on a fluid element? There are two kind of forces [Andrew]:

- **Body forces**

  This kind of forces act directly on the volumetric mass of the fluid element. Example of these forces are gravitational and magnetic forces.

- **Surface forces**

  Act directly on the surface of the fluid element. They are due to only two sources:

  - *Pressure*

    The pressure distribution acting on the surface, imposed by the outside fluid surrounding the fluid element.

– *Viscous*

The shear and normal stress acting on the surface.

The schematic representation of these forces is presented in Figure 8. In Figure 9 a detailed view of forces acting on an infinitesimally small fluid element is illustrated.

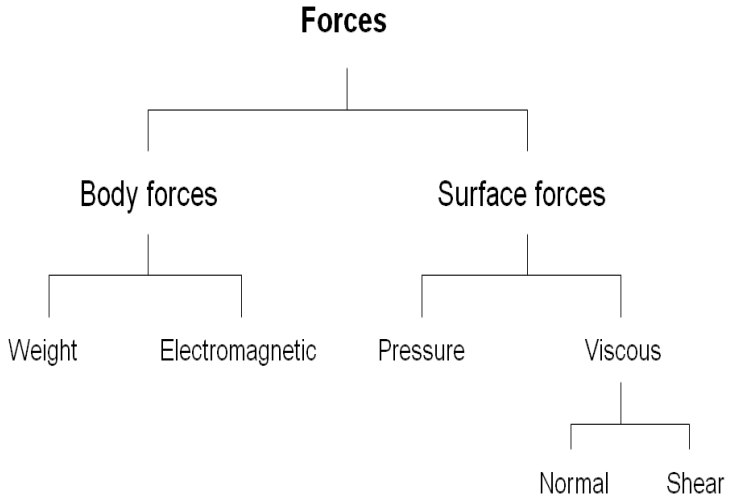

Figure 8: Forces that act on a fluid element

Let us denote the body force per unit mass acting on the fluid element in x-direction by f_x. Then we have:

$$\text{Body force} = \rho f_x dxdydz$$

As explained in [9]: *The shear and normal stresses in a fluid are related to the time rate of change of the deformation of the fluid element. The shear stress, τ_{xy}, is related to the time rate of change of the shearing deformation of the fluid element, whereas the normal stress, τ_{xx}, is related to the time rate of change of volume of the fluid element. As a result, both shear and normal stresses depend on velocity gradients in the flow. The notation τ_{ij} denotes a stress in the j direction exerted on a plane perpendicular to the i axis. The net surface force in x direction is*

$$(p - (p + \frac{\partial p}{\partial x}dx))dydz + ((\tau_{xx} + \frac{\partial \tau_{xx}}{\partial x}dx) - \tau_{xx})dydz + \tag{29}$$

$$((\tau_{yx} + \frac{\partial \tau_{yx}}{\partial y}dy) - \tau_{yx})dxdz + ((\tau_{zx} + \frac{\partial \tau_{zx}}{\partial z}dz) - \tau_{zx})dxdy$$

The total force in x direction, F_x, is the sum of the body force and the net surface force:

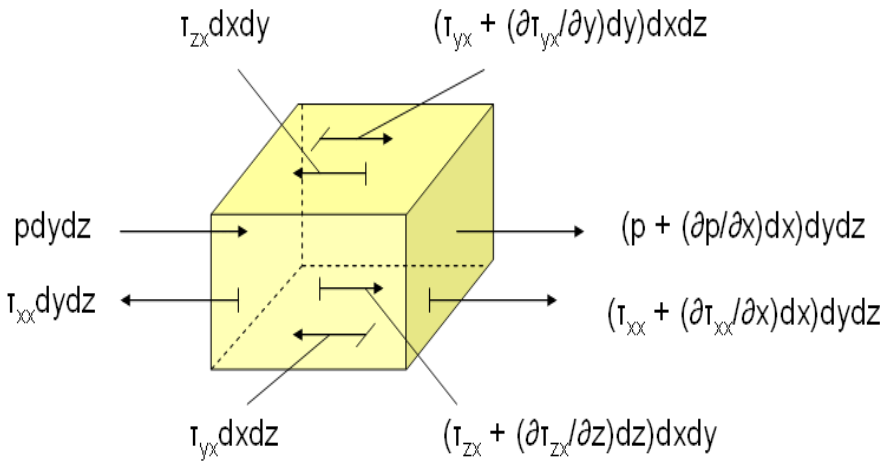

Figure 9: Forces acting on an infinitesimal small element

$$F_x = (-\frac{\partial p}{\partial x} + \frac{\partial \tau_{xx}}{\partial x} + \frac{\partial \tau_{yx}}{\partial y} + \frac{\partial \tau_{zx}}{\partial z})dxdydz + \rho f_x dxdydz$$

For the term ma in the Newton´s second law we have

$$\frac{\partial \rho v_x}{\partial t} + \nabla \cdot (\rho v_x^2) \tag{30}$$

And so Newton´s second law assumes the following form for the fluid element

$$\frac{\partial \rho v_x}{\partial t} + \nabla \cdot (\rho v_x^2) = (-\frac{\partial p}{\partial x} + \frac{\partial \tau_{xx}}{\partial x} + \frac{\partial \tau_{yx}}{\partial y} + \frac{\partial \tau_{zx}}{\partial z})dxdydz + \rho f_x dxdydz \tag{31}$$

This is the momentum equation. If we neglect viscous effects and body forces, the equation simplifies to:

$$\frac{\partial \rho v_x}{\partial t} + \nabla \cdot (\rho v_x^2) = -\frac{\partial p}{\partial x}dxdydz \tag{32}$$

This is exactly the second equation in our Euler system.

3.2.3 The Energy Equation

The first law of thermodynamics says that energy is conserved. In the following we shall apply this principle to the fluid element: *The rate of change of energy inside the fluid*

element is equal to the net flux of heat into the element plus the rate of work done on the element due to body and surface forces [9]. In Figure 10 we consider again the scheme illustrating the principal actors for the derivation of the energy equation.

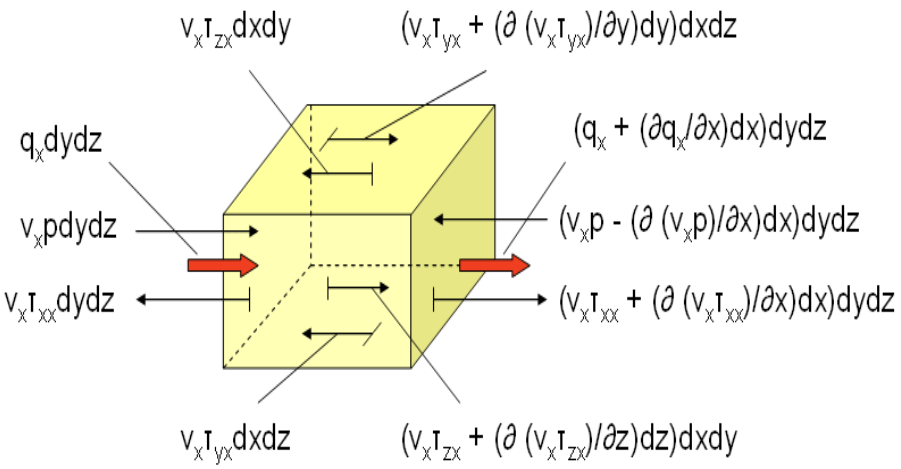

Figure 10: Energy in infinitesimal small fluid element

Here $\dot{q}$ is the rate of volumetric heat addition per unit mass. After following the same approach as for the continuity and momentum equation, we find the energy equation:

$$\begin{aligned}\frac{\partial}{\partial t}(\rho(e+\frac{V^2}{2}))+\nabla\cdot(\rho(e+\frac{V^2}{2})V)=\rho\dot{q}+ \\ \frac{\partial}{\partial x}(k\frac{\partial T}{\partial x})+\frac{\partial}{\partial y}(k\frac{\partial T}{\partial y})+\frac{\partial}{\partial z}(k\frac{\partial T}{\partial z})- \\ \frac{\partial v_x p}{\partial x}-\frac{\partial v_y p}{\partial y}-\frac{\partial v_z p}{\partial z}+\frac{\partial v_x\tau_{xx}}{\partial x}+\frac{\partial v_x\tau_{yx}}{\partial y}+\frac{\partial v_x\tau_{zx}}{\partial z}+ \\ \frac{\partial v_y\tau_{xy}}{\partial x}+\frac{\partial v_y\tau_{yy}}{\partial y}+\frac{\partial v_y\tau_{zy}}{\partial z}+ \\ \frac{\partial v_z\tau_{xz}}{\partial x}+\frac{\partial v_z\tau_{yz}}{\partial y}+\frac{\partial v_z\tau_{zz}}{\partial z}+\rho f\cdot V\end{aligned} \tag{33}$$

3.3 Navier-Stokes and Euler Equations

The equations derived so far govern the viscous flow. *In the viscous flow the transport phenomena of friction, thermal conduction, and mass diffusion are included.* To summarize what we have derived and by adding the other components not considered in the derivation we obtain the governing equations for an unsteady, three-dimensional,

compressible, viscous flow, called the Navier-Stokes equations:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho V) = 0 \tag{34}$$

$$\frac{\partial \rho v_x}{\partial t} + \nabla \cdot (\rho v_x V) = -\frac{\partial p}{\partial x} + \frac{\partial \tau_{xx}}{\partial x} + \frac{\partial \tau_{yx}}{\partial y} + \frac{\partial \tau_{zx}}{\partial z} + \rho f_x \tag{35}$$

$$\frac{\partial \rho v_y}{\partial t} + \nabla \cdot (\rho v_y V) = -\frac{\partial p}{\partial y} + \frac{\partial \tau_{xy}}{\partial x} + \frac{\partial \tau_{yy}}{\partial y} + \frac{\partial \tau_{zy}}{\partial z} + \rho f_y \tag{36}$$

$$\frac{\partial \rho v_z}{\partial t} + \nabla \cdot (\rho v_z V) = -\frac{\partial p}{\partial z} + \frac{\partial \tau_{xz}}{\partial x} + \frac{\partial \tau_{yz}}{\partial y} + \frac{\partial \tau_{zz}}{\partial z} + \rho f_z \tag{37}$$

$$\begin{aligned} \frac{\partial}{\partial t}(\rho(e + \frac{V^2}{2})) + \nabla \cdot (\rho(e + \frac{V^2}{2})V) = \rho\dot{q} + \\ \frac{\partial}{\partial x}(k\frac{\partial T}{\partial x}) + \frac{\partial}{\partial y}(k\frac{\partial T}{\partial y}) + \frac{\partial}{\partial z}(k\frac{\partial T}{\partial z}) - \\ \frac{\partial v_x p}{\partial x} - \frac{\partial v_y p}{\partial y} - \frac{\partial v_z p}{\partial z} + \frac{\partial v_x \tau_{xx}}{\partial x} + \frac{\partial v_x \tau_{yx}}{\partial y} + \frac{\partial v_x \tau_{zx}}{\partial z} + \\ \frac{\partial v_y \tau_{xy}}{\partial x} + \frac{\partial v_y \tau_{yy}}{\partial y} + \frac{\partial v_y \tau_{zy}}{\partial z} + \\ \frac{\partial v_z \tau_{xz}}{\partial x} + \frac{\partial v_z \tau_{yz}}{\partial y} + \frac{\partial v_z \tau_{zz}}{\partial z} + \rho f \cdot V \end{aligned} \tag{38}$$

If however, we neglect the dissipative transport phenomena of viscosity, mass diffusion, and thermal conductivity, then we obtain from the equations above the equations for an inviscid flow. The resulting equations describe an unsteady, three-dimensional, compressible inviscid flow:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho V) = 0 \tag{39}$$

$$\frac{\partial \rho v_x}{\partial t} + \nabla \cdot (\rho v_x V) = -\frac{\partial p}{\partial x} + \rho f_x \tag{40}$$

$$\frac{\partial \rho v_y}{\partial t} + \nabla \cdot (\rho v_y V) = -\frac{\partial p}{\partial y} + \rho f_y \tag{41}$$

$$\frac{\partial \rho v_z}{\partial t} + \nabla \cdot (\rho v_z V) = -\frac{\partial p}{\partial z} + \rho f_z \tag{42}$$

$$\frac{\partial}{\partial t}(\rho(e + \frac{V^2}{2})) = \rho\dot{q} - \frac{\partial(v_x p)}{x} - \frac{\partial(v_y p)}{y} - \frac{\partial(v_z p)}{z} + \rho f \cdot V \tag{43}$$

These are the Euler equations.

3.4 Conservation Laws

In Chapter 3.2 we derived the conservation laws composing the Euler system. In this chapter we will look at another way of deriving these equations (for simplicity we shall consider only the continuity equation), which is more suitable for the understanding of the finite volume method. Let us take as example a thin tube with section area A, along which a fluid can flow. Consider now some portion of the tube $[x_1, x_2]$ of length L (Figure 11). We are interested in how the fluid quantity changes in this portion of the

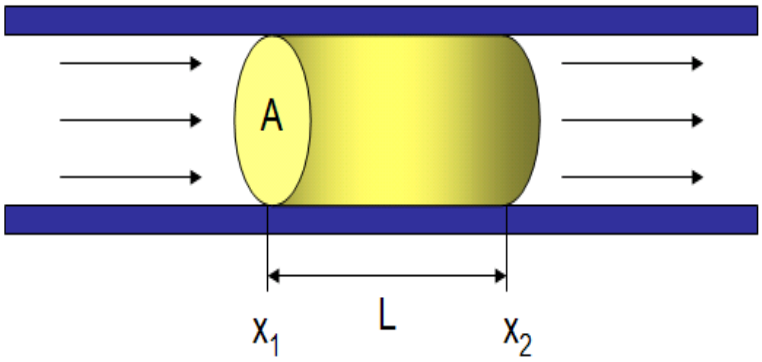

Figure 11: Tube

tube. Let d be the three dimensional density of the fluid. Multiplying it by the section area A we get the density $q(x,t)$, which tells us the quantity of the fluid per meter. If we take now the small volume in the section $[x_1, x_2]$ with the length dx, the fluid quantity in the volume $A*dx$ will be equal to $q(x,t)*dx$. The total quantity in the section will be then

$$\int_{x_1}^{x_2} q(x,t)dx \tag{44}$$

How does this quantity change? If we assume that the fluid is neither created nor destroyed in the section, then the quantity of the fluid in the section can only change due to fluxes through the points x_1 and x_2. Let $F(x_i,t)$ be the flux at the point x_i at time t. Then we can write:

$$\frac{d}{dt}\int_{x_1}^{x_2} q(x,t)dx = F(x_1,t) - F(x_2,t) \tag{45}$$

This is the integral form of the conservation law. It says that the rate of change of the total mass is due only to fluxes through the endpoints. It is important to note that although in this example q is the mass density, we can consider in general q as the density of some quantity.
If we assume that q and $f(q)$ are sufficiently smooth, we can derive the differential form of the conservation law by using the integral form.

$$\frac{d}{dt}\int_{x_1}^{x_2} q(x,t)dx = F(x_1,t) - F(x_2,t) \tag{46}$$

$$\frac{d}{dt}\int_{x_1}^{x_2} q(x,t)dx = -\int_{x_1}^{x_2} \frac{\partial}{\partial x} f(q(x,t))dx \tag{47}$$

$$\int_{x_1}^{x_2} (\frac{\partial}{\partial t} q(x,t)dx + \frac{\partial}{\partial x} f(q(x,t)))dx = 0 \tag{48}$$

Since this integral must be zero for all values of x_1 and x_2, it follows that the integrand must be identically zero

$$\frac{\partial}{\partial t} q(x,t)dx + \frac{\partial}{\partial x} f(q(x,t)) = 0 \tag{49}$$

and we obtain the differential form of the conservation law. It is important to note that if q has a discontinuity at some point, like in the case of shock waves, then the differential form of the conservation law does not hold. The integral form of the conservation law, however, continues to hold. This is the basic idea of the finite volume method, which is based on the integral form instead of the differential form that is used for example in the finite-difference method.
It remains to establish the relation between the flux function and the density.
In the case of fluid flow, the flux at any point x at time t is simply the product of the density $q(x,t)$ and the velocity $v(x,t)$:

$$flux(x,t) = v(x,t)q(x,t) \tag{50}$$

Since the velocity $v(x,t)$ tells us at which velocity the particles move through the point x, say m/s, and the density $q(x,t)$ tells us the quantity of mass in a meter of fluid, say g/m,then the product $v(x,t)q(x,t)$ gives us the the rate at which the mass is passing through the point x. Consider for example the tube in Figure 12. In time Δt the particles moved a distance $v(x_1,t)\Delta t$ from the point x_1. The volume of the red piece is $Av(x_1,t)\Delta t$. Hence the mass contained in this piece is $d(x,t)Av(x_1,t)\Delta t$. Since $d(x,t)A = q(x,t)$, we have $q(x,t)v(x_1,t)\Delta t$, and if we take $\Delta t = 1$ (unit of time), we arrive finally at $q(x,t)v(x_1,t)$ which tells us the rate at which the mass is passing through the point x_1. In the particular case where the velocity is constant, $v(x,t) = \overline{v}$, the flux at the point x in time t is

$$flux(x,t) = f(q) = \overline{v}q(x,t) \tag{51}$$

Thus the flux at the point x at time t depends in this case directly on the conserved

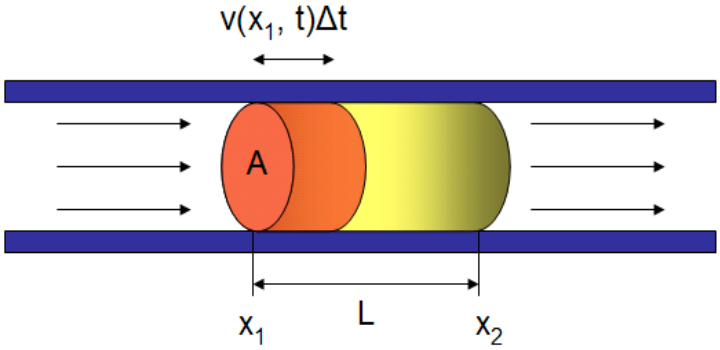

Figure 12: Tube: red volume contains the quantity of mass entered through the point x_1 in time Δt

quantity q at that point and time. In this case the equation is called *autonomous*.
So far we assumed that the cross-sectional area is constant. In Figure 13 we see an example of a tube with four different cross-sectional areas. In this case the cross-sectional

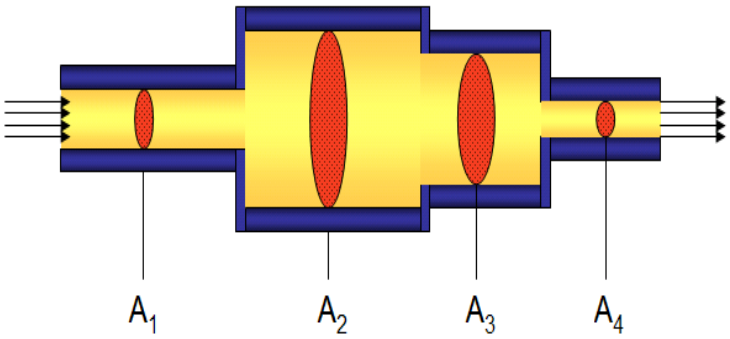

Figure 13: Tube

area varies with space. This has an impact on the velocity of the fluid, which will be now different in different regions of the tube. In the following we shall not consider this case but the interested reader can consult the chapter 9 of the Randall LeVecque book [2].
By following the same approach we can find the momentum flux and energy flux and obtain in the end the following system of equations:

$$\begin{cases} \rho_t + (\rho v)_x = 0 \\ (\rho v)_t + (\rho v^2 + p)_x = 0 \\ E_t + ((E+p)v)_x = 0 \end{cases} \tag{52}$$

where

- Mass flux: ρv
- Momentum flux: $\rho v^2 + p$
- Energy flux: $(E + p)v$

Thus we are searching the unknown functions ρ, v, p and E that satisfies simultaneously the three equations. In other words, at each time, at every location in the domain the three conservation laws must be all satisfied.
The system just obtained has four unknowns ρ, v, p and E and only three equations. In order to complete the problem we need the fourth equation. This can be obtained from the equation of state of ideal gases. For example, for the polytropic gas we have:

$$p = (\gamma - 1)(E - \rho\frac{v^2}{2}) \tag{53}$$

where $\gamma = 1.4$ for air. In polytropic gases internal energy is simply proportional to the temperature. With the fourth equation we have now the complete system:

$$\begin{cases} \rho_t + (\rho v)_x = 0 \\ (\rho v)_t + (\rho v^2 + p)_x = 0 \\ E_t + ((E + p)v)_x = 0 \\ p = (\gamma - 1)(E - \rho\frac{v^2}{2}) \end{cases} \tag{54}$$

In the following chapters we will solve this system with the Method of Lines and Finite Volume Methods.

4 Method of Lines

4.1 Introduction

The Method of Lines (MOL) technique solves numerically a PDE by discretizing the space variable but leaving the time variable continuous. By doing so we obtain a system of ordinary differential equations (ODEs). The MOL method converts thus a PDE into a set of ODEs. This is advantageous because efficient ODE solvers are known. Consider for example the one dimensional heat equation problem:

$$\frac{\partial u}{\partial t} = \sigma \cdot \frac{\partial^2 u}{\partial x^2} \tag{55}$$

$$u(x,0) = cos(\pi x) \tag{56}$$

$$u(0,t) = e^{\frac{-t}{10}} \tag{57}$$

$$\frac{\partial u}{\partial x}(1,t) = 0 \tag{58}$$

where σ is a constant value in this problem. The solution to this equation is a function $u(x,t)$. The MOL technique discretizes the spatial variable and looks at the solutions $u(x_i,t)$. In Figure 14 the basic idea of the MOL approach is illustrated: the solution is computed along a series of lines, resulting in cross sections of the solution surface. We replace the second-order derivative by a finite difference

$$\frac{\partial^2 u}{\partial x^2} \approx \frac{u_{i+1} - 2u_i + u_{i-1}}{\delta x^2} \tag{59}$$

where δx is the grid width. Inserting into equation 55 we obtain

$$\frac{du_i}{dt} = \sigma \cdot \frac{u_{i+1} - 2u_i + u_{i-1}}{\delta x^2} \tag{60}$$

Thus we changed the PDE in u into a set of ODEs in u.
In the case of five grid points we have:

$$u_1 = \exp^{\frac{-t}{10}} \tag{61}$$

$$u_2 = \frac{n^2}{10\pi^2}(u_3 - 2u_2 + u_1) \tag{62}$$

$$u_3 = \frac{n^2}{10\pi^2}(u_4 - 2u_3 + u_2) \tag{63}$$

$$u_4 = \frac{n^2}{10\pi^2}(u_5 - 2u_4 + u_2) \tag{64}$$

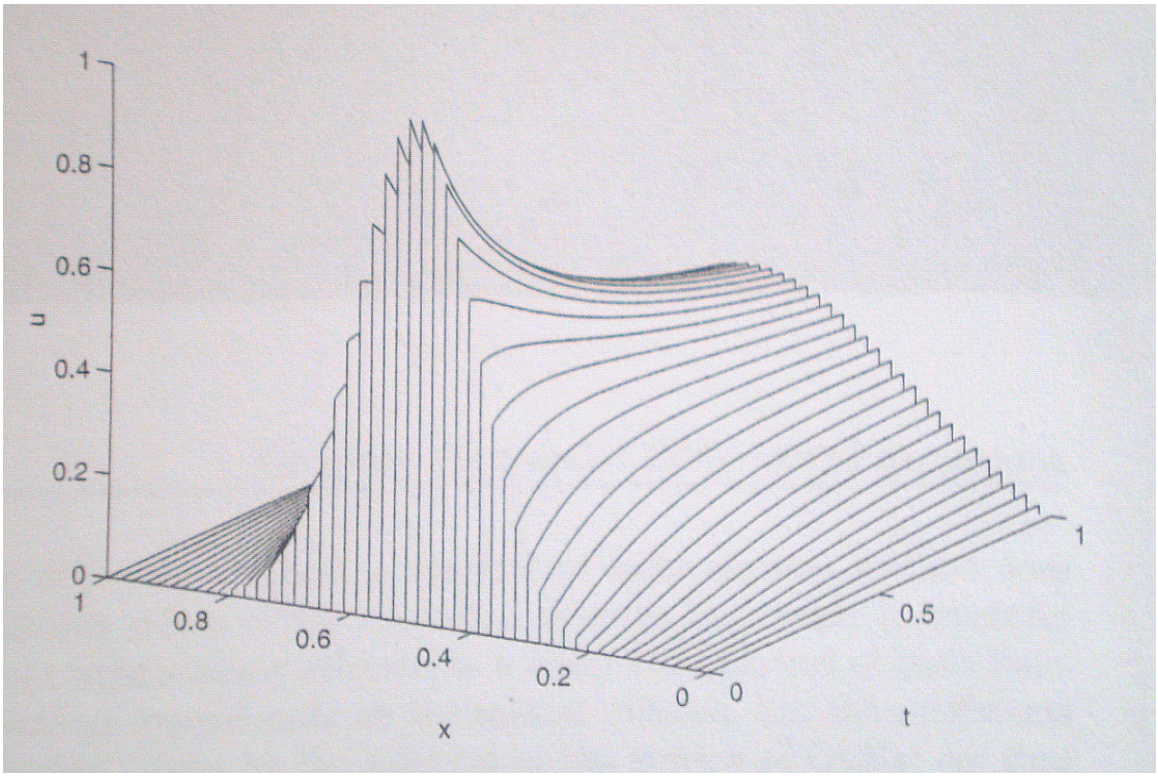

Figure 14: MOL solution of heat equation (picture taken from [4])

$$u_5 = \frac{n^2}{10\pi^2}(u_6 - 2u_5 + u_4) \tag{65}$$

$$u_6 = \frac{n^2}{5\pi^2}(-u_6 + u_5) \tag{66}$$

with the initial conditions:

$$u_i = cos(\frac{(i-1)\pi}{i}) \tag{67}$$

for i = 2, ..., 5. With this boundary and initial conditions, we can now apply ODE solvers to find the solution.
In Figure 15 the Modelica solution of this problem is illustrated. The system of ODEs we obtain above can be written in matrix form

$$u_t = \frac{\sigma}{(\delta x)^2}\begin{pmatrix} -2 & 1 & 0 & \dots & 0 \\ 1 & -2 & 1 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & 1 & -2 \end{pmatrix} u = Au \tag{68}$$

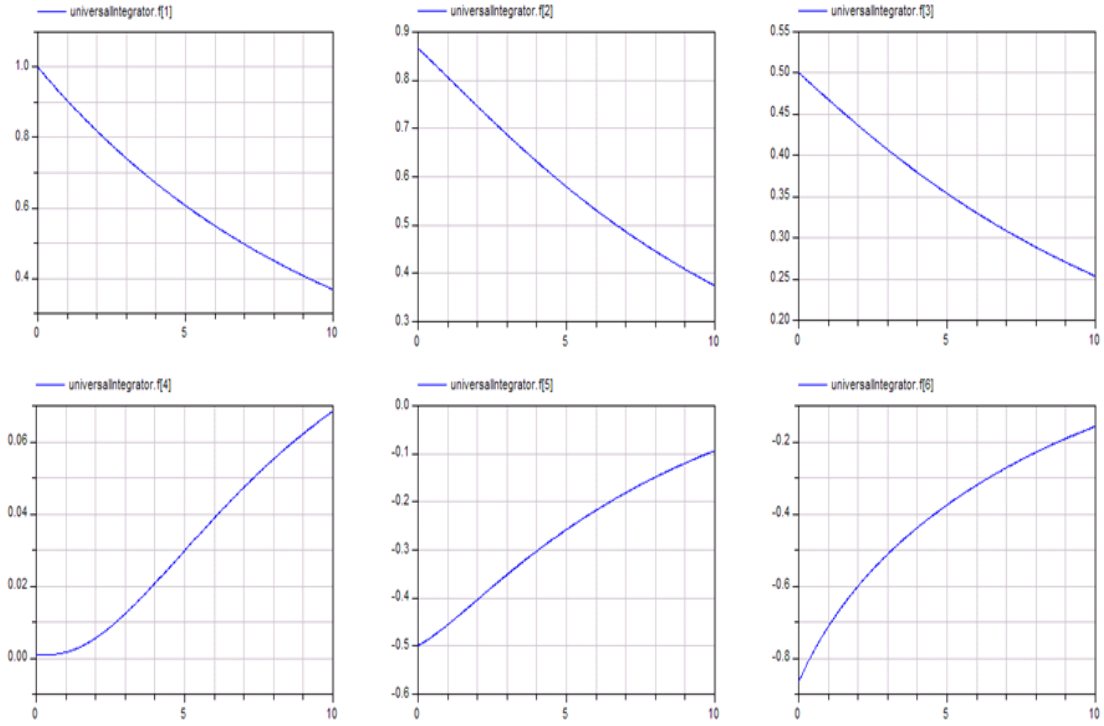

Figure 15: Modelica solution of the heat equation to the six grid points

The Jacobian of the matrix A of this system has eigenvalues in the range $[\frac{-4\sigma}{(\Delta x)^2}]$, which makes the ODE very stiff as the grid size Δx decreases. This stiffness, which is typical of ODEs derived from a parabolic PDE in this way, must be taken into account in choosing an appropriate ODE method for solving the semidiscrete system. Although it seems that the MOL and finite-difference techniques are completely different, because the former does not discretize the time variable, whereas the latter does, there is no real difference between the two approaches, because the time variable is discretized in both methods. Whereas in MOL approach it is the ODE software package that has the task to appropriately choose the time step in order to maintain stability and achieve the desired accuracy, in finite difference technique the task of choosing the time step is left to the user. In Figure 16 the basic MOL steps are illustrated. Let us see these points in more details.

4.2 Discretization

In 1D the discretization step is easy, we must only decide how many grid points we need, say n, and the domain will be partitioned into $n-1$ intervals of length $\frac{1}{n-1}$. It is also possible to subdivide the domain non uniformly, or better, discretize the domain during

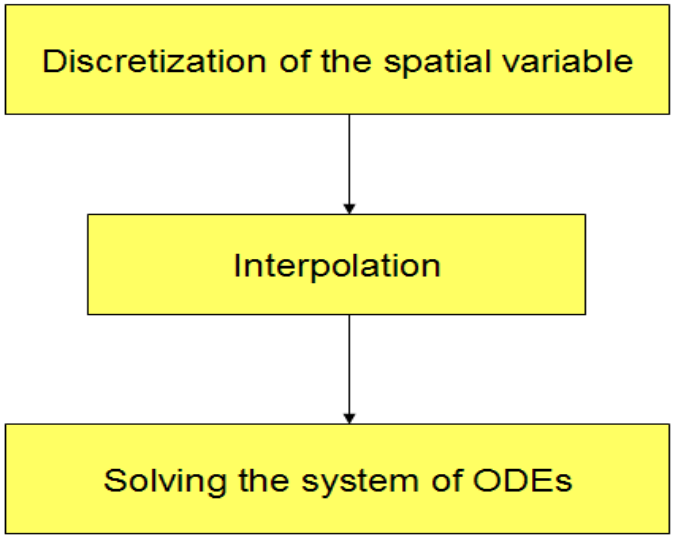

Figure 16: The MOL process

the simulation run according to some criteria. This way regions which are interesting could be subdivided finer. This topic is discussed in chapter 5. In the following we will adopt the uniform discretization.

4.3 Interpolation

In the above example we used the second-order central difference formula to approximate the second-order derivative. What if we need a more accurate approximation? We could interpolate by using more points. Consider for example the Newton-Gregory polynomials and say that we want to fit a polynomial of the fourth order. From algebra we know that five points are needed to fit the fourth order polynomial. If we use the central differences, this means that we fit the polynomial through the five points x_{i-2}, x_{i-1}, x_i, x_{i+1} and x_{i+2}.
If we use Newton-Gregory backward polynomials, we need to write the polynomial around the most right point, which in our example is x_{i+2}. After some calculations we obtain the following discretization formula for the second-order derivative at point x_i:

$$\frac{\partial^2 u}{\partial x^2} \mid_{x=x_i} = \frac{-u_{i+2} + 16u_{i+1} - 30u_i + 16u_{i-1} - u_{i-2}}{12\delta x^2} \tag{69}$$

which is the fourth-order central difference approximation. Let us first take a quick look at how Newton interpolation works. Newton polynomial has the following form:

$$P_n(x) = b_0 + b_1(x - x_0) + b_2(x - x_0)(x - x_1) + ... + b_n(x - x_0) \cdot ... \cdot (x - x_{n-1}) \tag{70}$$

Because this polynomial must fit the points $x_0, ..., x_n$, we have the system of equations:

$$f(x_0) = b_0 \tag{71}$$

$$f(x_1) = b_0 + b_1(x_1 - x_0) \tag{72}$$

$$f(x_2) = b_0 + b_1(x_2 - x_0) + b_2(x_2 - x_0)(x_2 - x_1) \tag{73}$$

$$...$$

Solving for b_i we obtain:

$$b_0 = f(x_0) =: f[x_0] \tag{74}$$

$$b_1 = \frac{f(x_1) - f(x_0)}{x_1 - x_0} =: f[x_0 x_1] \tag{75}$$

$$b_2 = \frac{f[x_0 x_2] - f[x_0 x_1]}{x_2 - x_1} =: f[x_0 x_1 x_2] \tag{76}$$

$$...$$

Thus we obtain the following polynomial:

$$P(x) = f[x_0] + \sum_{k=1}^{n} f[x_0...x_k](x - x_0) \cdot ... \cdot (x - x_{k-1}) \tag{77}$$

which is called Newton´s interpolatory divided-difference formula. This polynomial assumes a simpler form if $x_0, ..., x_n$ are spaced equally, that is, if h $= x_{i+1} - x_i$ for each $i = 0, ..., n - 1$:

$$P(x) = P(x_0 + sh) = f[x_0] + \sum_{k=1}^{n} \binom{s}{k} k! h^k f[x_0...x_k] \tag{78}$$

The polynomial is further simplified by introducing the forward difference $\Delta f(x_k) = f(x_{k+1}) - f(x_k)$ notation which substitute $f[x_0...x_k]$ by

$$f[x_0...x_k] = \frac{1}{k! h^k} \Delta^k f(x_0) \tag{79}$$

and enables us to write the polynomial in the following form

$$P(x) = f[x_0] + \sum_{k=1}^{n} \binom{s}{k} \Delta^k f(x_0) \tag{80}$$

which is called the Newton forward-difference formula. If the interpolating nodes are

written as $x_n, ..., x_0$ we can write

$$P(x) = f[x_n] + ... + f[x_0...x_n](x - x_n) \cdot ... \cdot (x - x_1) \tag{81}$$

If the interpolating nodes are equally spaced, with $x = x_n + sh$ the polynomial can be written as

$$P(x) = f[x_n] + ... + s(s+1) \cdot ... \cdot (s+n-1)h^n f[x_0...x_n] \tag{82}$$

and by introducing the backward difference $\nabla f(x_k) = f(k) - f(k-1)$ we can substitute $f[x_{n-k}...x_n]$ by

$$f[x_{n-k}...x_n] = \frac{1}{k!h^k}\nabla^k f(x_n) \tag{83}$$

and write the polynomial in the form

$$P(x) = f[x_n] + \sum_{k=1}^{n}(-1)^k \binom{-s}{k} \nabla^k f(x_n) \tag{84}$$

which is called the Newton backward-difference formula.
Having the polynomial we can compute its first derivative. For the Newton backward difference formula this is

$$\dot{P}(x) = \frac{d}{dt}P(x) = \frac{\partial}{\partial s}P(s) \cdot \frac{ds}{dt} \approx \tag{85}$$
$$\frac{1}{h}(\nabla P(x_0) + (s + \frac{1}{2})\nabla^2 P(x_0) + (\frac{3s^2 + 6s + 2}{6})\nabla^3 P(x_0) + ...)$$

and the second derivative is

$$\ddot{P}(x) \approx \frac{1}{h^2}(\nabla^2 P(x_0) + (s+1)\nabla^3 P(x_0) + (\frac{s^2}{2} + \frac{3s}{2} + \frac{11}{12})\nabla^4 P(x_0) + ...) \tag{86}$$

We can now use this information to solve the heat equation. If we desire a second-order central difference approximation, then we need to fit the polynomial through the three points x_{i-1}, x_i and x_{i+1}. By using the Newton-Gregory backward polynomial, we write the polynomial around the point x_{i+1} and obtain for the second derivative

$$\ddot{P}(x) \approx \frac{1}{h^2}(\nabla^2 P(x_{i+1}) + (s+1)\nabla^3 P(x_{i+1})) \tag{87}$$

To evaluate the second derivative around the point x_i we have to set $s = -1$ and we

obtain

$$\frac{\partial^2 P}{\partial x^2}|_{x=x_i} \approx \frac{1}{h^2}(\nabla^2 P(x_{i+1})) = \frac{1}{h^2}(P(x_{i+1}) - 2P(x_i) + P(x_{i+1})) \tag{88}$$

Say that we want now a fourth-order central difference approximation. In this case we need to fit the polynomial through the five points x_{i-2}, x_{i-1}, x_i, x_{i+1} and x_{i+2}. If we use the Newton-Gregory backward polynomial, we need to write the polynomial around the point x_{i+2}. The second derivative will then be:

$$\ddot{P}(x) \approx \frac{1}{h^2}(\nabla^2 P(x_{i+2}) + (s+1)\nabla^3 P(x_{i+2}) + (\frac{s^2}{2} + \frac{3s}{2} + \frac{11}{12})\nabla^4 P(x_{i+2}) + ...) \tag{89}$$

and if we want to evaluate the second derivative at the point x_i, we just set $s = -2$ and obtain

$$\frac{\partial^2 P}{\partial x^2}|_{x=x_i} \approx \frac{1}{h^2}(\nabla^2 P(x_{i+2}) - \nabla^3 P(x_{i+2}) - \frac{1}{12}\nabla^4 P(x_{i+2})) = \tag{90}$$

$$\frac{1}{h^2}((P(x_{i+2}) - 2P(x_{i+1}) + P(x_i)) - (P(x_{i+2}) - 3P(x_{i+1}) + 3P(x_i) - P(x_{i-1}))$$

$$-\frac{1}{12}(P(x_{i+2}) - 4P(x_{i+1}) + 6P(x_i) - 4P(x_{i-1}) + P(x_{i-2}))) =$$

$$\frac{1}{12h^2}(-P(x_{i+2}) + 16P(x_{i+1}) - 30P(x_i) + 16P(x_{i+1}) - P(x_{i-2}))$$

Let us now consider the boundaries of the domain. If we use the second-order central difference scheme, then we see that the formula we derived above does not apply for the points x_1 and x_n because in either cases we need points that lie outside the domain. The same applies to the fourth-order central difference scheme where the "special" points are x_1, x_2, x_{n-1} and x_n. In this case we can use biased formulas. For the fourth-order central difference scheme we can find the formulas for the point x_1 by using the Newton-Gregory polynomial around the point x_5 and then set $s = -4$ and for x_2 set $s = -3$. The same ideas apply to the other points. Finally we obtain

$$\frac{\partial^2 P}{\partial x^2}|_{x=x_1} = \frac{1}{12h^2}(-11P(x_5) - 56P(x_4) + 114P(x_3) - 104P(x_2) + 35P(x_1)) \tag{91}$$

$$\frac{\partial^2 P}{\partial x^2}|_{x=x_2} = \frac{1}{12h^2}(-P(x_5) + 4P(x_4) + 6P(x_3) - 20P(x_2) + 11P(x_1)) \tag{92}$$

$$\frac{\partial^2 P}{\partial x^2}|_{x=x_{n-1}} = \frac{1}{12h^2}(11P(x_n) - 20P(x_{n-1}) + 6P(x_{n-2}) + 4P(x_{n-3}) - P(x_{n-4})) \tag{93}$$

$$\frac{\partial^2 P}{\partial x^2}|_{x=x_n} = \frac{1}{12h^2}(35P(x_n) - 104P(x_{n-1}) + 114P(x_{n-2}) - 56P(x_{n-3}) + 11P(x_{n-4})) \tag{94}$$

In the following we will use the notation u_i, indicating the value of the function u at

the grid point x_i. The approximation to the second-derivative discussed so far is implemented in u_{xx} block (Figure 17). We can proceed the same way to find the formulas for

```
if bcl == -1 then
  y[1] = (1/(12*deltax^2))*(-2*u[3] + 32*u[2] - 30*u[1]);
  y[2] = (1/(12*deltax^2))*(-u[4] + 16*u[3] - 31*u[2] + 16*u[1]);
else
  y[1] = (1/(12*deltax^2))*(11*u[5] - 56*u[4] + 114*u[3] - 104*u[2] + 35*u[1]);
  y[2] = (1/(12*deltax^2))*(-u[5] + 4*u[4] + 6*u[3] - 20*u[2] + 11*u[1]);
end if;

for i in 3:n-2 loop
  y[i] = (1/(12*deltax^2))*(-u[i+2] + 16*u[i+1] - 30*u[i] + 16*u[i-1] - u[i-2]);
end for;

if bcr == -1 then
  y[n-1] = (1/(12*deltax^2))*(16*u[n] - 31*u[n-1] + 16*u[n-2] - u[n-3]);
  y[n] = (1/(12*deltax^2))*(-30*u[n] + 32*u[n-1] - 2*u[n-2]);
else
  y[n-1] = (1/(12*deltax^2))*(11*u[n] - 20*u[n-1] + 6*u[n-2] + 4*u[n-3] - u[n-4]);
  y[n] = (1/(12*deltax^2))*(35*u[n] - 104*u[n-1] + 114*u[n-2] - 56*u[n-3] + 11*u[n-4]);
end if;
```

Figure 17: Portion of u_{xx} block code

the first-derivative. If we wish to obtain the second-order central difference scheme, we use the formula (94) to write the polynomial around the point x_{i+1} and set $s = -1$. In this way we obtain

$$\dot{P}(x) = \frac{1}{h}(\nabla P(x_{i+1}) - \frac{1}{2}\nabla^2 P(x_{i+1})) = \tag{95}$$
$$\frac{1}{h}(u_{i+1} - u_i - \frac{1}{2}(u_{i+1} - 2u_i + u_{i-1})) = \frac{u_{i+1} - u_{i-1}}{2h}$$

For the boundary point x_1 we use the biased formula by interpolating around u_3 and then setting $s = -2$:

$$\dot{P}(x_1) = \frac{1}{h}(\nabla P(x_3) - (\frac{2s+1}{2})\nabla^2 P(x_3)) = \tag{96}$$
$$\frac{1}{h}(\nabla P(x_3) - \frac{3}{2}\nabla^2 P(x_3)) =$$
$$\frac{1}{h}(u_3 - u_2 - \frac{3}{2}(u_3 - 2u_2 + u_1)) = \frac{-u_3 + 4u_2 - 3u_1}{2h}$$

In the same way we obtain the biased formula for the boundary point x_n:

$$\dot{P}(x_n) \approx \frac{3u_n - 4u_{n-1} + u_{n-2}}{2h} \tag{97}$$

In the same way we obtain the fourth-order central difference scheme

$$\frac{\partial P}{\partial x}|_{x=x_1} = \frac{-3u_5 + 16u_4 - 36u_3 + 48u_2 - 25u_1}{12h} \tag{98}$$

$$\frac{\partial P}{\partial x}|_{x=x_2} = \frac{u_5 - 6u_4 + 18u_3 - 10u_2 - 3u_1}{12h} \tag{99}$$

$$\frac{\partial P}{\partial x}|_{x=x_i} = \frac{-u_{i+2} + 8u_{i+1} - 8u_{i-1} + u_{i-2}}{12h} \tag{100}$$

$$\frac{\partial P}{\partial x}|_{x=x_{n-1}} = \frac{3u_n + 10u_{n-1} - 18u_{n-2} + 6u_{n-3} - u_{n-4}}{12h} \tag{101}$$

$$\frac{\partial P}{\partial x}|_{x=x_n} = \frac{25u_n - 48u_{n-1} + 36u_{n-2} - 16u_{n-3} + 3u_{n-4}}{12h} \tag{102}$$

and the sixth-order central difference scheme

$$\frac{\partial P}{\partial x}|_{x=x_1} = \frac{265u_7 - 478u_6 + 50u_5 + 400u_4 - 450u_3 + 360u_2 - 147u_1}{60h} \tag{103}$$

$$\frac{\partial P}{\partial x}|_{x=x_2} = \frac{-3463u_7 + 13845u_6 - 20740u_5 + 13760u_4 - 3315u_3 - 77u_2 - 10u_1}{60h} \tag{104}$$

$$\frac{\partial P}{\partial x}|_{x=x_3} = \frac{-u_7 + 8u_6 - 30u_5 + 80u_4 - 35u_3 - 24u_2 + 2u_1}{60h} \tag{105}$$

$$\frac{\partial P}{\partial x}|_{x=x_i} = \frac{u_{i+3} - 9u_{i+2} + 45u_{i+1} - 45u_{i-1} + 9u_{i-2} - u_{i-3}}{60h} \tag{106}$$

$$\frac{\partial P}{\partial x}|_{x=x_{n-2}} = \frac{-2u_n + 24u_{n-1} + 35u_{n-2} - 80u_{n-3} + 30u_{n-4} - 8u_{n-5} + u_{n-6}}{60h} \tag{107}$$

$$\frac{\partial P}{\partial x}|_{x=x_{n-1}} = \frac{10u_n + 77u_{n-1} - 150u_{n-2} + 100u_{n-3} - 50u_{n-4} + 15u_{n-5} - 2u_{n-6}}{60h} \tag{108}$$

$$\frac{\partial P}{\partial x}|_{x=x_n} = \frac{147u_n - 360u_{n-1} + 450u_{n-2} - 400u_{n-3} + 225u_{n-4} - 72u_{n-5} + 10u_{n-6}}{60h} \tag{109}$$

For the first-order derivative, the second-order central difference scheme together with second-order biased formula for the boundary points are implemented in block $uxCD2B2$ (Figure 18 shows some portion of the uxCD2B2 block code), whereas the fourth-order central difference scheme with fourth-order biased formula and the sixth-order central difference scheme with sixth-order biased formula are implemented in blocks $uxCD4B4$ (Figure 19 shows some portion of the uxCD4B4 block code) and $uxCD6B6$ (Figure 20 shows some portion of the uxCD6B6 block code) respectively. These three blocks are combined together in *Derivator* block (Figure 21). Which one will be used is specified by the user in the $WorldModel1$ block through the parameter qss: $qss = 1$ for the $uxCD2B2$, $qss = 2$ for $uxCD4B4$ and $qss = 3$ for $uxCD6B6$.

```
if bcl == -1 then
  y[1] = 0;
else
  y[1] = (-u[3] + 4*u[2] - 3*u[1])/(2*deltax);
end if;

for i in 2:n-1 loop
  y[i] = (u[i+1] - u[i-1])/(2*deltax);
end for;

if bcr == -1 then
  y[n] = 0;
else
  y[n] = (3*u[n] - 4*u[n-1] + u[n-2])/(2*deltax);
end if;
```

Figure 18: Portion of uxCD2B2 block code

```
if bcl == -1 then
  y[1] = 0;
  y[2] = (-u[4] + 8*u[3] + u[2] - 8*u[1])/(12*deltax);
else
  y[1] = (-3*u[5] + 16*u[4] - 36*u[3] + 48*u[2] - 25*u[1])/(12*deltax);
  y[2] = (u[5] - 6*u[4] + 18*u[3] - 10*u[2] - 3*u[1])/(12*deltax);
end if;

for i in 3:n-2 loop
  y[i] = (-u[i+2] + 8*u[i+1] - 8*u[i-1] + u[i-2])/(12*deltax);
end for;

if bcr == -1 then
  y[n-1] = (8*u[n] - u[n-1] - 8*u[n-2] + u[n-3])/(12*deltax);
  y[n] = 0;
else
  y[n-1] = (3*u[n] + 10*u[n-1] - 18*u[n-2] + 6*u[n-3] - u[n-4])/(12*deltax);
  y[n] = (25*u[n] - 48*u[n-1] + 36*u[n-2] - 16*u[n-3] + 3*u[n-4])/(12*deltax);
end if;
```

Figure 19: Portion of uxCD4B4 block code

```
if bcl == -1 then
  y[1] = 0;
  y[2] = (u[5] - 9*u[4] + 44*u[3] + 9*u[2] - 45*u[1])/(60*deltax);
  y[3] = (u[6] - 9*u[5] + 45*u[4] - 46*u[2] + 9*u[1])/(60*deltax);
else
  y[1] = (265*u[7] - 478*u[6] + 50*u[5] + 400*u[4] - 450*u[3] + 360*u[2] - 147*u[1])/(60*deltax);
  y[2] = (-3463*u[7] + 13845*u[6] - 20740*u[5] + 13760*u[4] - 3315*u[3] - 77*u[2] - 10*u[1])/(60*deltax);
  y[3] = (-u[7] + 8*u[6] - 30*u[5] + 80*u[4] - 35*u[3] - 24*u[2] + 2*u[1])/(60*deltax);
end if;

for i in 4:n-3 loop
  y[i] = (u[i+3] - 9*u[i+2] + 45*u[i+1] - 45*u[i-1] + 9*u[i-2] - u[i-3])/(60*deltax);
end for;

if bcr == -1 then
  y[n-2] = (-9*u[n] + 46*u[n-1] - 45*u[n-3] + 9*u[n-4] - u[n-5])/(60*deltax);
  y[n-1] = (45*u[n] - 9*u[n-1] - 44*u[n-2] + 9*u[n-3] - u[n-4])/(60*deltax);
  y[n] = 0;
else
  y[n-2] = (-2*u[n] + 24*u[n-1] + 35*u[n-2] - 80*u[n-3] + 30*u[n-4] - 8*u[n-5] + u[n-6])/(60*deltax);
  y[n-1] = (10*u[n] + 77*u[n-1] - 150*u[n-2] + 100*u[n-3] - 50*u[n-4] + 15*u[n-5] - 2*u[n-6])/(60*deltax);
  y[n] = (147*u[n] - 360*u[n-1] + 450*u[n-2] - 400*u[n-3] + 225*u[n-4] - 72*u[n-5] + 10*u[n-6])/(60*deltax);
end if;
```

Figure 20: Portion of uxCD6B6 block code

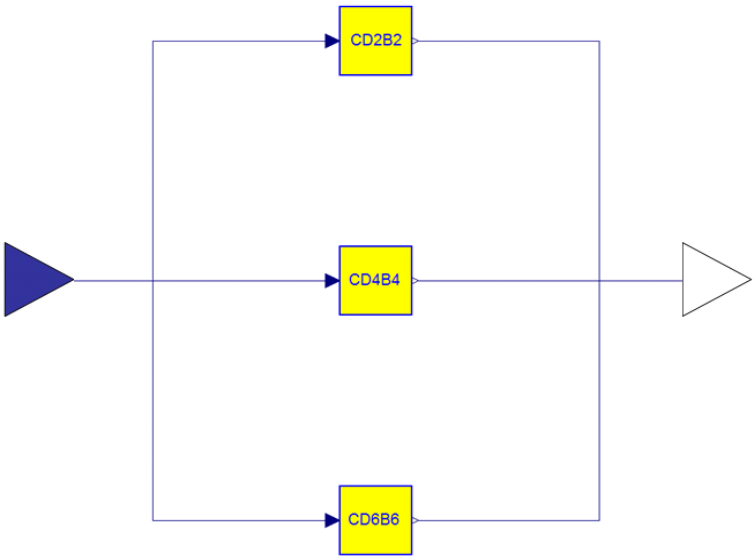

Figure 21: Derivator block

4.4 Solving ODEs

The system of ODEs we obtained above for the heat diffusion problem can be written in a more compact form:

$$\dot{\mathbf{x}} = \mathbf{A} \cdot x + \mathbf{b} \cdot u \tag{110}$$

where

$$A = \frac{n^2}{10\pi^2} \begin{pmatrix} -2 & 1 & 0 & 0 & \dots & 0 & 0 & 0 \\ 1 & -2 & 1 & 0 & \dots & 0 & 0 & 0 \\ 0 & 1 & -2 & 1 & \dots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 1 & -2 & 1 \\ 0 & 0 & 0 & 0 & \dots & 0 & 2 & -2 \end{pmatrix} \tag{111}$$

$$b \cdot u = \begin{pmatrix} \exp(\frac{-t}{10}) \\ 0 \\ \vdots \\ 0 \end{pmatrix} \tag{112}$$

The matrix A is a band-structured matrix and its eigenvalues are real and strictly negative. In general, all parabolic PDEs converted to a set of ODEs by using the method of lines have this property. If we consider the $stiffness$ ratio, defined as

$$Stiffness = \frac{max|e_i|}{min|e_j|} \tag{113}$$

then it turns out that this ratio depends on the number of segments, more precisely it grows quadratically with the number of segments. *The more accurate we wish to solve the diffusion equation, the stiffer the corresponding ODE problem will become. Since the diffusion problems are usually quite smooth, the BDF algorithms are optimal for the simulation of the resulting set of ODEs.* [1]
How can we solve the system of ODEs just obtained? Well, there is a reason why we transformed the heat PDE into a set of ODEs: Dymola provides many methods to solve ODEs. For instance, I used the DASSL method to solve all of the examples of PDEs in this thesis. DASSL implements the backward difference formula (BDF) of orders one to five and is the default simulation algorithm in Dymola. For more details about DASSL see [1].

4.5 Boundary Conditions

Generally a PDE has many solutions and some additional information must be specified to define the solution uniquely. Besides the initial condition, boundary conditions are

needed. There are many possibilities to specify boundary conditions:

- **Dirichlet**

  The solution u of the function at the boundary point is specified.

- **Neumann**

  The derivative du/dx of the function at the boundary point is specified.

- **Robin**

  a combination of the solution and derivative values of the function u at the boundary point is specified.

The inputs *BL* and *BR* in the integrator block accept either constant values or time dependent values. Both can be specified by the source block in $Modelica- > Blocks- > Sources$ (see Examples). The Neumann type of boundary condition can be specified directly in the derivative block. Setting *bcl* or *bcr* to -1 activates the Neumann boundary condition $du/dx = 0$ at the left or at the right part of the domain respectively. Of course the Package does not offer a complete treatment of boundary conditions. Some problems would for example need boundary conditions that "appear" or "disappear". This boundary condition, called nonlinear boundary condition, says that at some time we either have a boundary condition at the corresponding grid point or not. If we have one, then its value is specified by the boundary condition, if not, we have an equation for this grid point. Depending on the method, we need special formulas for either one or more grid points at each end of the domain. So the boundary grid points are treated in a special way. In the FVM Chapter, we will see another way of implementing the boundary conditions that does not require this special treatment.

4.6 MOL Implementation in Modelica

As mentioned in Chapter 2, one of the main goals of the thesis was to provide Modelica with basic blocks that enable the construction of the PDEs along with the numerical solution. The kind of the numerical method used should be transparent to the user. In the present thesis two numerical methods are implemented: Method of Lines and finite volume method. This chapter is only concerned with the method of lines. In order to solve a PDE with the method of lines, the Integrator block was built (Figure 22). This block could be seen, as many others, as a black box that takes some inputs and provides outputs. What is happening inside the box must not concern the user. When constructing some specific PDE, we must connect correctly the inputs with outputs and initialize the block parameters. If everything is done correctly, we can start with the simulation. In Figure 23 some portion of the integrator block code is shown. As can be seen, the boundary conditions are checked and initialized, the initial conditions are set in the "initial equation" section. Under "equation" the values at the grid points at each

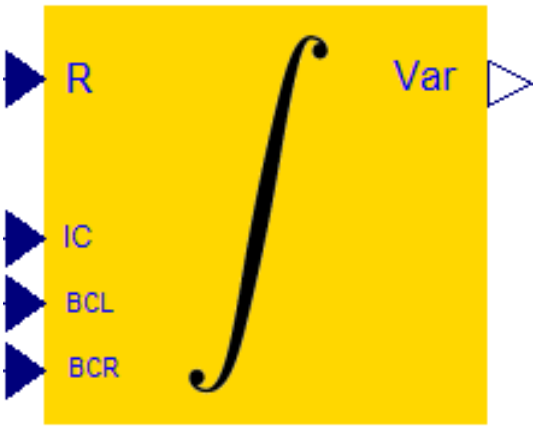

Figure 22: MOL Integrator

```
Real f[n];

equation
  y = f;

  if bcl == 1 then
    f[1] = u2;
  end if;
  if bcr == 1 then
    f[n] = u3;
  end if;

  for i in vb:ve loop
    der(f[i]) = u[i];
  end for;

initial equation

  for i in icb:ice loop
    f[i] = u1[i];
  end for;
```

Figure 23: Portion of integrator block code

time step are computed.
The integrator block computes the equations of the form

$$u_t = f(u, u_x, u_{xx}, ...) \tag{114}$$

If the equation has another form, we must reduce it to this form before giving it to the integrator. The vector of grid points is given as output by the integrator. We can use it to construct the right part of the equation and at the end give the whole to the R input of the integrator (see for example the heat equation in Figure 24). We will now see many examples of how to use these blocks in order to construct PDEs.

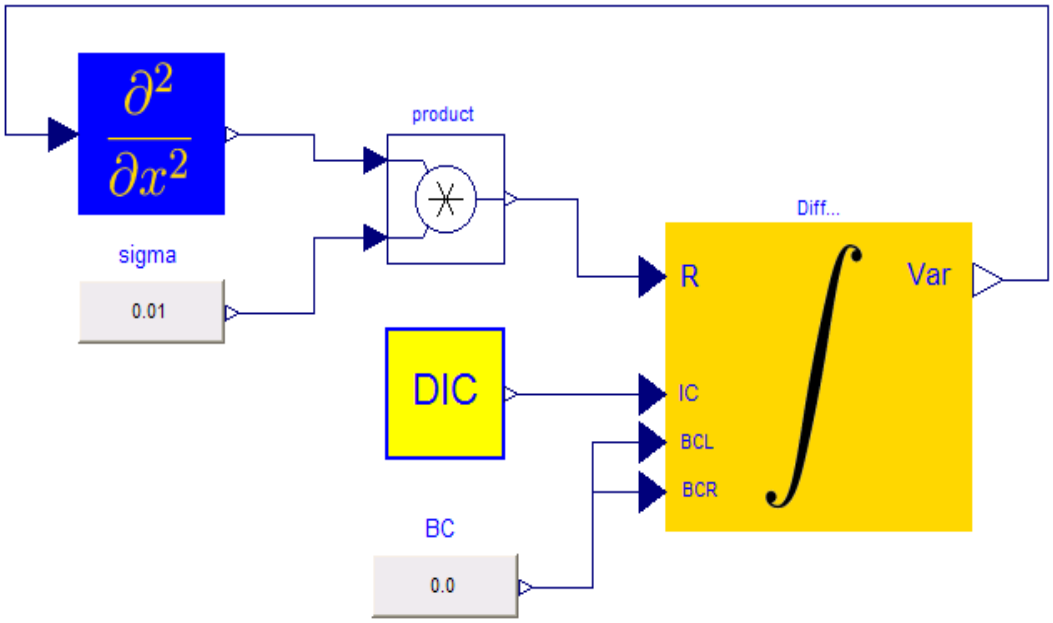

Figure 24: MOL construction of the heat equation in Modelica

4.7 Examples

In the previous chapter we illustrated how to implement the heat equation with the method of lines. Many other examples were implemented and they can be found under $PDE \rightarrow MOL \rightarrow Examples$. In the following we shall illustrate some of them. Let us start with the advection equation.

4.7.1 Advection Equation

The advection equation is one of the most simple PDEs. Nevertheless, it plays an important role as we will see in the Finite Volume Method chapter. Given a speed c, the

advection equation has the form

$$\frac{\partial u}{\partial t} = -c\frac{\partial u}{\partial x} \tag{115}$$

If we have as initial condition some function $u(x, 0)$, then the advection equation will just shift this function with the speed c without changing its shape. Thus the solution of the advection equation is

$$u(x - ct, t)$$

In Figure 25 the MOL implementation of the advection equation with speed $c = 0.1$ can be seen.

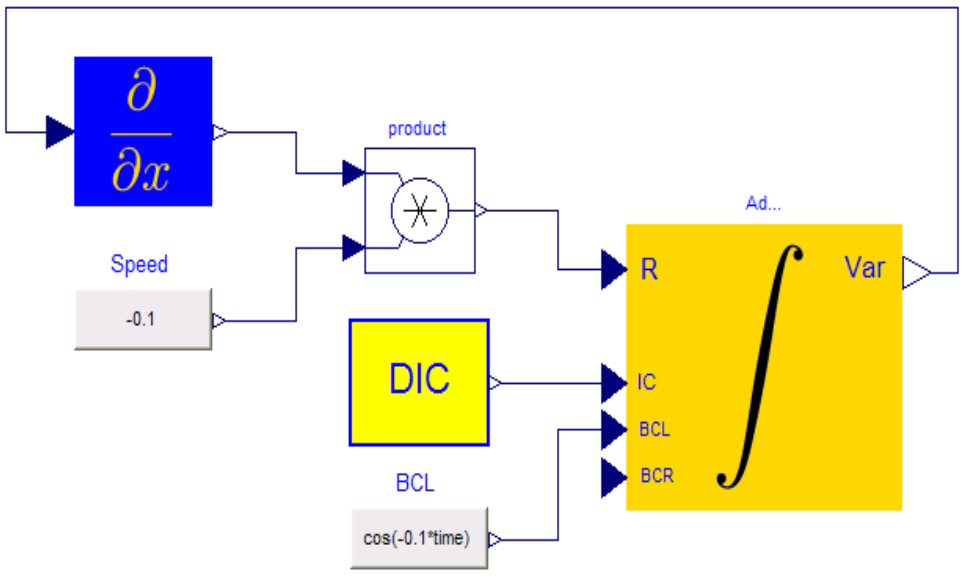

Figure 25: Advection equation with MOL

The initial and boundary conditions that were used are:

Initial condition: $cos(x)$

Boundary condition: $cos(-ct)$

Note that only one boundary condition suffices (the one that is upwind). What is amazing is that already with only ten cells we have a very good approximation (Figure 26). The greatest error values are reached in the seventh cell (Figure 27). The maximum error is of order 10^{-3}.

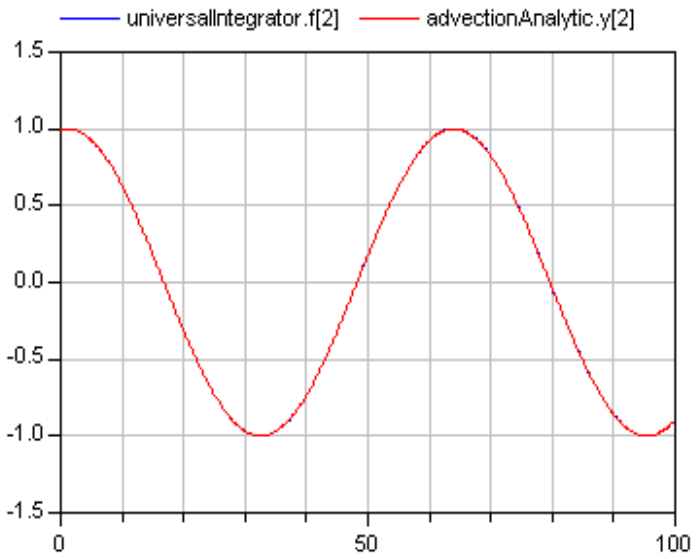

Figure 26: Advection equation: numerical versus analytical solution (10 grid points were used)

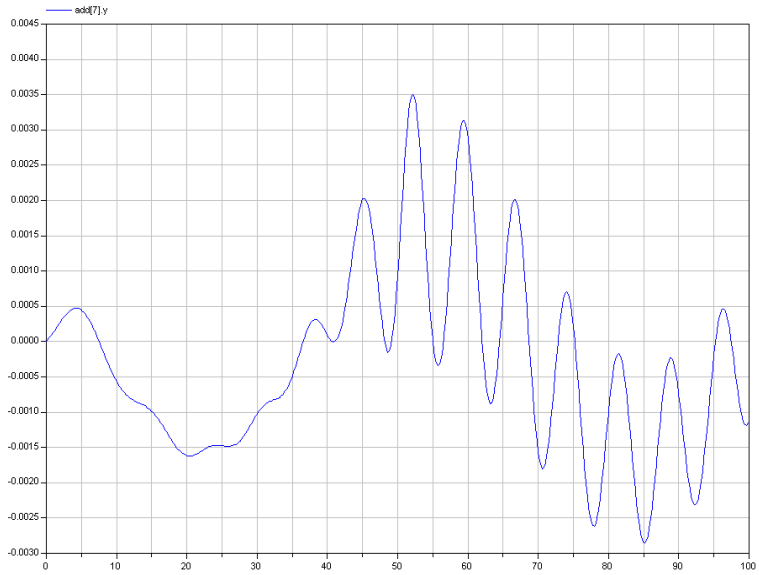

Figure 27: Advection equation: error of the 7-th grid point (10 grid points were used)

4.7.2 Wave Equation

The wave equation is the hyperbolic PDE

$$\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2} \tag{116}$$

At first glance it seems that we cannot solve this equation with our *integrator* block. We can however transform this second-order PDE into two first-order PDEs:

$$\frac{\partial u}{\partial t} = v \tag{117}$$

$$\frac{\partial v}{\partial t} = c^2 \frac{\partial^2 u}{\partial x^2} \tag{118}$$

Now we have reached the form that is appropriate for our *integrator* block. The complete problem, that is implemented in Modelica, together with the initial and boundary conditions is listed below:

$$\frac{\partial u}{\partial t} = v \tag{119}$$

$$\frac{\partial v}{\partial t} = \frac{\partial^2 u}{\partial x^2} \tag{120}$$

$$u(x,0) = sin(\frac{\pi}{2}x) \tag{121}$$

$$\frac{\partial u}{\partial t}(x,0) = 0.0 \tag{122}$$

$$u(x_1,t) = 0.0 \tag{123}$$

$$\frac{\partial u}{\partial x}(x_n,t) = 0.0 \tag{124}$$

The process of solving this system of equations is the same as described above, we start with discretizing the spatial derivative and in the end we reach the set of ODEs:

$$u_1 = 0.0 \tag{125}$$

$$\dot{u}_2 = v_2 \tag{126}$$

$$\dot{u}_3 = v_3 \tag{127}$$

$$... \tag{128}$$

$$\dot{u}_n = v_n \tag{129}$$

$$v_1 = 0.0 \tag{130}$$

$$\dot{v}_2 = n^2(u_3 - 2u_2 + u_1) \tag{131}$$

$$\dot{v}_3 = n^2(u_4 - 2u_3 + u_2) \tag{132}$$

$$... \tag{133}$$

$$\dot{v}_n = n^2(u_{n-1} - u_n) \tag{134}$$

and the initial conditions are

$$u_2 = sin(\frac{\pi}{2n}) \tag{135}$$

$$u_3 = sin(\frac{\pi}{n}) \tag{136}$$

$$... \tag{137}$$

$$u_n = sin(\frac{(n-1)\pi}{2n}) \tag{138}$$

$$v_2 = 0.0 \tag{139}$$

$$v_3 = 0.0 \tag{140}$$

$$... \tag{141}$$

$$v_n = 0.0 \tag{142}$$

In Figure 28 we see how the wave equation is implemented in Modelica. Figure 29 shows the implementation of the initial condition for the first equation. The good news are that for this problem there is an analytical solution

$$u(x,t) = \frac{1}{2}sin(\frac{\pi(x-t))}{2}) + \frac{1}{2}sin(\frac{\pi(x+t))}{2}) \tag{143}$$

The analytical solution is implemented in Modelica too. In Figure 30 the graphs of the two first and two last grid points are shown. The analytical solution can be seen in red and the solution provided by the method of lines in blue. As can be seen, if we discretize the domain with only ten grid points, we have already a good approximation (error of order 10^{-2}).

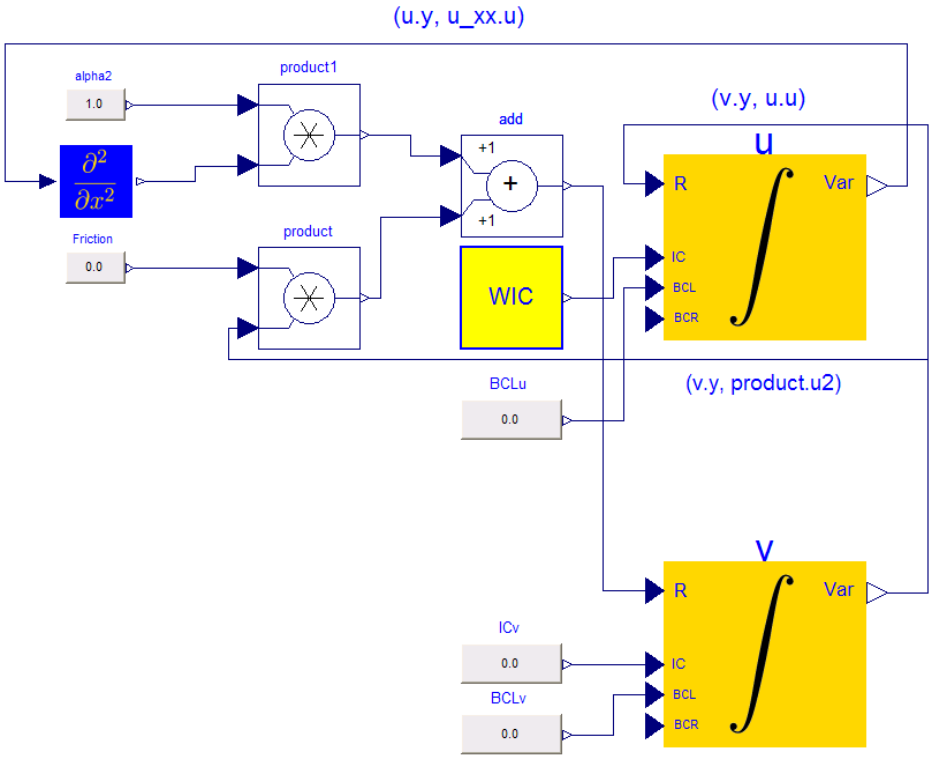

Figure 28: Wave equation in Modelica

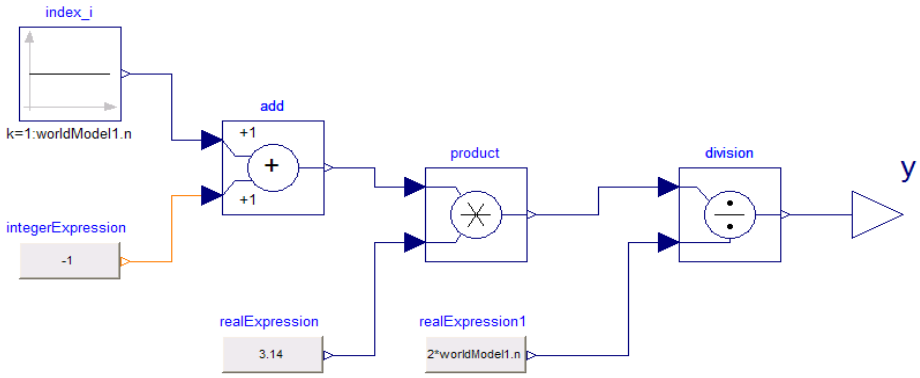

Figure 29: Initial conditions

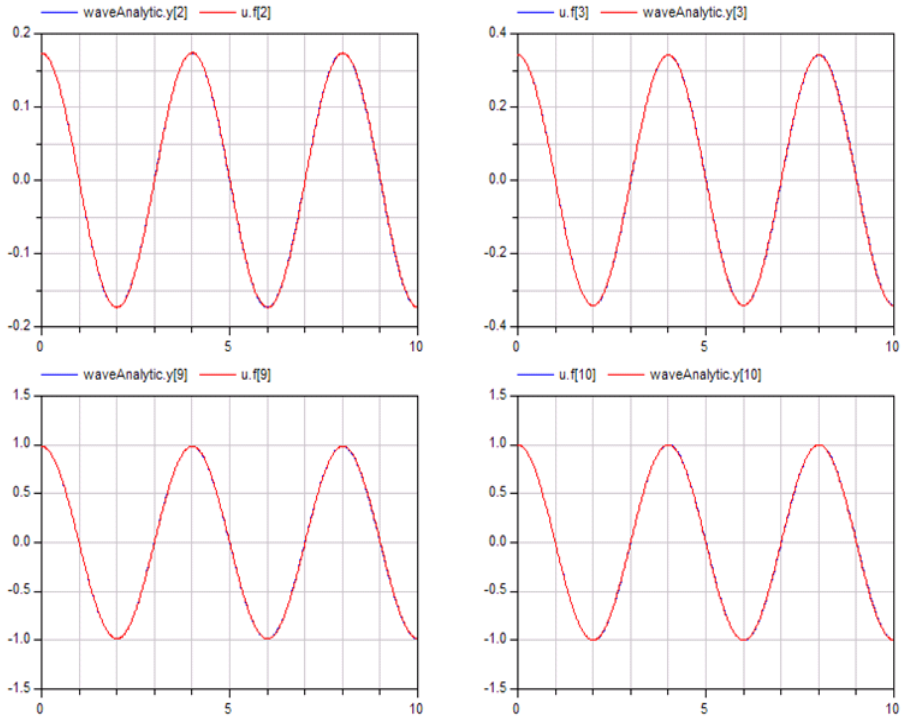

Figure 30: Comparison of the analytical and numerical solution of the wave equation. The solution of the first two and last two grid points is shown (10 grid points were used).

4.7.3 Vibrating String Equation

Another interesting problem is that of the vibrating string. The finite vibrating string of length L is described by the wave equation together with special boundary and initial conditions. The complete problem is

$$\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2} \tag{144}$$

$$u(0,t) = 0 \tag{145}$$

$$u(L,t) = 0 \tag{146}$$

$$u(x,0) = sin(\frac{\pi x}{L}) + \frac{1}{2} sin(\frac{3\pi x}{L}) \tag{147}$$

$$u_t(x,0) = 0 \tag{148}$$

The MOL implementation of this problem can be seen in Figure 31. The analytic solu-

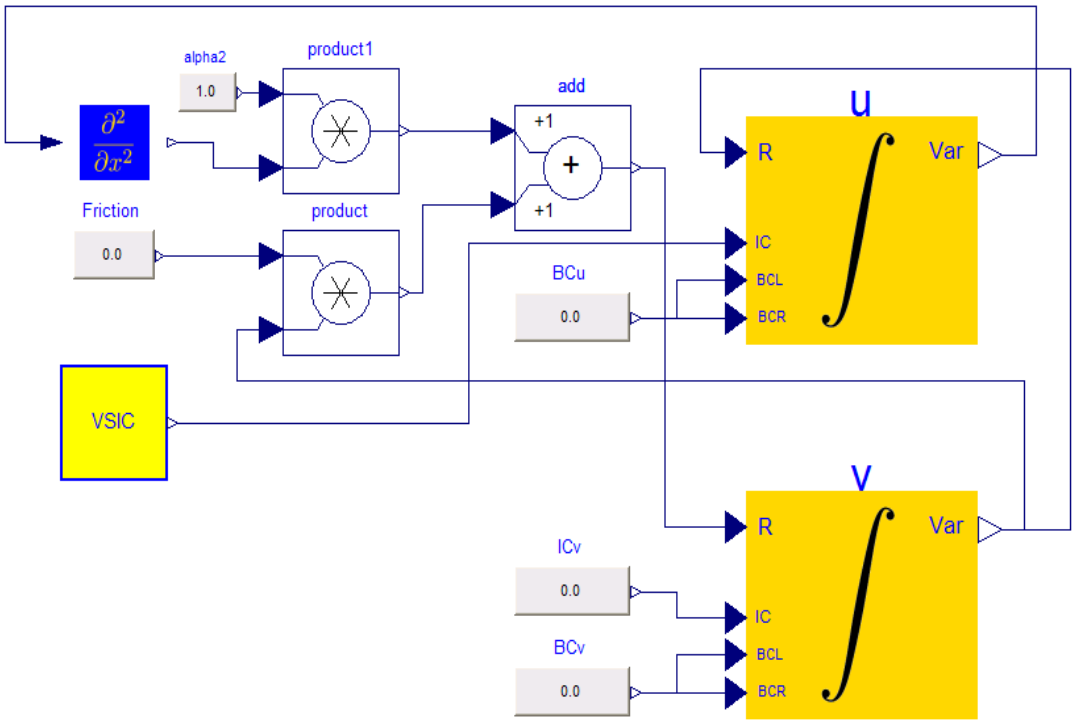

Figure 31: Vibrating string equation in Modelica

tion to this problem is given by

$$u(x,t) = sin(\frac{\pi x}{L})cos(\frac{\pi ct}{L}) + \frac{1}{2}sin(\frac{3\pi x}{L})cos(\frac{3\pi ct}{L}) \tag{149}$$

Figure 32 shows the numerical together with analytical solution for the second grid point and the point in the middle of the domain. The approximation in not satisfactory enough. With 40 cells (Figure 32 below) better results are achieved. The error of the solution for the second grid point with 10 and 40 cells can be seen in Figure 32. As can be seen from graphs, the error decreases dramatically if we increase the number of cells from 10 to 40.

4.7.4 Shock Wave Equation

Let us now consider a more complicated example taken from [1]: Euler equations. Problem description:

A thin tube is located at sea level, i.e, the surrounding atmosphere has a pressure of $p_0 = 1$ *atm. The current temperature is* $T = 300$ *K. At time zero, the tube is opened at one of its two ends. We wish to determine the pressure, velocity and density at various*

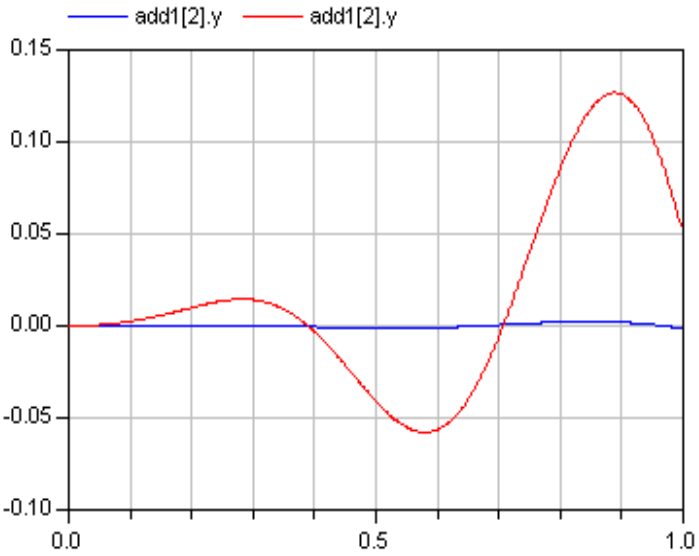

Figure 32: Vibrating string equation: Comparison of errors for the second grid point with 10 and 40 grid points

places inside the tube as functions of time.

The problem is described by the following system of PDEs:

$$\frac{\partial \rho}{\partial t} = -v \cdot \frac{\partial \rho}{\partial x} - \rho \cdot \frac{\partial v}{\partial x} \tag{150}$$

$$\frac{\partial v}{\partial t} = -v \cdot \frac{\partial v}{\partial x} - \frac{a}{\rho} \tag{151}$$

$$\frac{\partial p}{\partial t} = -v \cdot a - \gamma \cdot p \cdot \frac{\partial v}{\partial x} \tag{152}$$

$$a = \frac{\partial p}{\partial x} + \frac{\partial q}{\partial x} + f \tag{153}$$

$$q = \begin{cases} \beta \cdot \delta x^2 \cdot \rho \cdot (\frac{\partial v}{\partial x})^2 & if \frac{\partial v}{\partial x} < 0.0 \\ 0.0 & if \frac{\partial v}{\partial x} \geq 0.0 \end{cases} \tag{154}$$

$$f = \frac{\alpha \cdot \rho \cdot v \cdot |v|}{\delta x} \tag{155}$$

where

- ρ: gas density

- v: gas velocity
- p: gas pressure
- q: pseudo viscous pressure
- f: frictional resistance
- γ: ratio of specific heat constants ($\frac{c_p}{c_v} = 1.4$)
- α, β: fudge factors

The initial conditions are

$$\rho(x, 0) = \rho_i \tag{156}$$

$$v(x, 0) = 0.0 \tag{157}$$

$$p(x, 0) = p_i \tag{158}$$

where ρ_i is determined by the equation of state for ideal gases:

$$\rho_i = \frac{p_i M_{air}}{RT} \tag{159}$$

where $T = 300K$, $R = 8.314\frac{J}{Kmole}$ and $M_{air} = 28.96\frac{g}{mole}$.
and the boundary conditions are

$$v(0, t) = 0.0 \tag{160}$$

$$\rho(1, t) = \rho_0 \tag{161}$$

$$\begin{cases} v(1,t) = -\sqrt{\frac{2(p_0 - p(1,t))}{\rho(1,t)}}, & if v(1,t) < 0.0 \\ p(1,t) = p_0, & if v(1,t) \geq 0.0 \end{cases} \tag{162}$$

Implementing this kind of boundary condition involve checking at each time the condition $v(1, t) < 0.0$ and changing accordingly the boundary conditions. This would require together for each boundary condition input in the integrator block, the boolean input, that tells whether the condition is satisfied or not. Depending on this condition the boundary condition input will receive either a value or the equation. For this reason we will use, for simplicity, the boundary condition

$$p(1, t) = p_0 \tag{163}$$

The implementation of the Euler equations is shown in Figure 33. Here we used as initial conditions

$$\rho_0 = 1.0$$
$$v_0 = 0.0$$
$$p_0 = 1.0$$

and boundary conditions

$$\rho(x_n, t) = 0.125$$
$$v(x_1, t) = 0.0$$
$$p(x_n, t) = 0.1$$

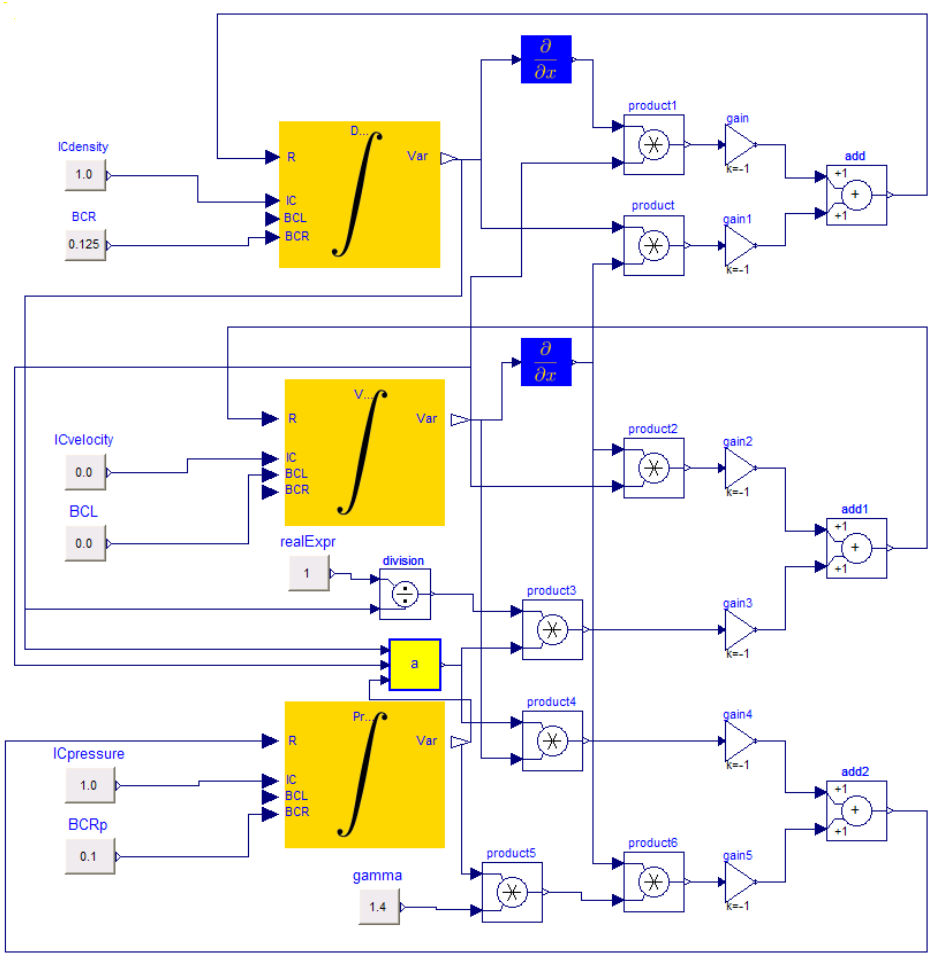

Figure 33: Shock wave equation in Modelica

The pseudo viscous pressure and the frictional resistance are implemented in v and f blocks respectively (Figure 34 and 35) The viscosity together with the friction term are integrated into the a block (Figure 36). The analytical solution to this problem does not exist. In Figure 37 the numerical solution of the density, velocity and pressure in the second and middle grid point, where 10 grid points were used, is shown.

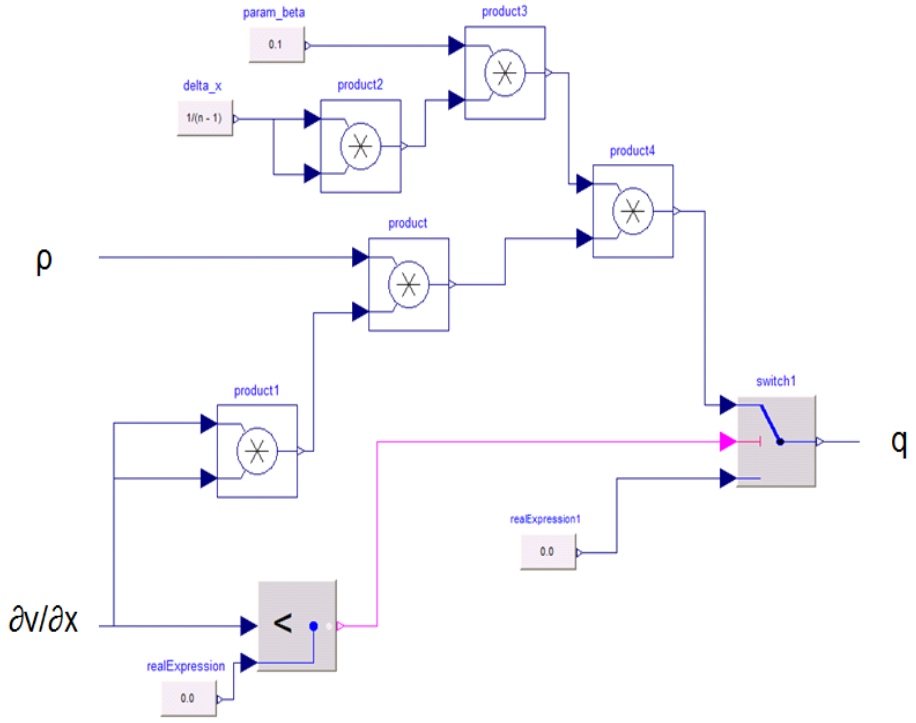

Figure 34: Pseudo viscous pressure block

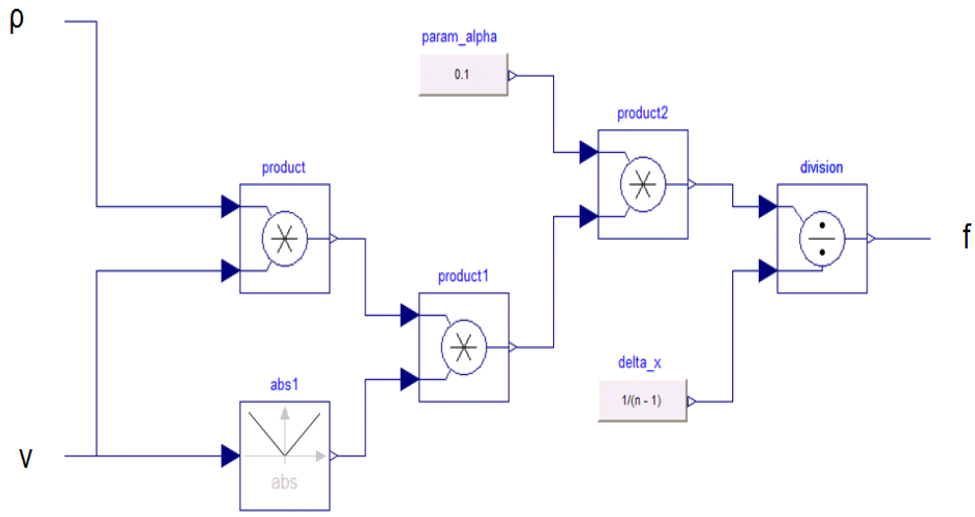

Figure 35: Frictional resistance block

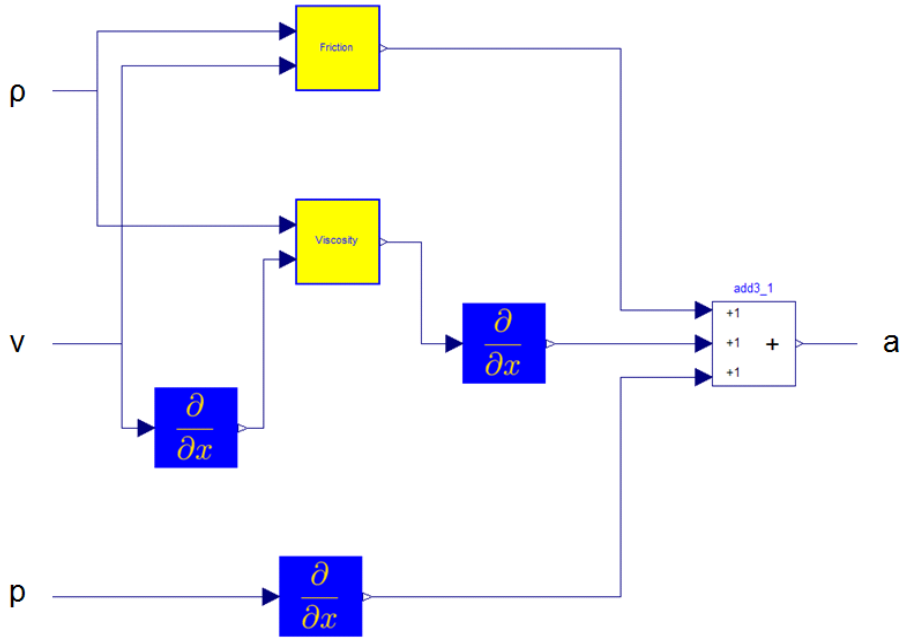

Figure 36: *a* block

4.7.5 Diffusion Equation

We have already seen how to construct the diffusion equation in Modelica. In the following we will consider the diffusion equation with the initial condition

$$u(x,0) = sin(\pi x) + \frac{1}{2} sin(3\pi x) \tag{164}$$

and boundary conditions

$$u(x_1,t) = 0.0 \tag{165}$$

$$u(x_n,t) = 0.0 \tag{166}$$

The diffusion equation with this initial and boundary conditions has the analytical solution

$$u(x,t) = e^{-(\pi\sqrt{\sigma})^2 t} sin(\pi x) + \frac{1}{2} e^{-(3\pi\sqrt{\sigma})^2 t} sin(3\pi x) \tag{167}$$

In Figure 38 the numerical solution is compared with the analytical solution. The parameter σ is set to 0.1.

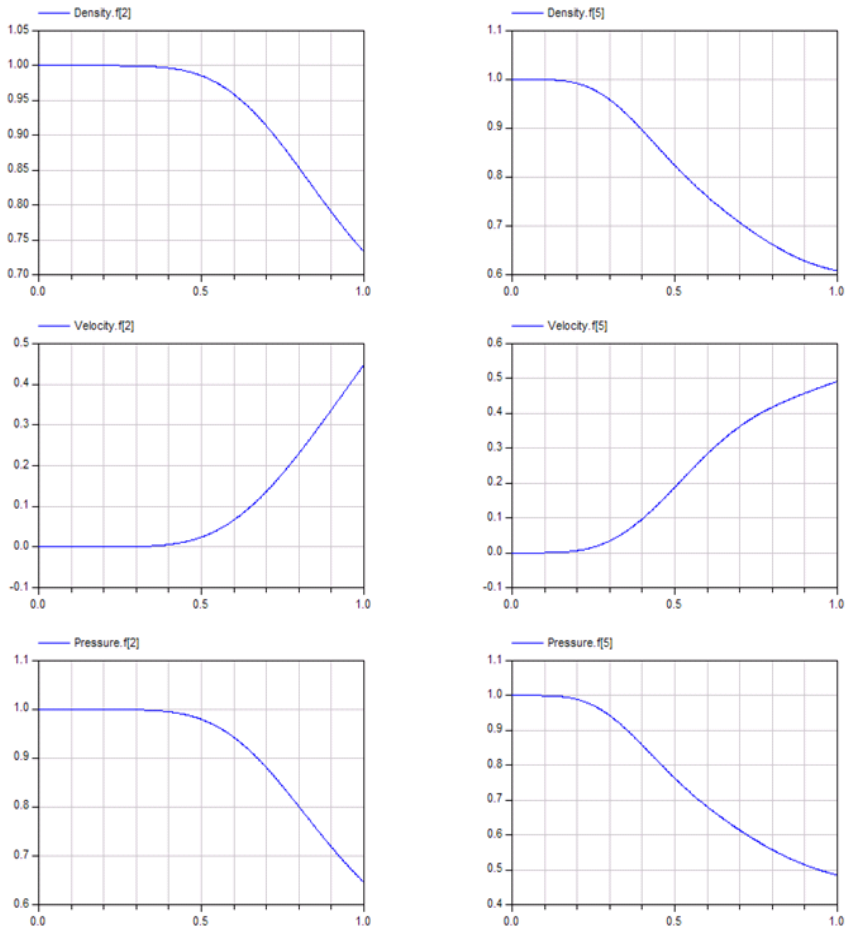

Figure 37: Numerical solution for density, velocity and pressure to the second and middle grid points (10 grid points were used)

4.7.6 Transmission Line Equation

Transmission line equation is a PDE

$$u_{tt} = c^2 u_{xx} - h u_t - k u \tag{168}$$

The name derives from the equation that describes the electric current along a wire

$$CLi_{tt} = i_{xx} - (CR + GL)i_t - GRi \tag{169}$$

where

- C = capacitance/unit length of wire

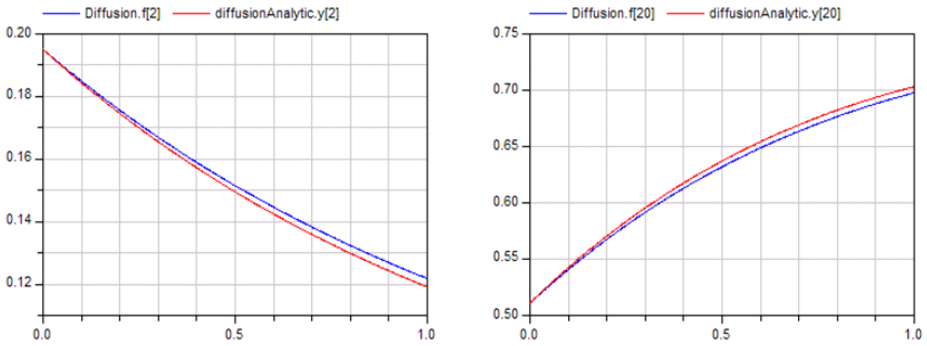

Figure 38: Numerical (blue) and analytical (red) solution of diffusion equation for the second and middle grid points (40 grid points were used)

- L = self-inductance/unit length of wire
- R = resistance/unit length of wire
- G = leakage conductance/unit length of wire
- current along the wire

The Modelica implementation of the transmission line equation can be seen in Figure 39. As already seen for the wave equation, transmission line equation was converted into two ODEs:

$$u_t = v \tag{170}$$

$$v_t = c^2 u_{xx} - h u_t - k u \tag{171}$$

The following initial and boundary conditions were used

$$u(x, 0) = sin(\pi x) \tag{172}$$

$$\frac{\partial u}{\partial t} = 0 \tag{173}$$

$$u(x_1, t) = 0.0 \tag{174}$$

$$u(x_n, t) = 0.0 \tag{175}$$

The analytical solution is not implemented. In Figure 40 the solution for the grid point in the middle of the domain with 10 and 40 cells is shown.

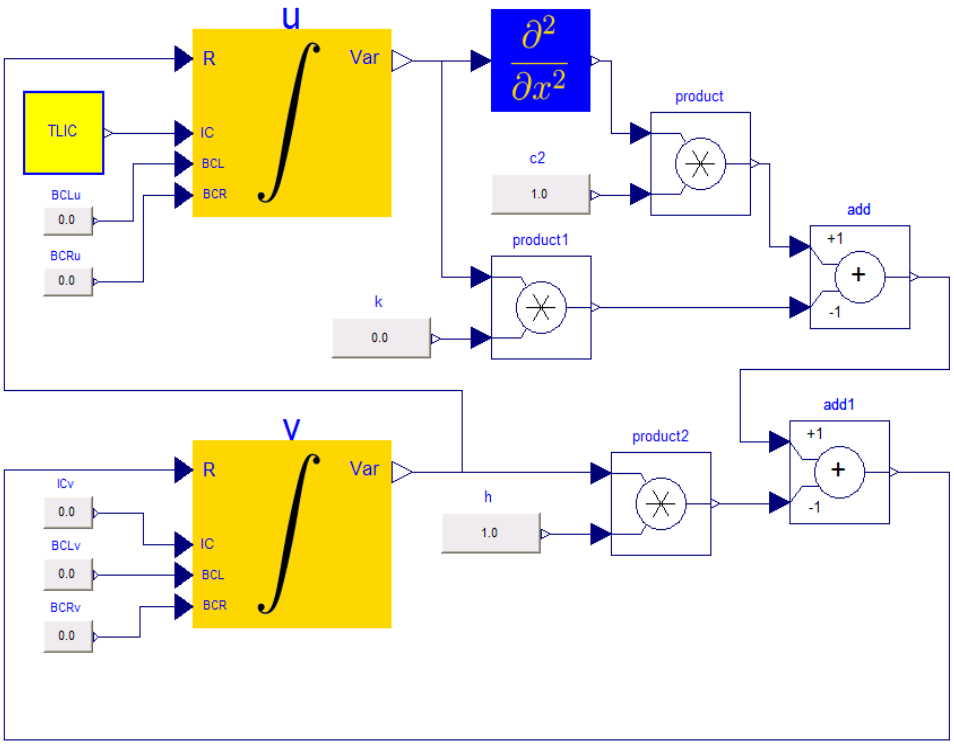

Figure 39: Transmission line equation in Modelica

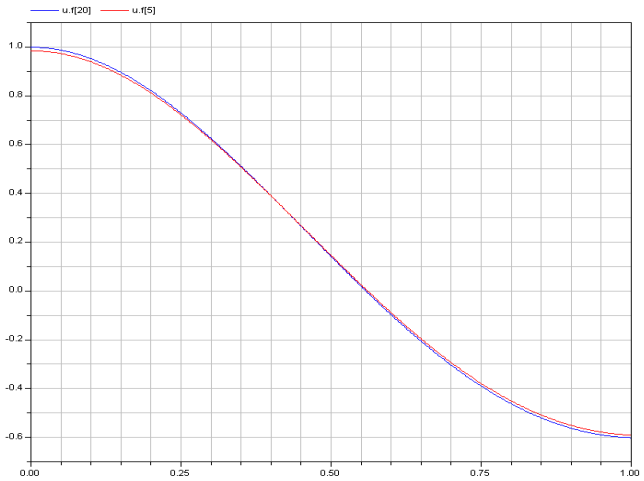

Figure 40: Transmission line equation: Solution for the grid point in the middle of the domain with 10 (blue) and 40 (red) grid points.

4.7.7 Burger´s Equation

The Burger equation is a nonlinear PDE

$$u_t + (\frac{u^2}{2})_x = \epsilon u_{xx} \tag{176}$$

If we neglect the viscosity we obtain the *inviscid* form of the Burger equation

$$u_t + (\frac{u^2}{2})_x = 0 \tag{177}$$

The complete problem together with initial and boundary conditions is

$$u_t + (\frac{u^2}{2})_x = 0 \tag{178}$$

$$\begin{aligned} \mathrm{u(x,\ 0)} &= \mathrm{x} \\ \mathrm{u}(x_1,\ \mathrm{t}) &= 0 \\ \mathrm{u}(x_n,\ \mathrm{t}) &= 0 \end{aligned}$$

The analytical solution of this problem is

$$u(x,t) = \frac{x}{1+t} \tag{179}$$

The MOL implementation of the Burger equation is shown in Figure 41. In Figure 42

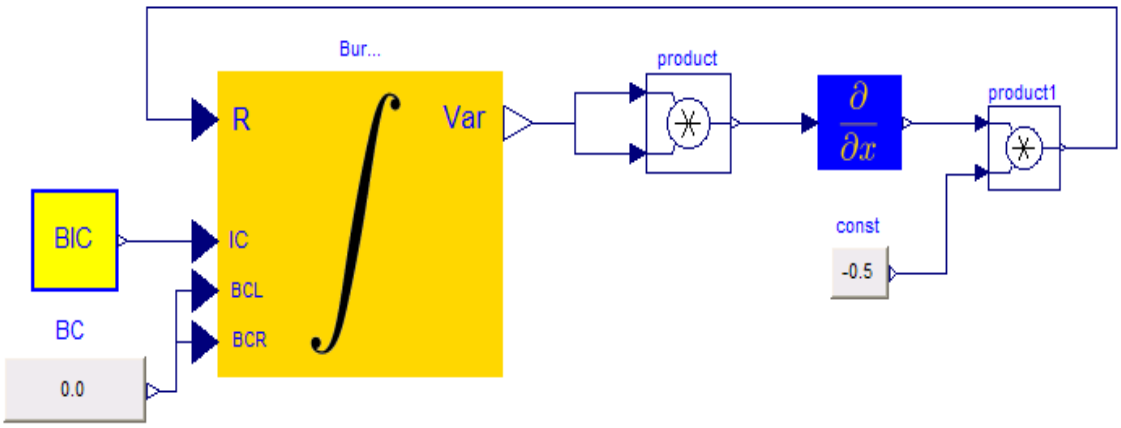

Figure 41: Burger´s equation in Modelica

the comparison of the analytical and numerical solution is shown. Already with 10 cells we have satisfactory results. The biggest error is of order 10^{-4}.

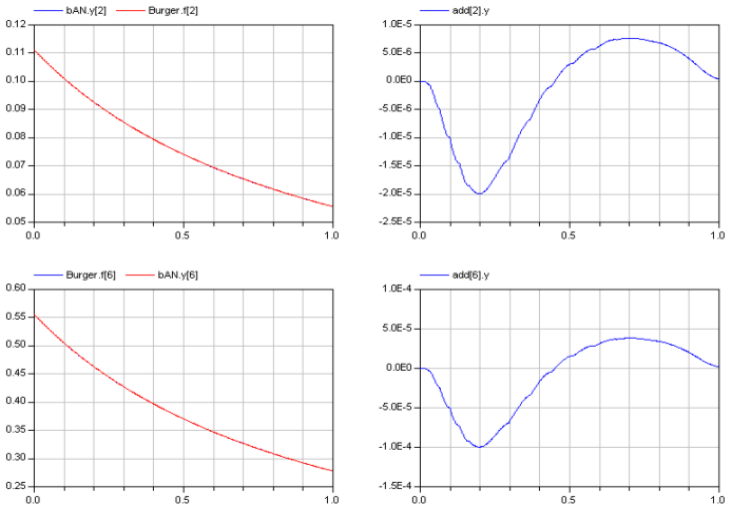

Figure 42: Burger´s equation: analytical vs numerical solution for second and sixth grid points, together with their respective errors (10 grid points were used)

4.7.8 Buckley-Leverett Equation

Buckley-Leverett equation is a nonlinear PDE

$$u_t + (\frac{4u^2}{4u^2 + (1-u)^2})_x = 0 \tag{180}$$

This equation can be found in $PDE- > MOL- > Examples$. In Figure 43 MOL implementation of the Buckley-Lever equation is shown.

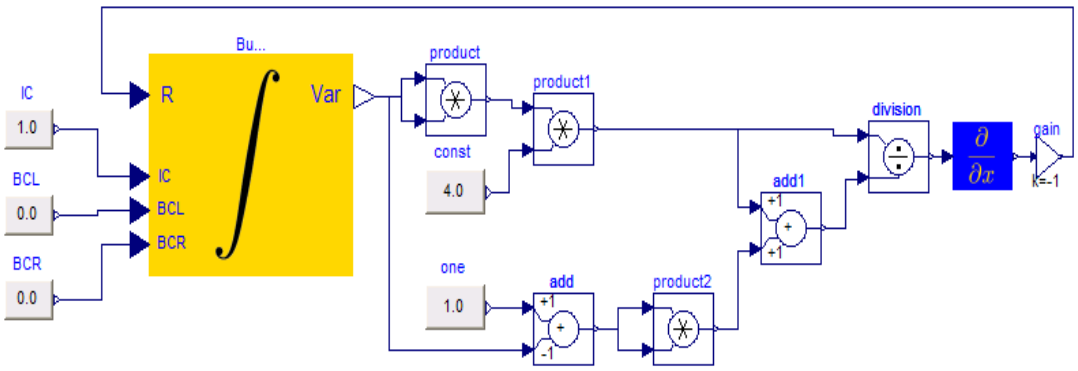

Figure 43: MOL implementation of the Buckley-Leverett equation

The initial condition used is

$$u(x, 0) = 1.0 \tag{181}$$

and boundary conditions are

$$u(x_1, t) = 0.0 \tag{182}$$

$$u(x_n, t) = 0.0 \tag{183}$$

In Figure 44 we can see the solution of Buckley-Leverett equation for the second and middle grid point respectively, where ten grid points were used.

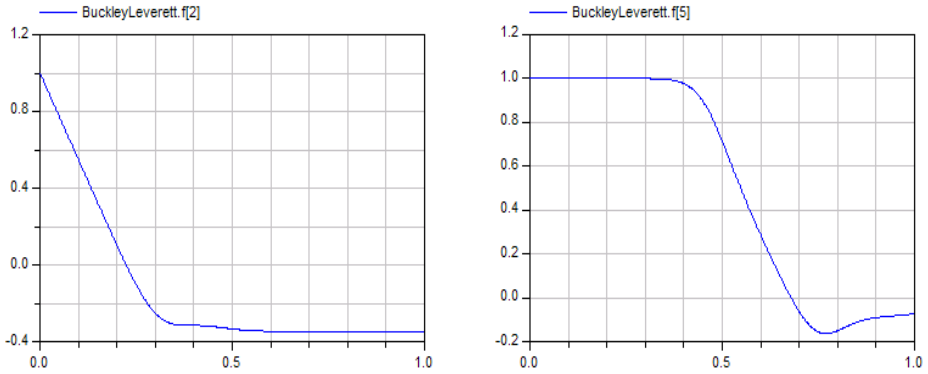

Figure 44: Numerical solution of the Buckley Leverett equation for the first (left) and middle (right) grid points (10 grid points were used)

4.7.9 Simple Supported Beam Equation

Simple supported beam PDE is an interesting problem because it involves the fourth space derivative of the unknown function:

$$u_{tt} + u_{xxxx} = 0 \tag{184}$$

The initial conditions are

$$u(x, 0) = sin(\pi x) + \frac{1}{2} sin(3\pi x) \tag{185}$$

$$u_t(x, 0) = 0 \tag{186}$$

and boundary conditions are

$$u(x_1, t) = 0 \tag{187}$$

$$u_{xx}(x_1, t) = 0 \tag{188}$$

$$u(x_n, t) = 0 \tag{189}$$

$$u_{xx}(x_n, t) = 0 \tag{190}$$

To solve the beam problem with the PDE Package, we must transform it in another

form. Through the variable transformation $v = u_t$ and $w = u_{xx}$ we reach the system of equations:

$$v_t = -w_{xx} \tag{191}$$

$$w_t = v_{xx} \tag{192}$$

The complete problem is implemented in $PDE \rightarrow MOL \rightarrow Examples$ (Figure 45). The

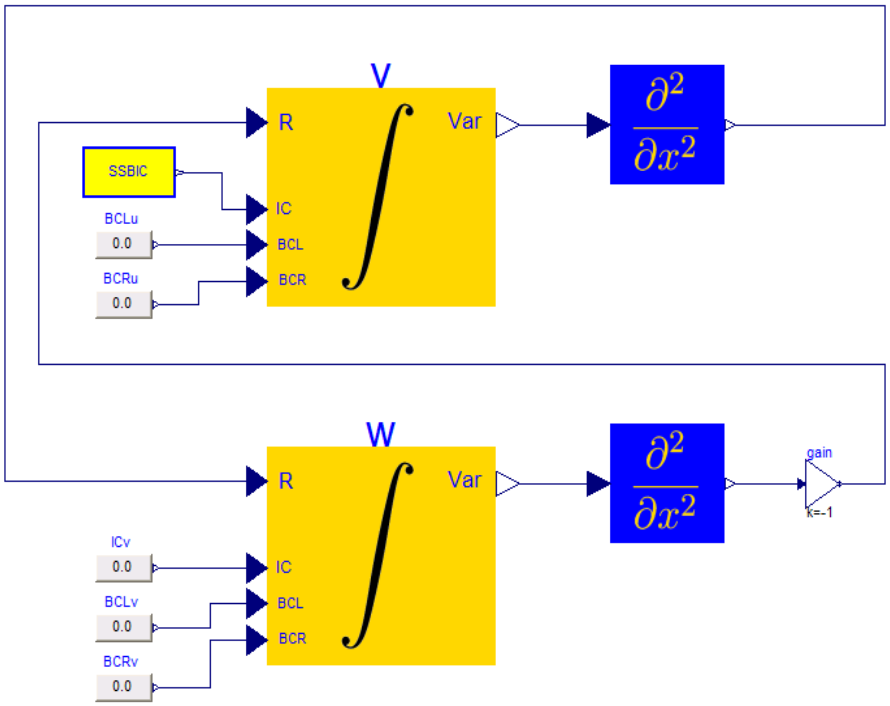

Figure 45: MOL implementation of the simple supported beam problem in Modelica

analytical solution to this problem is also implemented and is

$$u(x,t) = cos(\pi^2 t)sin(\pi x) + \frac{1}{2}cos(9\pi^2 t)sin(3\pi x) \tag{193}$$

The numerical solution for the first and middle grid points, with 10 and 60 grid points respectively, can be seen in Figure 46. As can be seen, ten grid points give a very poor approximation. To reach good approximations we need to partition the domain with at least 60 grid points. However, small oscillations at the pics still remain. Higher order discretization schemes, such as the fourth-order central difference approximation, do not help.

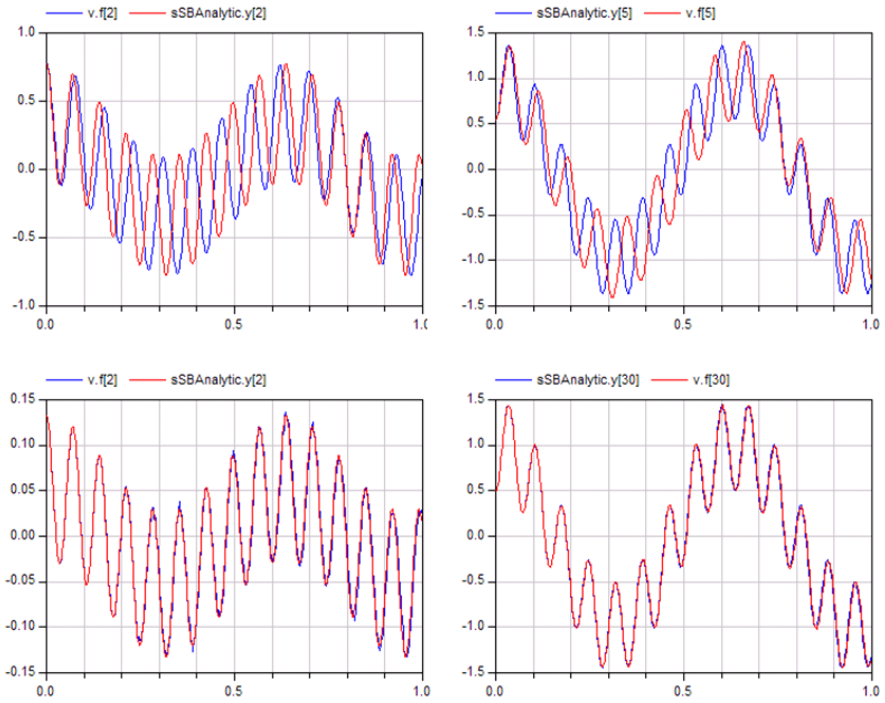

Figure 46: Numerical versus analytical solution of the simple supported beam problem for the second (left) and middle (right) grid points with 10 (above) and 60 (below) grid points

4.7.10 Poisson Equation

In the following we consider the problem

$$u_{xx} = -sin(\pi x) \tag{194}$$

$$u(x_1) = 0 \tag{195}$$

$$u(x_n) = 0 \tag{196}$$

This is the Poisson problem. Poisson problem is a time independent PDE, which means that the solution to this problem does not change with time. For instance, the solution to the above problem is

$$u(x) = \frac{1}{\pi^2} sin(\pi x) \tag{197}$$

Until now we considered only PDEs that are time dependent. But how can we implement time independent PDEs such as the Poisson problem above? Is it possible still to use

the integrator block? The answer is yes, but we must transform our time independent problem in a time dependent problem that reach with time a steady state corresponding to the solution of the time independent problem. The time dependent version of the problem above is

$$u_t = u_{xx} + sin(\pi x) \tag{198}$$

$$u(x_1, t) = 0 \tag{199}$$

$$u(x_n, t) = 0 \tag{200}$$

and additionally we have now the initial condition

$$u(x, 0) = sin(3\pi x) \tag{201}$$

This can be interpreted in the following way: we start with the function $sin(3\pi x)$ that reach with time the steady state $\frac{1}{\pi^2} sin(\pi x)$, that is the solution of the time independent Poisson problem. The implementation of the problem above can be seen in Figure 47. In Figure 48 the analytical and numerical solution of the Poisson problem is shown.

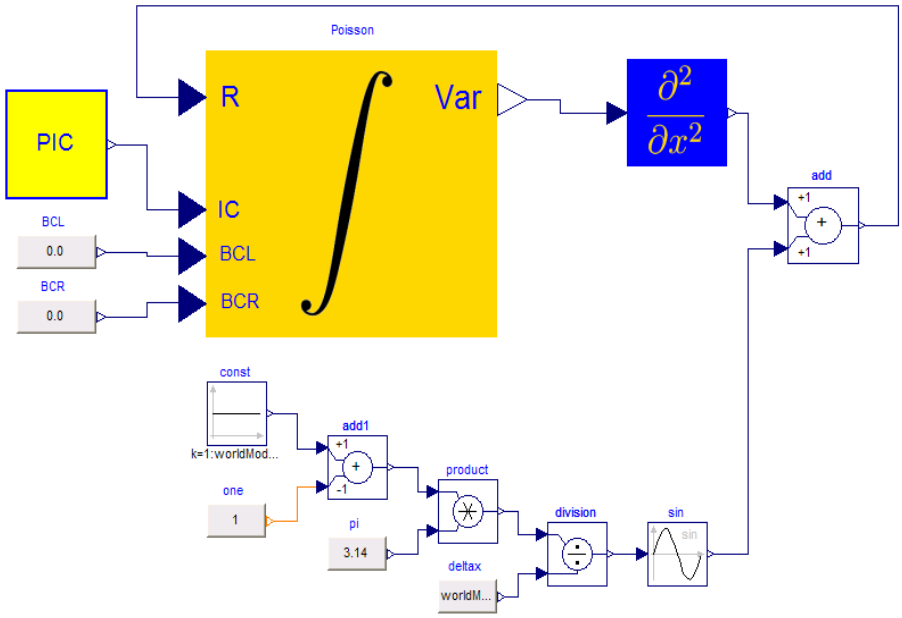

Figure 47: MOL implementation of the Poisson problem in Modelica

As expected, the time dependent Poisson solution approaches the solution of the time independent Poisson problem. Poisson problem is the only time independent problem implemented in the PDE package.

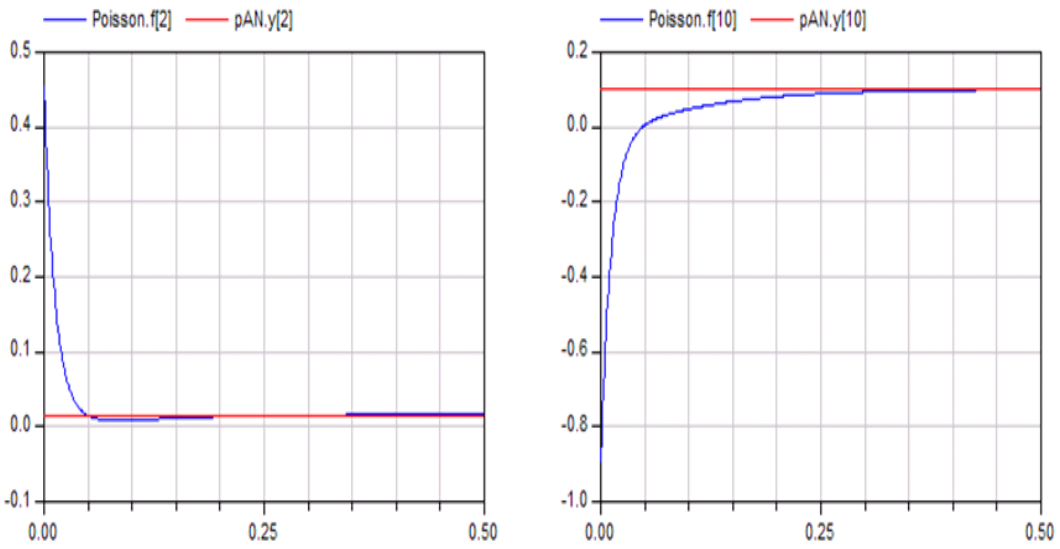

Figure 48: Numerical (blue) versus analytical (red) solution of the poisson problem for the second (left) and middle (right) grid points (20 grid points were used

5 Adaptive Method of Lines

5.1 Introduction

In many practical problems we must refine the grid in order to achieve the required accuracy. Unfortunately this has high storage and computation costs. Sometimes we need the high accuracy only in regions of interest and it would be ideal to refine the grid in this region of interest while keeping the grid size in other regions coarse. This is the idea of adaptive grid refinement. Adaptive grids are often used in practice and give good results. In Figure 49 we see an example of adaptive grid: Delaunay triangulation around an airfoil. (The grid is finer near the airfoil because this is the region of interest). Many grid generation algorithms exists, but most companies try to keep the secret of certain algorithms for themselves. Nevertheless, there are open source projects, such as

www.openmesh.org
www.cgal.org

that allow to download software and documentation and encourage their further development. In practice usually the Delaunay Triangulation is used to generate the grid. In 2D Delaunay Triangulation partitions the domain in triangles while in 3D in tethraedras. When generating an adaptive grid, we need some criteria that tell us where the grid must be refined. So, during the simulation time, these criteria must be checked, and when satisfied, grid refinement must take place. In contrast, if the refined region is

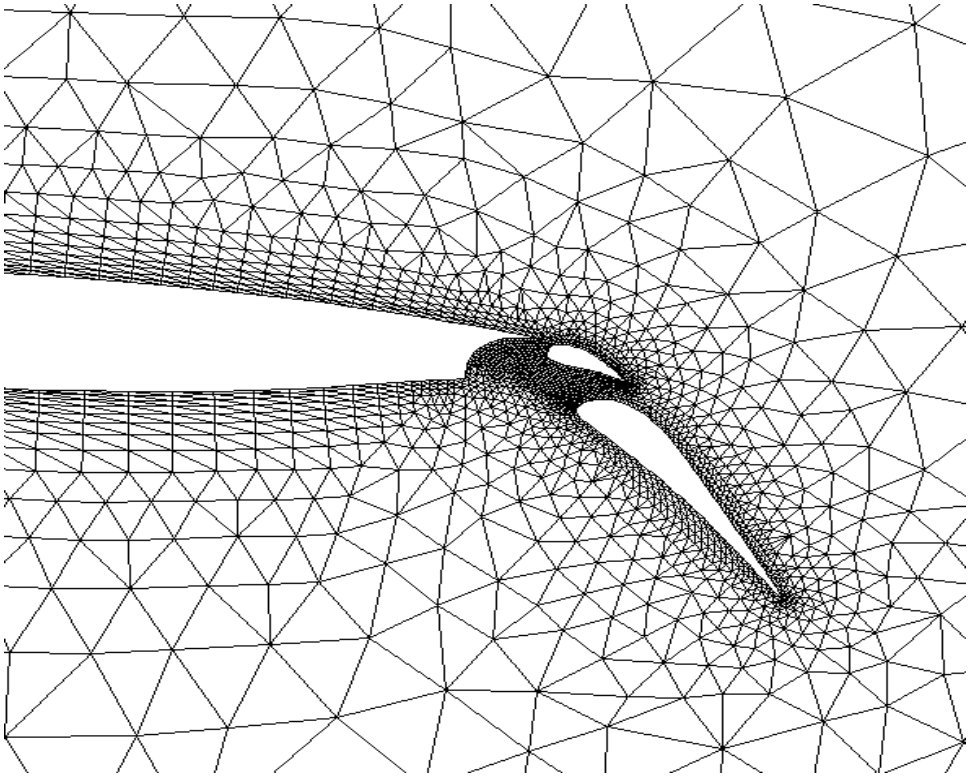

Figure 49: 2D Delaunay Triangulation (picture from www.cerfacs.fr/mueller)

not interesting anymore, that is the refinement criteria are not satisfied, then we would like to eliminate the refinement. This is an important point in adaptive grid refinement, which tells us that the process is not static but dynamic: refinement is "created" or "eliminated" at simulation time. It remains now the question of which criteria must be satisfied. In the following we shall describe the process in 1D. Mack Hyman proposed in [10] the following approach: we operate on a fixed grid but we pay attention that the following criteria are satisfied at all times:

$$\delta x_i(t) \cdot |\frac{\partial u}{\partial x}(x_i, t)| \leq k_{max} \tag{202}$$

This suggests that we must decrease $\delta x_i(t)$ if at time t the absolute spatial gradient at point x_i is big enough so that the criteria 202 are not satisfied anymore. We do that by introducing a new grid point between two existing grid points and initializing its value (initial condition) by using spatial interpolation. The process is illustrated in Figure 50. Note that by introducing a new grid point, we introduce a new differential equation for this point and by eliminating a grid point we eliminate the differential equation associated with this point. The generation/elimination of the auxiliary grid points happens during simulation time and is fully transparent to the user. At the end of the simulation only the solution for the grid points that the user specified in the beginning of the simulation is given.

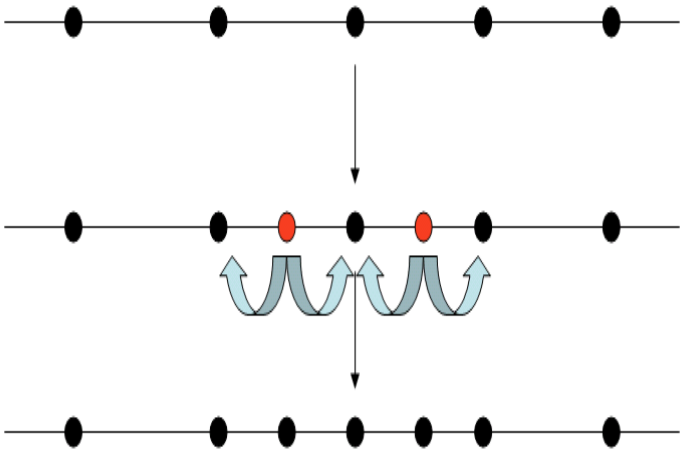

Figure 50: Grid refinement: the auxiliary points (in red) are created. The arrows shows the grid points used for interpolation

5.2 Implementation

As already said, the nature of the adaptive grid refinement algorithms require that the auxiliary points are created/removed dynamically at run time. This creates a big problem in Dymola because it is not allowed to create/destroy instances at run time in Dymola. To overcome this problem, we can try the following ways:

1. Declare an array with known size (as a *parameter* variable in Modelica), so that the user can modify the size of the array at the beginning of each simulation. Problem: the user doesn´t know how many auxiliary points would be necessary for the simulation. It is possible to declare a big array in order to be sure that there is enough space for auxiliary points, but this requires high storage.

2. Write an external function that solves the problem of creating the auxiliary points and computing their solutions. Problem: external help is needed.

Although the second idea solves the problem of adaptivity, the process doesn´t happen in Dymola. It is as trying to ask somebody else for help. For this reason the adaptive grid refinement wasn´t implemented at all in this thesis. Yet future research projects will hopefully solve this problem in Dymola and maybe in the future it will be possible to implement adaptive grid refinement directly in Dymola, without the need of external help.

6 Finite Volume Method

6.1 Introduction

In one dimension, the finite volume method consists in subdividing the spatial domain into intervals, "finite volumes" (or cells), and approximate the integral of the function q over each of these volumes at each time step [2]. Denote the i-th finite volume by

$$C_i = (x_{i-1/2}, x_{i+1/2}) \tag{203}$$

Then the approximation to the average of q in the cell C_i at time t, which we denote with Q_i^t, is

$$Q_i^t \approx \frac{1}{\triangle x} \int_{C_i} q(x,t)dx \tag{204}$$

Remains the question of how to find this approximation. If we think about the conservation law, we note that the average within the cell can only change due to the fluxes at the boundaries (if we assume that no source or sink is present in the cell). The integral form of the conservation law is

$$\frac{d}{dt} \int_{C_i} q(x,t)dx = f(q(x_{i-1/2},t)) - f(q(x_{i+1/2}),t) \tag{205}$$

If we integrate this expression in time from t to $t + \Delta t$, we obtain

$$\int_{C_i} q(x,t+\Delta t)dx - \int_{C_i} q(x,t)dx = \int_t^{t+\Delta t} f(q(x_{i-1/2},t))dt - \int_t^{t+\Delta t} f(q(x_{i+1/2}),t)dt \tag{206}$$

and dividing by Δx we reach the form

$$\frac{1}{\Delta x} \int_{C_i} q(x,t+\Delta t)dx = \frac{1}{\Delta x} \int_{C_i} q(x,t)dx - \frac{1}{\Delta x}(\int_t^{t+\Delta t} f(q(x_{i+1/2},t))dt - \int_t^{t+\Delta t} f(q(x_{i-1/2},t))dt) \tag{207}$$

which gives us an explicit time marching algorithm. This is more clearly seen if we rewrite the expression using the notation we introduced above:

$$Q_i^{t+\Delta t} = Q_i^t - \frac{\Delta t}{\Delta x}(F_{i+1/2}^t - F_{i-1/2}^t) \tag{208}$$

where $F_{i-1/2}^t$ approximates the average flux along the interface $x_{i-1/2}$:

$$F_{i-1/2}^t \approx \frac{1}{\Delta t} \int_t^{t+\Delta t} f(q(x_{i-1/2},t))dt \tag{209}$$

As can be seen from the equation 208, in order to find the average at the next time step we need to find the fluxes at the interfaces. The flux at the interface $x_{i-1/2}$ for example, depends on $q(x_{i-1/2}, t)$, which changes with time along the interface and for which we do not know the analytical solution. For this reason we need to find some approximation to these fluxes in order to calculate the averages at the next time step. Let us now see some simple flux approximations.

Advection Equation
Consider the advection equation $q_t + \bar{u}q_x = 0$, where $\bar{u}$ is the fluid velocity. We have seen in the previous chapters, that the flux of the contaminant at some point x, at some time t, could be written as $\bar{u}q(x,t)$. Consider now the flux through the interface $x_{i-1/2}$. If $\bar{u} > 0$ then the flux will be $\bar{u}Q_{i-1}$, otherwise, if $\bar{u} < 0$, the flux will be $\bar{u}Q_i$. Inserting it into the average update rule, we obtain the finite volume method for the advection equation:

$$Q_i^{t+\Delta t} = Q_i^t - \frac{\bar{u}\Delta t}{\Delta x}(Q_i^t - Q_{i-1}^t) \tag{210}$$

if $\bar{u} > 0$, and

$$Q_i^{t+\Delta t} = Q_i^t - \frac{\bar{u}\Delta t}{\Delta x}(Q_{i+1}^t - Q_i^t) \tag{211}$$

if $\bar{u} < 0$.

Diffusion Equation
In the advection equation, the flux depends on q: $f(q) = \overline{u}q$. The flux in the diffusion equation depends on the derivative of q:

$$f(q_x) = -\beta q_x \tag{212}$$

where β is the conductivity. If β is space dependent then the flux will depend on space too ($f(x, q_x) = -\beta(x)q_x$). In the following we will assume for simplicity that β is constant.
Now remains the question of how to approximate numerically the diffusion flux. One possibility might be:

$$\textbf{??}F_{i-1/2} = -\beta(\frac{Q_i - Q_{i-1}}{\Delta x}) \tag{213}$$

By inserting this flux approximation into the average update rule 208, we obtain:

$$Q_i^{t+\Delta t} = Q_i^t + \frac{\Delta t}{\Delta x^2}\beta(Q_{i-1}^t - 2Q_i^t + Q_{i+1}^t) \tag{214}$$

It is interesting to note, that after some algebraic manipulations, we can write the average update rule in the form

$$\frac{Q_i^{t+\Delta t} - Q_i^t}{\Delta t} = -\frac{1}{\Delta x}(F_{i+1/2}^t - F_{i-1/2}^t) \tag{215}$$

which is equivalent to the finite difference discretization of the conservation law equation $q_t + f(q)_x = 0$. As said in [2]: *Many methods can be equally well viewed as finite difference approximations to this equation or as finite volume methods.*
Another form of the average update rule is

$$\frac{d}{dt}Q_i = -\frac{1}{\Delta x}(F_{i+1/2}^t - F_{i-1/2}^t) \tag{216}$$

which gives us an ODE for each average cell. This form is more suitable for the implementation in Dymola, and all Finite Volume Method blocks are based on this form of update rule.

6.2 Unstable Method

The unstable flux just takes the arithmetic average of the fluxes based on either side of the interface $x_{i-1/2}$:

$$F_{i-1/2}^t = \frac{1}{2}(f(Q_{i-1}^t) + f(Q_i^t)) \tag{217}$$

By using this flux, the average update rule becomes:

$$Q_i^{t+\Delta t} = Q_i^t - \frac{\Delta t}{2\Delta x}(f(Q_{i+1}^t) - f(Q_{i-1}^t)) \tag{218}$$

This method is generally unstable.

6.3 Lax-Friedrichs Method

The Lax-Friedrichs flux is defined as:

$$F_{i-1/2}^t = \frac{1}{2}(f(Q_{i-1}^t) + f(Q_i^t)) - \frac{\Delta t}{2\Delta x}(Q_i^t - Q_{i-1}^t) \tag{219}$$

inserting it into the average update rule, we obtain the Lax-Friedrichs method:

$$Q_i^{t+\Delta t} = \frac{1}{2}(Q_{i-1}^t + Q_{i+1}^t) - \frac{\Delta t}{2\Delta x}(f(Q_{i+1}^t) - f(Q_{i-1}^t)) \tag{220}$$

If we take a closer look at the Lax-Friedrichs flux, we notice that it is similar to the

unstable flux, but with the addition of some correction term. This correction term looks like the diffusion flux **??** with

$$\beta = \frac{(\Delta x)^2}{2\Delta t} \tag{221}$$

The Lax-Friedrichs flux can thus be interpreted as the unstable flux plus a numerical diffusion. This numerical diffusion dampens the instabilities that arise in the unstable method, however, it dampens it too much. Later we will see another method, the Lax-Wendroff method, that adds less diffusion.

6.4 Implementation

The finite volume method in its general form is implemented in the "FVMIntegrator" block (Figure 51). In Figure 52 the principal code of this block is shown: The treatment

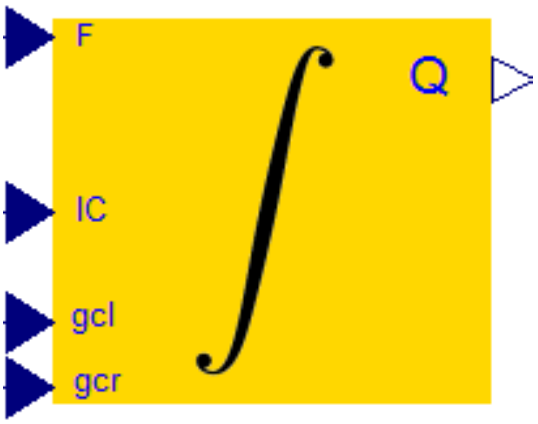

Figure 51: FVM integrator block

of boundary conditions (at the beginning of the code) will be explained later. For now, the important thing to say is that each entry i of the array q is the i-th average. The parameters vb and ve, as well as icb and ice are specified by the user.

6.5 System of Equations

Until now we have seen how to solve one equation with the finite volume method. Many interesting problems in practice are described as a system of equations. The Euler system of equations that we described in the previous Chapter is an example of such a system:

```
Real q[n+gcl+gcr];

equation
  y = q;

  for i in 1:gcl loop
     q[i]= u3[i];
  end for;

  if bcl == 1 then
     q[gcl+1] = q[gcl];
  end if;

  for i in 1:gcr loop
     q[gcl+n+i] = u4[i];
  end for;

  if bcr == 1 then
     q[gcl+n] = q[gcl+n+1];
  end if;

  for i in vb:ve loop
    der(q[i]) = -(1/delta_x)*(u[i-gcl+1]-u[i-gcl]);
  end for;

initial equation

  for i in icb:ice loop
    q[i] =  u2[i-gcl];
  end for;
```

Figure 52: FVM integrator code

$$\begin{cases} \rho_t + (\rho v)_x = 0 \\ (\rho v)_t + (\rho v^2 + p)_x = 0 \\ E_t + ((E+p)v)_x = 0 \end{cases} \tag{222}$$

The Euler system is a nonlinear hyperbolic system. There are other systems that are much simpler. We can have for example a system with linear constant coefficients. An example of such a system are the acustics equations:

$$\begin{cases} p_t + K_0 v_x = 0 \\ v_t + \frac{p_x}{\rho_0} = 0 \end{cases} \tag{223}$$

where p is the pressure, v the velocity and K_0, ρ_0 are constant values.
Consider in general a system of the form

$$q_t + Aq_x = 0 \tag{224}$$

where A is an $m \times m$ matrix. Such a system is called *hyperbolic* if the matrix A is diagonalizable and has real eigenvalues.
For example in the acoustics system considered above we have

$$A = \begin{pmatrix} 0 & K_0 \\ \frac{1}{\rho_0} & 0 \end{pmatrix} \tag{225}$$

The matrix A is diagonalizable and has real eigenvalues

$$\lambda_1 = -c_0 \tag{226}$$

$$\lambda_2 = c_0 \tag{227}$$

where $c_0 = \sqrt{\frac{K_0}{\rho_0}}$ is the speed of sound in the gas.
In the following we will study hyperbolic system of equations. The remarkable property of the hyperbolic system is that we can rewrite it in a way which is simpler to solve.
If a system is hyperbolic then we can rewrite it as

$$q_t + Aq_x = 0 \tag{228}$$

$$q_t + R\Lambda R^{-1} q_x = 0 \tag{229}$$

$$R^{-1} q_t + \Lambda R^{-1} q_x = 0 \tag{230}$$

With a simple variable transformation

$$w = R^{-1} q \tag{231}$$

the system can be rewritten now as

$$w_t + \Lambda w_x = 0 \tag{232}$$

where Λ is the eigenvalue matrix.
This is a very important fact, because now from a coupled system 224 we obtained, through variable transformation, a decoupled system 232. Moreover, the individual equations of the new system are advection equations! Because we now have a system of decoupled equations, we can solve each equation separately

$$w_t^i + \Lambda w_x^i \qquad i = 1, ..., m \tag{233}$$

and combine the solutions $w^i(x,t)$ into a vector $w(x,t)$. The solution to the i-th advection equation is, as already seen in Chapter 4,

$$w^i(x,t) = w^i(x - \lambda^i t, 0) \tag{234}$$

Finally, we obtain, through the variable transformation

$$q(x,t) = Rw(x,t) \tag{235}$$

the solution q(x, t). If we write the solution q(x, t) in the form

$$q(x,t) = Rw(x,t) = \begin{pmatrix} r_1^1 & \dots & r_1^m \\ \vdots & \ddots & \vdots \\ r_m^1 & \vdots & r_m^m \end{pmatrix} \begin{pmatrix} w^1(x,t) \\ \vdots \\ w^m(x,t) \end{pmatrix} = \tag{236}$$

$$\begin{pmatrix} r_1^1 w^1(x,t) + & \dots & + r_1^m w^m(x,t) \\ \vdots & \ddots & \vdots \\ r_1^m w^1(x,t) + & \dots & + r_m^m w^m(x,t) \end{pmatrix} =$$

$$\begin{pmatrix} r_1^1 \\ \vdots \\ r_m^1 \end{pmatrix} \cdot w^1(x,t) + \begin{pmatrix} r_1^m \\ \vdots \\ r_m^m \end{pmatrix} \cdot w^m(x,t) = \sum_{i=1}^{m} w^i(x,t) r^i$$

we see that the solution q(x, t) can be expressed as the linear combination of the right eigenvectors $r^1, ..., r^m$ at each point in space-time, and hence as a superposition of waves propagating at velocities λ^i [2].

6.6 The Riemann Problem

Consider the piecewise constant data with a single jump discontinuity:

$$q(x) = \begin{cases} q_l & \text{if x} < 0 \\ q_r & \text{if x} > 0 \end{cases} \tag{237}$$

The Riemann problem consists of this data as initial data, plus a hyperbolic equation. Example: Let us take the scalar advection equation

$$q_t + u q_x = 0 \tag{238}$$

For this example, the Riemann problem is this equation together with the discontinuity 237. The solution is the discontinuity $q_r - q_l$ that propagates along the characteristic with the speed u, that is:

$$q(x,t) = q_0(x - ut) \tag{239}$$

The same reasoning applies to the system of equations. Let us decompose the q_l and q_r as:

$$q_l = \sum_{i=1}^{m} w_l^i r^i \tag{240}$$

$$q_r = \sum_{i=1}^{m} w_r^i r^i \tag{241}$$

With this decomposition we have for the i-th advection equation:

$$w_0^i = \begin{cases} w_l^i, & \text{if x} < 0 \\ w_r^i, & \text{if x} > 0 \end{cases} \tag{242}$$

and this jump propagates with speed λ^i. The solution is thus

$$w^i(x,t) = \begin{cases} w_l^i, & \text{if } x - \lambda^i t < 0 \\ w_r^i, & \text{if } x - \lambda^i t > 0 \end{cases} \tag{243}$$

If we let P(x, t) be the maximum value of i for which $x - \lambda^i t > 0$ then [2]

$$q(x,t) = \sum_{i=1}^{P(x,t)} w_r^i(x,t) r^i + \sum_{i=P(x,t)+1}^{m} w_l^i(x,t) r^i \tag{244}$$

Consider the example of Figure 53 in which $w^1 = w_r^1$ and $w^2 = w_l^2$. The solution at the point (x_p, t_p) is then

$$q(x_p, t_p) = w_r^1 r^1 + w_l^2 r^2 \tag{245}$$

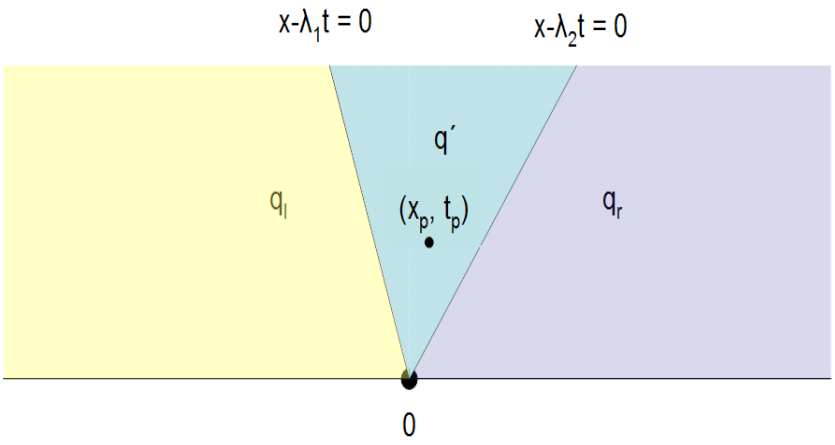

Figure 53: Riemann problem at (x_p, t_p)

The different regions are colored with different colors. The value of q in each region is constant, but as we cross the i-th characteristic, we have a jump from w_l^i to w_r^i.

Since

$$q_r - q_l = \sum_{i=1}^{m} w_r^i r^i - \sum_{i=1}^{m} w_l^i r^i = \sum_{i=1}^{m} (w_r^i - w_l^i) r^i \tag{246}$$

we see that across the i-th characteristic the solution jumps with the jump in q given by

$$(w_r^i - w_l^i) r^i \equiv \alpha^i r^i \tag{247}$$

and this tell us that the jump in q is an eigenvector of the matrix A. This condition is called the *Rankine-Hugoniot jump condition.*
As said in [2]: For the case of a linear system, solving the Riemann problem consists of taking the initial data (q_l, q_r) and decomposing the jump $q_r - q_l$ into eigenvectors of A:

$$q_r - q_l = \alpha^1 r^1 + ... + \alpha^m r^m \tag{248}$$

This requires solving the linear system of equations

$$R\alpha = q_r - q_l \tag{249}$$

for the vector α.
Since $\alpha^i r^i$ is the jump in q across the i-th wave in the solution to the Riemann problem, we introduce the notation

$$W^i = \alpha^i r^i \tag{250}$$

for these waves. The solution q(x, t) can then be written in terms of waves in two different forms:

$$q(x,t) = q_l + \sum_{i:\lambda^i < x/t} W^i = q_r - \sum_{i:\lambda^i \geq x/t} W^i \tag{251}$$

Let us look at an example. Figure 55 and 54 illustrates an example with three characteristics that part at the origin with $\lambda^1 < 0$ and $\lambda^2 > 0$, $\lambda^3 > 0$. Consider the point (x_p, t_p) as illustrated in the figure. What is $q(x_p, t_p)$? We can answer this question in two ways:

- The value of $q(x_p, t_p)$ is q_l plus all the jumps that are on the way from q_l (yellow region) to $q(x_p, t_p)$ (Figure 54).

- The value of $q(x_p, t_p)$ is q_r minus all the jumps that are on the way from q_r (blue region) to $q(x_p, t_p)$ (Figure 55).

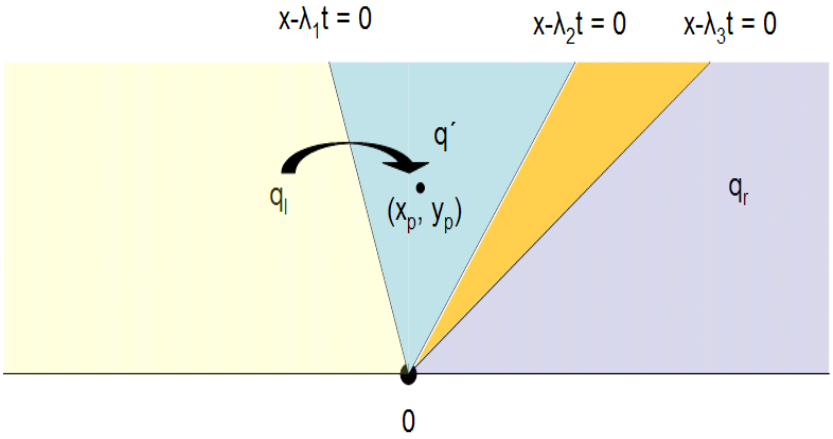

Figure 54: From q_l to $q(x_p, y_p)$

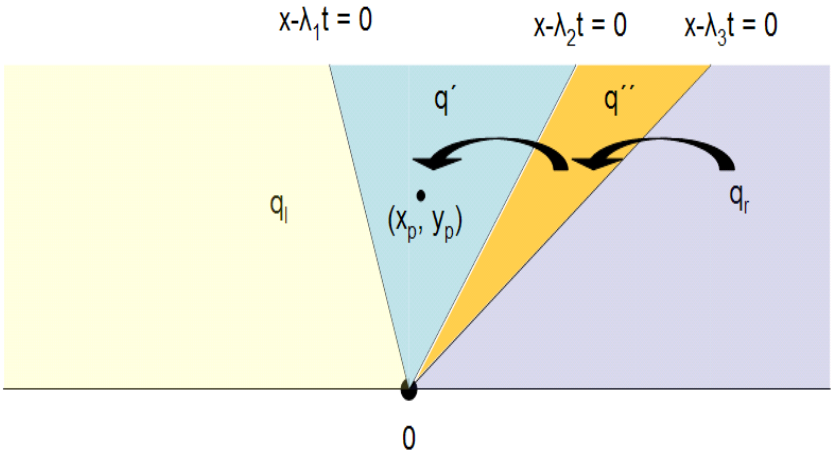

Figure 55: From q_r to $q(x_p, y_p)$

Example (taken from [2]): Coupled acoustics and advection

$$\begin{pmatrix} p \\ v \\ \phi \end{pmatrix}_t = \begin{pmatrix} v_0 & k_0 & 0 \\ \frac{1}{\rho_0} & v_0 & 0 \\ 0 & 0 & v_0 \end{pmatrix} \begin{pmatrix} p \\ v \\ \phi \end{pmatrix}_x \tag{252}$$

The matrix A has eigenvalues

$$\lambda^1 = v_0 - c_0 \tag{253}$$

$$\lambda^2 = v_0 \tag{254}$$

$$\lambda^3 = v_0 + c_0 \tag{255}$$

and eigenvectors

$$r^1 = \begin{pmatrix} -Z_0 \\ 1 \\ 0 \end{pmatrix},\ r^2 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix},\ r^3 = \begin{pmatrix} Z_0 \\ 1 \\ 0 \end{pmatrix} \tag{256}$$

To find the solution to the Riemann problem we must thus solve

$$\begin{pmatrix} p_r - p_l \\ v_r - v_l \\ \phi_r - \phi_l \end{pmatrix} = \alpha^1 \begin{pmatrix} -Z_0 \\ 1 \\ 0 \end{pmatrix} + \alpha^2 \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} + \alpha^3 \begin{pmatrix} Z_0 \\ 1 \\ 0 \end{pmatrix} \tag{257}$$

By solving this system we find

$$\alpha^1 = \frac{1}{2Z_0}(-(p_r - p_l) + Z_0(v_r - v_l)) \tag{258}$$

$$\alpha^2 = \phi_r - \phi_l \tag{259}$$

$$\alpha^3 = \frac{1}{2Z_0}((p_r - p_l) + Z_0(v_r - v_l)) \tag{260}$$

So far we have seen what influence the initial data has at some point x_0 on the solution $q(x,t)$. At any particular time, the data at this point will affect the solution along the characteristics at some set of points (Figure 56). This set of points is called the *range of influence* of the point x_0 [2]. Consider now the inverse problem: we have a point (x_p, t_p) and we are interested in which initial values influence the value of q at this point (Figure 57). In the figure we see that three points influence the point value at the point (x_p, t_p). This can be seen algebraically too, if we write $q(x,t)$ in terms of the initial data. This set of points is called *domain of dependence* of the point (x_p, t_p) [2]. The initial values at other points have no influence on the value at the point (x_p, t_p). In general, for hyperbolic equations, the domain of dependence is always a bounded set.

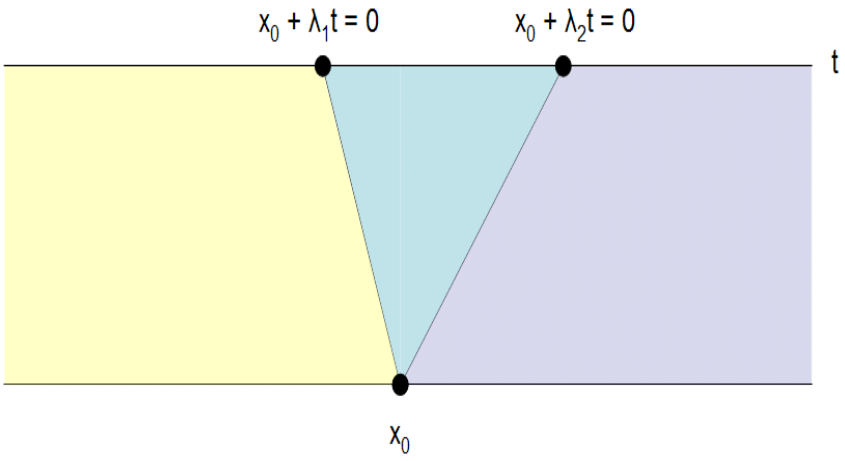

Figure 56: Range of influence

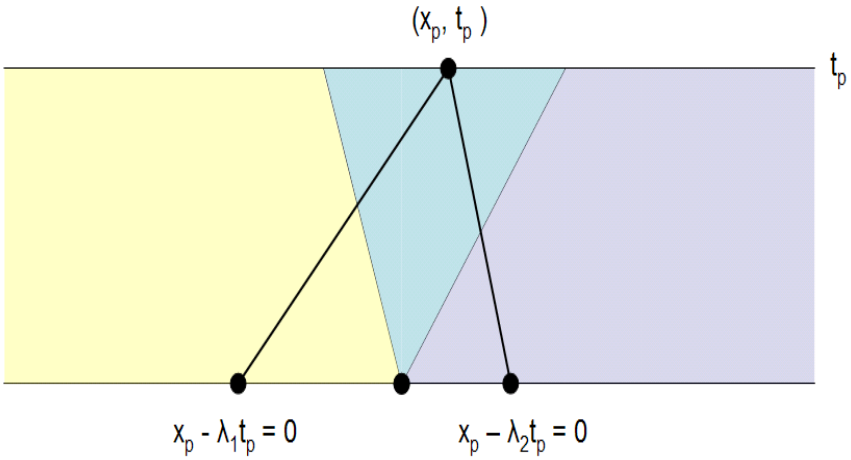

Figure 57: Domain of influence

6.7 Godunov´s Method for Linear Systems

Until now we discussed how to approximate fluxes at the interfaces. But how does the entire process work? The answer to this question is the reconstruct-evolve-average (REA) algorithm. At each time step, we repeat the following process [2]:

Reconstruct-Evolve-Average Algorithm

1. Reconstruct a piecewise polynomial function $\widetilde{q}^t(x,t)$ defined for all x, from the cell averages Q_i^t.
2. Evolve the hyperbolic equation exactly (or approximately) with this data to obtain $\widetilde{q}^t(x,t+\Delta t)$ a time Δt later.
3. Average this function over each grid cell to obtain new cell averages

$$Q_i^{t+\Delta t} = \frac{1}{\Delta x}\int_{C_i} \widetilde{q}^t(x,t+\Delta t)dx \tag{261}$$

Until now we have seen the simplest case, in which the average is constant across the cell. Other approximations are often used in practice, such as the linear approximation (see Flux Limiter Chapter). In this thesis we will use additionally another average reconstruction scheme, the *local double logarithmic reconstruction*. We shall start in the following with the constant reconstruction.
If we take as $\widetilde{q}^t(x,t)$ a piecewise constant function, we can use the Riemann problem theory seen so far in order to solve the hyperbolic equation in step 2 of the REA algorithm.
For this purpose, we need to determine the flux at the boundaries. Remember from section 6.1 that the numerical flux $F_{i-1/2}^t$ approximates the time average of the flux at $x_{i-1/2}$ over the time step Δt.

$$F_{i-1/2}^t \approx \frac{1}{\Delta t}\int_t^{t+\Delta t} f(q(x_{i-1/2},t))dt \tag{262}$$

The density $q(x_{i-1/2},t)$ in general varies in time and we do not know this variation of the exact solution [2]. If we choose constant piecewise reconstruction as $\widetilde{q}^t(x,t)$ in the REA algorithm, however, we can compute then this integral exactly. For better understanding, consider the Riemann problem centered at $x_{i-1/2}$ in the case of the example shown in Figure 58.
Because the interface $x_{i-1/2}$ is situated in a region between the characteristics and we know that $q(x,t)$ is constant in each region between characteristics, then consequently $\widetilde{q}^t(x,t)$ is constant at the interface $x_{i-1/2}$ over the time interval $(t,t+\Delta t)$. If we denote

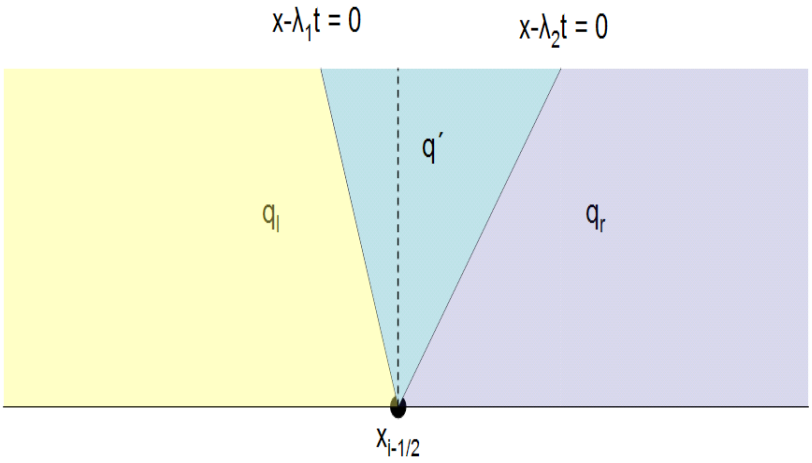

Figure 58: Godunov

this value by $q^{\downarrow}(Q^t_{i-1}, Q^t_i)$ we can write the numerical flux $F^t_{i-1/2}$ as

$$F^t_{i-1/2} = \frac{1}{\Delta t}\int_t^{t+\Delta t} f(q^{\downarrow}(Q^t_{i-1}, Q^t_i))dt = \frac{1}{\Delta t} f(q^{\downarrow}(Q^t_{i-1}, Q^t_i))\int_t^{t+\Delta t} dt = \tag{263}$$

$$\frac{1}{\Delta t} f(q^{\downarrow}(Q^t_{i-1}, Q^t_i))\Delta t = f(q^{\downarrow}(Q^t_{i-1}, Q^t_i))$$

The Godunov method becomes then [2]

- Solve the Riemann problem at $x_{i-1/2}$ to obtain $q^{\downarrow}(Q^t_{i-1}, Q^t_i)$
- Define the flux $F^t_{i-1/2} = f(q^{\downarrow}(Q^t_{i-1}, Q^t_i))$
- Apply the average update rule $Q^{t+\Delta t}_i = Q^t_i - \frac{\Delta t}{\Delta x}(F_{i+1/2} - F_{i-1/2})$

We will now look in detail how to develop formulas for Godunov´s method on linear systems of equations. For this purpose consider the example in Figure 59. In this example we have two waves, one propagating with the velocity $\lambda^1 < 0$ and one with the velocity $\lambda^2 > 0$. By using the notation introduced above, we can write the jump in q given by each wave as

$$W^1_{i-1/2} = \alpha^1_{i-1/2} r^1 \tag{264}$$

$$W^2_{i-1/2} = \alpha^2_{i-1/2} r^2 \tag{265}$$

As can be seen from the figure, only the right-going wave from $x_{i-1/2}$ interface and the

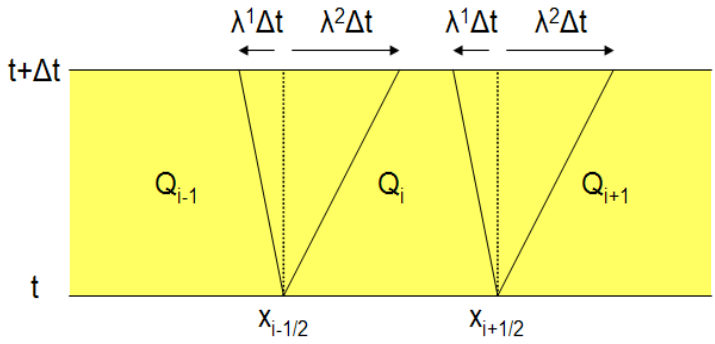

Figure 59: Waves

left-going wave from $x_{i+1/2}$ interface affect the cell average Q_i. After time step Δt the W^1 wave has moved a distance $\lambda^1 \Delta t$ or a portion $\frac{\lambda^1 \Delta t}{\Delta x}$ of the cell and W^2 a distance $\lambda^2 \Delta t$ or a portion $\frac{\lambda^2 \Delta t}{\Delta x}$ of the cell.
The W^1 wave changes thus the cell average value by the amount

$$-\frac{\lambda^1 \Delta t}{\Delta x} W^1_{i-1/2} \tag{266}$$

and the wave W^2 by

$$-\frac{\lambda^2 \Delta t}{\Delta x} W^2_{i+1/2} \tag{267}$$

The process is visualized in Figure 60, where we took a positive velocity u. The total change in the cell average is just the sum of these independent changes contributed by each wave.

$$Q_i^{t+\Delta t} = Q_i^t - \frac{\lambda^1 \Delta t}{\Delta x} W^1_{i-1/2} - \frac{\lambda^2 \Delta t}{\Delta x} W^2_{i+1/2} = \tag{268}$$
$$Q_i^t - \frac{\Delta t}{\Delta x}(\lambda^1 W^1_{i-1/2} + \lambda^2 W^2_{i+1/2})$$

This can be generalized to arbitrary hyperbolic systems of m equations. If we introduce the notation

$$\lambda^- = min(\lambda, 0) \tag{269}$$
$$\lambda^+ = max(\lambda, 0) \tag{270}$$

we can write 268 as

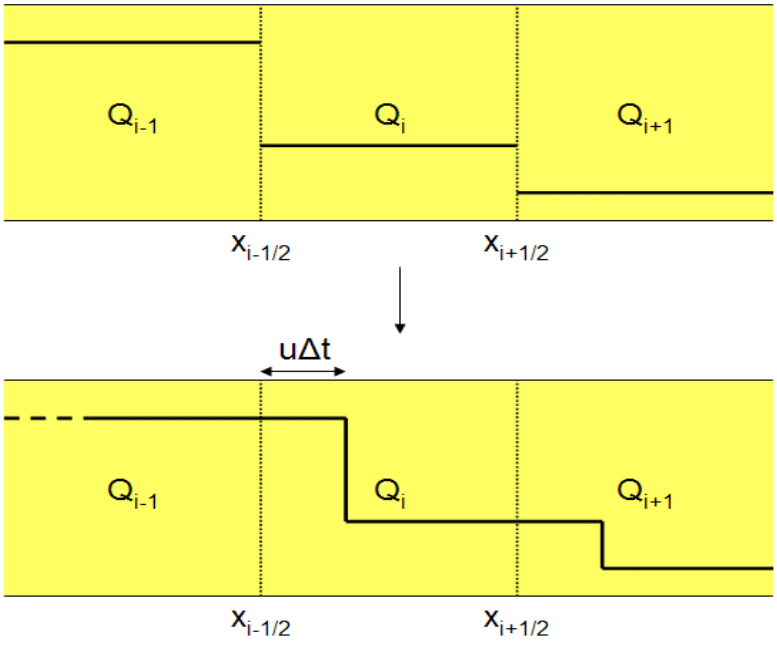

Figure 60: Waves

$$Q_i^{t+\Delta t} = Q_i^t - \frac{\Delta t}{\Delta x}(\sum_{j=1}^{m}(\lambda^j)^+ W_{i-1/2}^j + \sum_{j=1}^{m}(\lambda^j)^- W_{i+1/2}^j) \tag{271}$$

Only the right-going waves from the $x_{i-1/2}$ interface and the left-going waves from the $x_{i+1/2}$ interface affect the cell average Q_i. We introduce the symbols:

$$A^-\Delta Q_{i-1/2} = \sum_{j=1}^{m}(\lambda^j)^- W_{i-1/2}^j \tag{272}$$

$$A^+\Delta Q_{i-1/2} = \sum_{j=1}^{m}(\lambda^j)^+ W_{i-1/2}^j \tag{273}$$

In this way we can rewrite 271 as

$$Q_i^{t+\Delta t} = Q_i^t - \frac{\Delta t}{\Delta x}(A^+\Delta Q_{i-1/2} + A^-\Delta Q_{i+1/2}) \tag{274}$$

The symbols $A^+\Delta Q_{i-1/2}$ and $A^-\Delta Q_{i-1/2}$ should be interpreted as the net effect of all right-going waves, respectively left-going waves, from the interface $x_{i-1/2}$.
Define now the matrices

$$\Lambda^+ = \begin{pmatrix} (\lambda^1)^+ & 0 & \dots & 0 \\ 0 & (\lambda^2)^+ & \dots & \dots \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & (\lambda^m)^+ \end{pmatrix}, \quad \Lambda^- = \begin{pmatrix} (\lambda^1)^- & 0 & \dots & 0 \\ 0 & (\lambda^2)^- & \dots & \dots \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & (\lambda^m)^- \end{pmatrix} \tag{275}$$

and

$$A^+ = R\Lambda^+ R^{-1} \tag{276}$$

$$A^- = R\Lambda^- R^{-1} \tag{277}$$

we find that

$$A^+ + A^- = R(\Lambda^+ + \Lambda^-)R^{-1} = R\Lambda R^{-1} = A \tag{278}$$

Thus we can split the matrix A in $A^+ + A^-$. Why is it useful? To answer the question we go now further and do the following:

$$A^+\Delta Q_{i-1/2} = R\Lambda^+ R^{-1}(Q_i - Q_{i-1}) = R\Lambda^+\alpha_{i-1/2} = \sum_{j=1}^{m}(\lambda^j)^+\alpha^j_{i-1/2}r^j = N^+\Delta Q_{i-1/2} \tag{279}$$

In the same manner we find that

$$A^-\Delta Q_{i-1/2} = N^-\Delta Q_{i-1/2} \tag{280}$$

What does it tell us? It states that for the linear constant-coefficient hyperbolic system, the net effect of all right going waves from the interface $x_{i-1/2}$ can be calculated by simply multiplying the matrix A^+ by the jump in Q. Similarly, we can find the net effect of all left-going waves from the interface $x_{i-1/2}$ by multiplying the matrix A^- by the jump in Q.
Let us now look at the numerical flux function. We know that the value of q in the Riemann solution along the interface $x_{i-1/2}$ is

$$Q^{\downarrow}_{i-1/2} = q^{\downarrow}(Q_{i-1}, Q_i) = Q_{i-1} + \sum_{j:\lambda^j<0} W^j_{i-1/2} \tag{281}$$

Because in the linear case $f(Q^{\downarrow}_{i-1/2}) = AQ^{\downarrow}_{i-1/2}$, we have

$$F^t_{i-1/2} = AQ_{i-1} + \sum_{j:\lambda^j<0} AW^j_{i-1/2} = AQ_{i-1} + \sum_{j=1}^{m}(\lambda^j)^- W^j_{i-1/2} \tag{282}$$

Proceeding the same way we find $F^t_{i+1/2}$ and inserting both in the average update rule we obtain

$$Q_i^{t+\Delta t} = Q_i^t - \frac{\Delta t}{\Delta x}(F^t_{i+1/2} - F^t_{i-1/2}) = \tag{283}$$
$$Q_i^t - \frac{\Delta t}{\Delta x}(\sum_{j=1}^{m}(\lambda^j)^- W^j_{i+1/2} - \sum_{j=1}^{m}(\lambda^j)^+ W^j_{i-1/2})$$

Note that if we use $A = A^+ + A^-$ in

$$F^t_{i-1/2} = AQ_{i-1} + \sum_{j:\lambda^j<0} AW^j_{i-1/2} = AQ_{i-1} + \sum_{j=1}^{m}(\lambda^j)^- W^j_{i-1/2} \tag{284}$$

we obtain

$$F^t_{i-1/2} = A^+Q_{i-1} + A^-Q_i \tag{285}$$

This shows us the important difference between taking the average of $AQ_{i-1}$and AQ_i as in the unstable method and the average in which we take the part of AQ_{i-1} corresponding to right-going waves and combine it with the part of AQ_i corresponding to left-going waves in order to obtain the flux in between [2].

6.8 Boundary Conditions

In our derivation of the cell average update rule we used information from the neighboring cells. In the upwind method, we need the information from the cells Q_i and Q_{i-1} if the velocity is positive and we need cells Q_i and Q_{i+1} if the velocity is negative. So if our domain consists of n cells Q_1, ..., Q_n, then to update the first average cell for example we would need information from the previous cell if the velocity is positive. But there is no previous cell. We do not have information outside the domain. So to complete the problem, we need this information at the boundary of the domain, the boundary conditions. In the Method of line chapter we used special formulas or assigned values for the cells at the boundaries, that is, we treated them in a special way. Finite Volume Method follows another philosophy: extend the domain with additional cells, the ghost cells (Figure 61). As the name already suggests, ghost cells are not part of the domain, they are only used to provide necessary information for the computation of the boundary cells. By introducing ghost cells, we do not have to write special formulas for the boundary cells, we just use the same formula for all cells in the domain. At the start of each time step we have the average values for the cells in the domain obtained either from the previous time step or from the initial condition if $t = 0$ and we fill the ghost cells with the boundary values before applying the method on the next time step. What

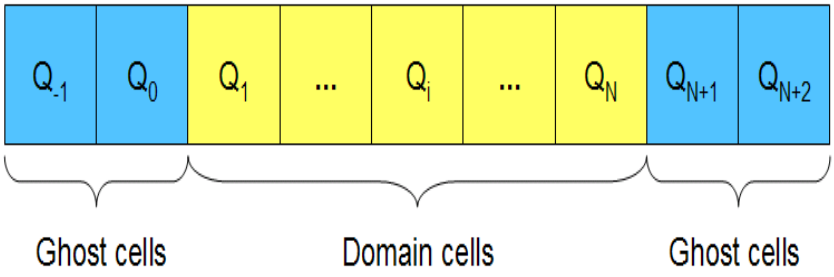

Figure 61: Boundary conditions: ghost cells and domain cells

kind of boundary values can we have? In the following we will consider some of the most used boundary conditions.

6.8.1 Periodic Boundary Conditions

Assume that we are using a 3-point stencil method. In this case we only need one ghost cell at each end of the boundary. Periodic boundary conditions set the value of the left ghost cell to the value of the last domain cell and the value of the right ghost cell to the value of the first domain cell (Figure 62).

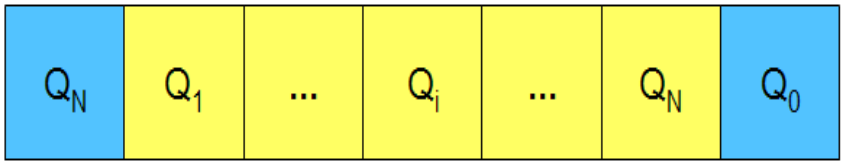

Figure 62: Periodic boundary conditions

In this way what leaves the domain at the right, enters the domain from the left. The same for the other direction. The same reasoning applies to the higher order stencil method.

6.8.2 Inflow and Outflow Boundary Conditions

In the following we will analyze the inflow and outflow boundary conditions by taking as an example the advection equation. This is a reasonable choice since as already seen, a linear constant-coefficient hyperbolic system can be transformed into a decoupled system of advection equations. We will begin with the outflow boundary condition. In practice usually the domain cells are used to assign the boundary condition. The simplest way is to extrapolate by a constant function, a "zero-order" extrapolation:

$$Q^t_{N+1} = Q^t_N \tag{286}$$
$$Q^t_{N+2} = Q^t_N \tag{287}$$

The "zero-order" extrapolation works well in practice and is preferred to "first-order" extrapolation, where a linear function is used, which can lead to instabilities.
Let us now discuss the inflow boundary condition. For this purpose we consider the advection equation with the positive velocity. Say, that we have at the left boundary a boundary condition that changes with time:

$$q(a,t) = g_0(t) \tag{288}$$

How can we compute the flux at the boundary $x_{1/2}$? One possibility would be to compute it exactly by using the formula for the flux:

$$F^t_{1/2} = \frac{1}{\Delta t}\int_t^{t+\Delta t} f(q(x_{1/2},t))dt \tag{289}$$

Since $f(q(x,t))$ for the advection equation is $\overline{v}q(x,t)$, we have

$$F^t_{1/2} = \frac{1}{\Delta t}\int_t^{t+\Delta t} \overline{v}q(a,t)dt = \frac{\overline{v}}{\Delta t}\int_t^{t+\Delta t} g_0(t)dt \tag{290}$$

Now that we have the flux at the interface $x_{1/2}$ we can use the average update rule.
What if we have more than one ghost cell at the left side of the domain? We would need the flux at the interface $x_{-3/2}$ too. At a first glance it seems impossible to compute this value, because the solution is not defined outside the domain. But if we consider the fact that we have some function that enters the domain and is simply advected, then we could use this information to find out the value of the function outside the domain. Consider an x in the interval $[a-\Delta x, a]$. Because we are dealing with the advection equation, the value of $q(x,t)$ at the point x is

$$q(x,t) = q(a, t + \frac{a-x}{\overline{v}}) = g_0(t + \frac{a-x}{\overline{v}}) \tag{291}$$

and the average Q^t_0 can then be computed as

$$Q^t_0 = \frac{1}{\Delta x}\int_{a-\Delta x}^{a} g_0(t + \frac{a-x}{\overline{v}})dx = \frac{\overline{v}}{\Delta x}\int_t^{t+\frac{\Delta x}{\overline{v}}} g_0(\tau)d\tau \tag{292}$$

Now we have enough information to pass to the system of equations. Let us take as an example, the acoustic system

$$p_t + k_0 v_x = 0 \tag{293}$$

$$\rho_0 v_t + p_x = 0 \tag{294}$$

This system has the characteristic variables:

$$w_1(x,t) = \frac{1}{2Z_0}(-p + Z_0 v) \tag{295}$$

$$w_2(x,t) = \frac{1}{2Z_0}(p + Z_0 v) \tag{296}$$

where Z_0 is a constant value (impedance). The problem that we encounter here is that the acoustic system has both negative and positive eigenvalues. This means that each boundary will have both incoming and outgoing characteristics. How can we set the boundary conditions in this case? Well, in the ideal case we would wish that waves that leave the domain, do not influence waves that enter the domain. This way we avoid back propagation of the spurious reflections into the domain. Boundary conditions that achieve this task are called *absorbing* boundary conditions.
It turns out that we can implement absorbing boundary conditions by just using the zero-order extrapolation, that is by setting

$$Q_0^t = Q_1^t \tag{297}$$

$$Q_{-1}^t = Q_1^t \tag{298}$$

Other boundary conditions that we could have are the incoming waves. In our example, say that we have some incoming signal that varies with time:

$$w_2(a,t) = g(t) \tag{299}$$

and we set nonreflection boundary conditions, that is no reflection of any outgoing signal. In the acoustics example the incoming signal comes from the second characteristic variable. By decomposing the average cell value Q_1 into

$$Q_1 = W_1^1 r_1 + W_2^2 r_2 \tag{300}$$

and using $w_2(a,t) = g(t)$ we have that the incoming wave at the interface $x_{1/2}$ is $g(t)r^2$ and by using the property of the advection equation we obtain for the average Q_0:

$$Q_0 = W_1^1 r_1 + g(t + \frac{\Delta x}{2c_0})r^2 \tag{301}$$

6.9 High Resolution Methods

Consider the linear constant-coefficient system $q_t + Aq_x = 0$. If we expand $q(x, t + \Delta t)$ in a Taylor series, we obtain:

$$q(x, t + \Delta t) = q(x, t) + \triangle t q_t(x, t) + \frac{1}{2}(\triangle t)^2 q_{tt}(x, t) + ... \tag{302}$$

From the equation $q_t + Aq_x = 0$ we know that

$$q_t = -Aq_x \tag{303}$$

and differentiating q_t with respect to time, we also find

$$q_{tt} = (-Aq_x)_t = (-Aq_t)_x = (-A(-Aq_x))_x = A^2 q_{xx} \tag{304}$$

Inserting now q_t and q_{tt} in the Taylor expansion gives

$$q(x, t + \Delta t) = q(x, t) - \triangle t A q_x(x, t) + \frac{1}{2}(\triangle t)^2 A^2 q_{xx}(x, t) + ... \tag{305}$$

If we truncate the series from the fourth term and substitute the spatial derivatives with the central difference scheme, we obtain a second order accurate method, called the Lax-Wendroff method:

$$Q_i^{t+1} = Q_i^t - \frac{\triangle t}{2\triangle x} A(Q_{i+1}^t - Q_{i-1}^t) + \frac{1}{2}\frac{\triangle t}{\triangle x}^2 A^2(Q_{i-1}^t - 2Q_i^t + Q_{i+1}^t) \tag{306}$$

Although we used a finite difference method to derive the Lax-Wendroff method, we can rewrite it in the flux-differencing form with the flux

$$F_{i-1/2}^t = \frac{1}{2}A(Q_{i-1}^t + Q_i^t) - \frac{1}{2}\frac{\triangle t}{\triangle x}A^2(Q_i^t - Q_{i-1}^t) \tag{307}$$

This looks like the unstable flux plus a diffusive flux, like in the Lax-Friedrichs flux, but this time the diffusion chosen exactly matches what appears in the Taylor series expansion [2].
Lax-Wendroff approximates well the smooth functions, but not the discontinuous ones. In fact, the Lax-Wendroff Method leads to an oscillatory solution near the discontinuities. This can be explained if we consider again the Taylor series expansion. Because we truncated the expansion from the fourth term, the dominant error results from the fourth term:

$$q_{ttt} = -A^3 q_{xxx} \tag{308}$$

which is the dispersive term that causes the oscillations.

6.10 Piecewise Linear Reconstruction

Until now we have reconstructed a piecewise constant function from the cell averages. By applying the REA algorithm with this reconstruction we derived the upwind method. This approach is, however, first-order accurate. In order to achieve better accuracy we must use another reconstruction. In the following we will explain the piecewise linear reconstruction.
Given the cell averages Q_i at some time t, we can use these averages to construct a piecewise linear function

$$\tilde{q}^t(x,t) = Q_i^t + \sigma_i^t(x - x_i) \tag{309}$$

where $x_{i-1/2} \leq x < x_{i+1/2}$, x_i is the center of the grid cell and σ_i is the slope on the *i-th* cell (Figure 63). It is important to note that over the cell C_i the value of $\tilde{q}^t(x,t)$ is

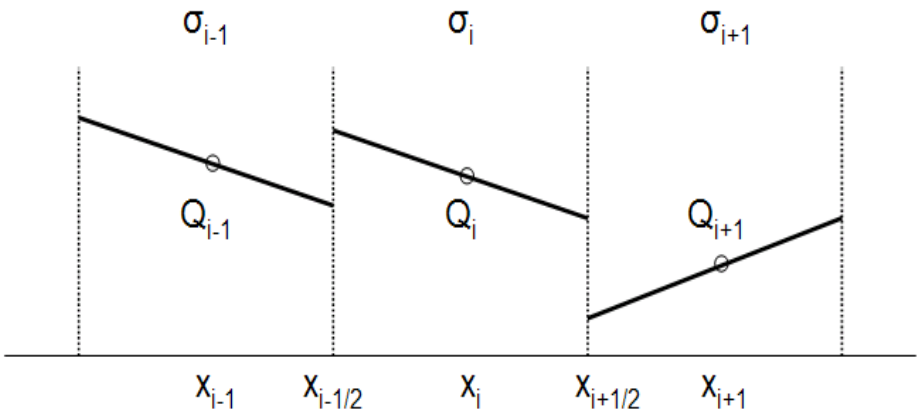

Figure 63: Linear reconstruction

Q_i. Once we have the reconstruction, it remains to solve the hyperbolic equation with this reconstruction and to compute the new cell averages. In the following we consider the scalar advection equation in which the velocity is positive, $\overline{v} > 0$. In this case we have

$$\tilde{q}^t(x, t + \Delta t) = \tilde{q}^t(x - \overline{v}\Delta t, t) \tag{310}$$

And the new cell averages can be computed in the following way

$$Q_i^{t+\Delta t} = \frac{\overline{v}\Delta t}{\Delta x}(Q_{i-1}^t + \frac{1}{2}(\Delta x - \overline{v}\Delta t)\sigma_{i-1}^t) + (1 - \frac{\overline{v}\Delta t}{\Delta x})(Q_i^t - \frac{1}{2}\overline{v}\Delta t\sigma_i^t) = \tag{311}$$
$$Q_i^t - \frac{\overline{v}\Delta t}{\Delta x}(Q_i^t - Q_{i-1}^t) - \frac{1}{2}\frac{\overline{v}\Delta t}{\Delta x}(\Delta x - \overline{v}\Delta t)(\sigma_i^t - \sigma_{i-1}^t)$$

To understand this equation, consider Figure 64.

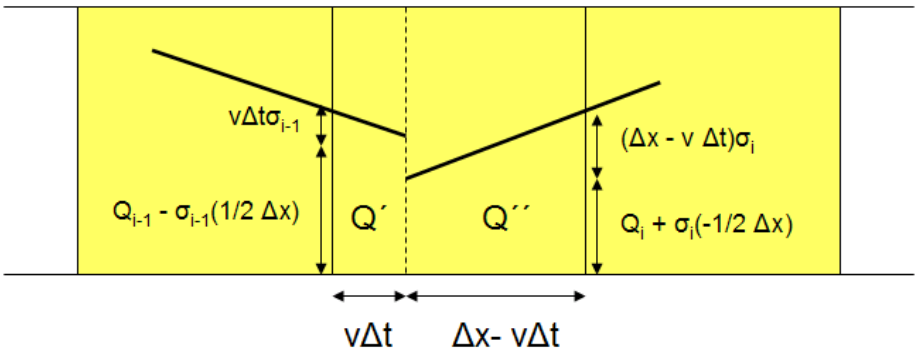

Figure 64: Linear cell average reconstruction

The new cell average, $Q_i^{t+\Delta t}$, is the sum of the two parallelepipeds Q^I and Q^{II}

$$Q_i^{t+\Delta t} = \frac{Q^I + Q^{II}}{\Delta x} \tag{312}$$

$$Q^I = \frac{((Q_{i-1}^t + \sigma_{i-1}(\frac{1}{2}\Delta x)) + (Q_{i-1}^t + \sigma_{i-1}(\frac{1}{2}\Delta x + \overline{v}\Delta t)))\overline{v}\Delta t}{2} = \tag{313}$$
$$(Q_{i-1}^t + (\frac{1}{2}(\Delta x + \overline{v}\Delta t))\sigma_{i-1})\overline{v}\Delta t$$

$$Q^{II} = \frac{((Q_i^t + \sigma_i(-\frac{1}{2}\Delta x)) + (Q_i^t + \sigma_i(\frac{1}{2}\Delta x - \overline{v}\Delta t)))(\Delta x - \overline{v}\Delta t)}{2} = \tag{314}$$
$$(Q_i^t - \frac{1}{2}\overline{v}\Delta t\sigma_i)(\Delta x - \overline{v}\Delta t)$$

and thus

$$Q_i^{t+\Delta t} = \frac{\overline{v}\Delta t}{\Delta x}(Q_{i-1}^t + \frac{1}{2}(\Delta x - \overline{v}\Delta t)\sigma_{i-1}^t) + (1 - \frac{\overline{v}\Delta t}{\Delta x})(Q_i^t - \frac{1}{2}\overline{v}\Delta t\sigma_i^t) \tag{315}$$

We have seen how to compute the new cell averages, but what about the new slopes

$\sigma_i^{t+\Delta t}$? How can we compute them? The simplest choice would be to set $\sigma_i^{t+\Delta t}$ to zero, which would lead us to Godunov´s method seen in the previous chapters. Thus this choice would give us first-order accuracy. To achieve second-order accuracy we choose the nonzero slopes such that they approximate the derivative over the *i-th* grid cell.

$$\sigma_i^t = \begin{cases} \frac{Q_{i+1}^t - Q_{i-1}^t}{2\Delta x} & Fromm \\ \frac{Q_i^t - Q_{i-1}^t}{\Delta x} & Beam - Warming \\ \frac{Q_{i+1}^t - Q_i^t}{\Delta x} & Lax\text{-}Wendroff \end{cases} \tag{316}$$

The cell average update rules that result from the application of these slopes are

Fromm´s Method

$$Q_i^{t+\Delta t} = Q_i^t - \frac{1}{4}\frac{\overline{v}\Delta t}{\Delta x}(Q_{i+1}^t + 3Q_i^t - 5Q_{i-1}^t + Q_{i-2}^t) - \frac{1}{4}(\frac{\overline{v}\Delta t}{\Delta x})^2(Q_{i+1}^t - Q_i^t - Q_{i-1}^t + Q_{i-2}^t) \tag{317}$$

Beam-Warming Method

$$Q_i^{t+\Delta t} = Q_i^t - \frac{1}{2}\frac{\overline{v}\Delta t}{\Delta x}(3Q_i^t - 4Q_{i-1}^t + Q_{i-2}^t) + \frac{1}{2}(\frac{\overline{v}\Delta t}{\Delta x})^2(Q_i^t - 2Q_{i-1}^t + Q_{i-2}^t) \tag{318}$$

Lax-Wendroff Method

$$Q_i^{t+\Delta t} = Q_i^t - \frac{1}{2}\frac{\overline{v}\Delta t}{\Delta x}(Q_{i+1}^t - Q_{i-1}^t) + \frac{1}{2}(\frac{\overline{v}\Delta t}{\Delta x})^2(Q_{i-1}^t - 2Q_i^t + Q_{i+1}^t) \tag{319}$$

Although more accurate, the second-order approximations create oscillations near the discontinuities.
To understand this better, let us consider the following example (Figure 65). If we apply Lax-Wendroff to choose the slopes on this data, then we obtain the piecewise linear function in Figure 66. The situation in the next time step is illustrated in Figure 67. As can be seen, the oscillations are created near the region of the discontinuity.
This observation shows that the slopes proposed above give second-order accuracy for the smooth data, but create oscillations near the discontinuities. We could avoid this problem if we set the slope to zero in the above example, but this will take us back to the first-order accuracy. We are now in front of a dilemma: we want second-order accuracy for the smooth data and we do not want oscillations near discontinuities that the second-order accuracy methods create. A solution could be to apply methods like Lax-Wendroff and Beam-Warming in regions where the solution is smooth and limit the

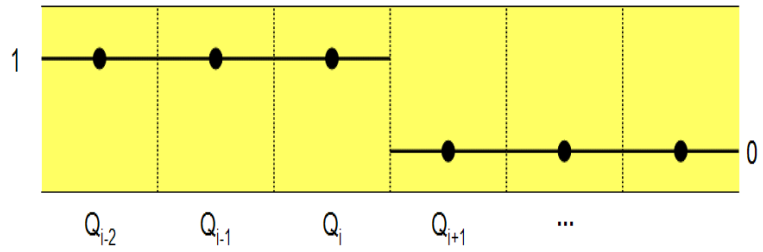

Figure 65: Initial data

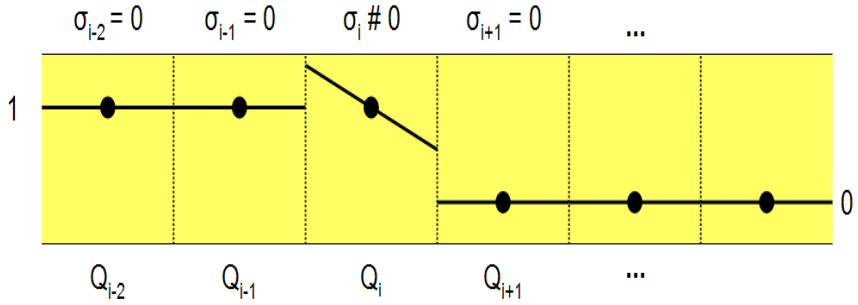

Figure 66: Lax-Wendroff applied to the initial data

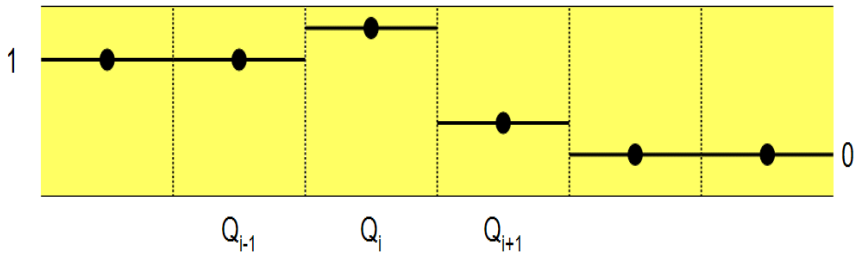

Figure 67: Lax-Wendroff: situation at the next time step

slope in regions where we have the discontinuities in order to avoid oscillations. Methods that incorporate this ideas are called "slope limiters", because they limit the slope. Remains the question of how much to limit the slope. This leads us to the "total variation" concept, which we will not develop further. To summarize it shortly, the total variation measures oscillations in the solution. For the grid function, it is

$$\sum_{i=-\infty}^{+\infty} |Q_i - Q_{i-1}| \tag{320}$$

It turns out that the REA algorithm does not increase the total variation, provided that the first step of the algorithm, the reconstruction step, does not increase it. This is because the evolving and averaging steps do not increase the total variation. Thus we can focus our attention on the reconstruction step. One of the slope-limiter methods that is second-order accurate for smooth functions and does not increase the total variation is the "minmod slope":

$$\sigma_i^t = minmod(\frac{Q_i^t - Q_{i-1}^t}{\Delta x}, \frac{Q_{i+1}^t - Q_i^t}{\Delta x}) \tag{321}$$

where the minmod function is defined as

$$minmod(x,y) = \begin{cases} x & \text{if } |x| < |y| \text{ and } xy > 0 \\ y & \text{if } |x| > |y| \text{ and } xy > 0 \\ 0 & \text{if } xy \leq 0 \end{cases} \tag{322}$$

Although the minmod slope method gives us what we desire, it limits the slope too much. The superbee method allows sharper resolution near a discontinuity while still giving second-order accuracy in the smooth regions. The superbee method compares each one-sided slope with twice the opposite one-sided slope and chooses the maxmod function to decide which slope to take:

$$\sigma_i^1 = minmod(\frac{Q_{i+1}^t - Q_i^t}{\Delta x}, 2(\frac{Q_i^t - Q_{i-1}^t}{\Delta x})) \tag{323}$$

$$\sigma_i^2 = minmod(2(\frac{Q_{i+1}^t - Q_i^t}{\Delta x}), \frac{Q_i^t - Q_{i-1}^t}{\Delta x}) \tag{324}$$

$$\sigma_i^t = maxmod(\sigma_i^1, \sigma_i^2) \tag{325}$$

The superbee limiter approximates the solution sharper near discontinuities in comparison to the minmod slope method.

6.10.1 Flux Limiters

We have derived in the previous section the average update rule

$$Q_i^{t+\Delta t} = Q_i^t - \frac{\overline{v}\Delta t}{\Delta x}(Q_i^t - Q_{i-1}^t) - \frac{1}{2}\frac{\overline{v}\Delta t}{\Delta x}(\Delta x - \overline{v}\Delta t)(\sigma_i^t - \sigma_{i-1}^t) \tag{326}$$

This formula can be manipulated to obtain the flux $F_{i-1/2}^t$ so that we can write the formula in the form

$$Q_i^{t+\Delta t} = Q_i^t - \frac{\Delta t}{\Delta x}(F_{i+1/2}^t - F_{i-1/2}^t) \tag{327}$$

Alternatively, we can compute $F_{i-1/2}^t$ by using the definition of the flux. In either case we obtain for the advection equation

$$F_{i-1/2}^t = \begin{cases} \overline{v}Q_{i-1}^t + \frac{1}{2}\overline{v}(\Delta x - \overline{v}\Delta t)\sigma_{i-1}^t & \text{if } \overline{v} \geq 0 \\ \overline{v}Q_i^t - \frac{1}{2}\overline{v}(\Delta x + \overline{v}\Delta t)\sigma_i^t & \text{if } \overline{v} \leq 0 \end{cases} \tag{328}$$

Until now we associated the slope with the cell. Another idea would be to associate the slope with the interface. Since across the interface $x_{i-1/2}$ we have the jump

$$\Delta Q_{i-1/2}^t = Q_i^t - Q_{i-1}^t \tag{329}$$

we can approximate q_x by dividing the jump by Δx. In this way we can rewrite the flux formula just seen as

$$F_{i-1/2}^t = \overline{v}^+ Q_{i-1}^t + \overline{v}^- Q_i^t + \frac{1}{2}|\overline{v}|(1 - |\frac{\overline{v}\Delta t}{\Delta x}|)\delta_{i-1/2}^t \tag{330}$$

where $\delta_{i-1/2}^t$ is a limited version of $\Delta Q_{i-1/2}^t$. It is interesting to note, that if $\delta_{i-1/2}^t = \Delta Q_{i-1/2}^t$ then we obtain the Lax-Wendroff method. Other second-order methods can be obtained from 330 by an appropriate choice of $\delta_{i-1/2}^t$. *The slope limiter methods can then be reinterpreted as flux-limiter methods by choosing* $\delta_{i-1/2}^t$ *to be a limited version of* $\Delta Q_{i-1/2}^t$ [2].
What we need now is some smoothness measure, θ. There are many possibilities to define it, one of which is

$$\theta_{i-1/2}^t = \begin{cases} \frac{\Delta Q_{i-3/2}^t}{\Delta Q_{i-1/2}^t} & \text{if } \overline{v} > 0 \\ \frac{\Delta Q_{i+1/2}^t}{\Delta Q_{i-1/2}^t} & \text{if } \overline{v} < 0 \end{cases} \tag{331}$$

$$\delta_{i-1/2}^t = \phi(\theta_{i-1/2}^t)\Delta Q_{i-1/2}^t \tag{332}$$

In this way $\theta^t_{i-1/2} \approx 1$ near smooth data (exception made for the case in which we have extrema) and far from 1 near a discontinuity. The function $\phi(\theta)$ is called the flux-limiter function. In general we wish to have $\phi(\theta) \approx 1$ for smooth data and values far from 1 near a discontinuity. If we play around with the values of $\phi(\theta)$ we find interesting results. Suppose for example $\phi(\theta) = 0$. In this case we obtain the upwind method. In the following, some methods are shown that result from assigning particular values to $\phi(\theta)$:

Linear Methods

- Upwind method: $\phi(\theta) = 0$
- Lax-Wendroff method: $\phi(\theta) = 1$
- Beam-Warming method: $\phi(\theta) = \theta$
- Fromm method: $\phi(\theta) = \frac{1}{2}(1 + \theta)$

High Resolution Methods

- Minmod method: $\phi(\theta) = minmod(1, \theta)$
- Superbee method: $\phi(\theta) = max(0, minmod(1, 2\theta), min(2, \theta))$
- van Leer: $\phi(\theta) = \frac{\theta+|\theta|}{1+|\theta|}$

The flux-limiter method with the notation introduced above could be written as

$$\begin{cases} Q_i^{t+\Delta t} = Q_i^t - \nu(Q_i^t - Q_{i-1}^t) - \frac{1}{2}\nu(1-\nu)(\phi(\theta^t_{i+1/2})(Q^t_{i+1} - Q^t_i) - \phi(\theta^t_{i-1/2})(Q^t_i - Q^t_{i-1})) & \text{if } \overline{v} > 0 \\ Q_i^{t+\Delta t} = Q_i^t - \nu(Q_{i+1}^t - Q_i^t) + \frac{1}{2}\nu(1+\nu)(\phi(\theta^t_{i+1/2})(Q^t_{i+1} - Q^t_i) - \phi(\theta^t_{i-1/2})(Q^t_i - Q^t_{i-1})) & \text{if } \overline{v} < 0 \end{cases} \tag{333}$$

where $\nu = \frac{\overline{v}\Delta t}{\Delta x}$ is the Courant number.
We can extend the flux limiter method to systems of equations. First let us take the Lax-Wendroff method and rewrite its flux as

$$F_{i-1/2} = (A^+ Q_{i-1} + A^- Q_i) + \frac{1}{2}|A|(I - \frac{\Delta t}{\Delta x}|A|)(Q_i - Q_{i-1})) \tag{334}$$

This can be viewed as the upwind method together with an additional correction term. Our attention is now focused on how to limit the magnitude of this term according to the data variation.
This is achieved by decomposing the jump

$$Q_i - Q_{i-1} \tag{335}$$

into a sum of jumps according to the eigencomponents

$$\sum_{j=1}^{m} \alpha_{i-1/2}^{j} r^{j} \tag{336}$$

and limit each component separately

$$\widetilde{\alpha}_{i-1/2}^{j} r^{j} = \phi(\theta_{i-1/2}^{j}) \alpha_{i-1/2}^{j} r^{j} \tag{337}$$

where

$$\theta_{i-1/2}^{j} = \begin{cases} \frac{\alpha_{i-3/2}^{j}}{\alpha_{i-1/2}^{j}} & \text{if } \lambda^{j} > 0 \\ \frac{\alpha_{i+1/2}^{j}}{\alpha_{i-1/2}^{j}} & \text{if } \lambda^{j} < 0 \end{cases} \tag{338}$$

In this way we obtain the limited flux function

$$F_{i-1/2} = A^{+} Q_{i-1} + A^{-} Q_{i} + \widetilde{F}_{i-1/2} \tag{339}$$

where

$$\widetilde{F}_{i-1/2} = \frac{1}{2} |A| (1 - \frac{\Delta t}{\Delta x} |A|) \sum_{i=1}^{m} \widetilde{\alpha}_{i-1/2}^{j} r^{j} \tag{340}$$

This can be rewritten as

$$\widetilde{F}_{i-1/2} = \frac{1}{2} \sum_{i=1}^{m} |\lambda^{j}| (1 - \frac{\Delta t}{\Delta x} |\lambda^{j}|) \widetilde{\alpha}_{i-1/2}^{j} r^{j} \tag{341}$$

which results from the definition of the eigenvectors ($|A| r^{j} = |\lambda^{j}| r^{j}$).
Conclusion: each wave is limited independently of other wave families.

6.10.2 Flux Limiter Implementation

The PDE package implements the flux-limiter methods for constant-coefficient linear system. At first glance we can say that what we need is to compute the matrices A^{+}, A^{-} and $|A|$ once and then use them in the formulas given above. However, because we need to decompose the jump $\Delta Q_{i-1/2}$ into waves $\alpha_{i-1/2}^{j} r^{j}$ and we need the λ^{j} too, then it is reasonable to use these informations to compute the new cell averages. The average

update rule can be rewritten, after some algebraic manipulations, as

$$Q_i^{t+\Delta t} = Q_i^t - \frac{\Delta t}{\Delta x}(A^+\Delta Q_{i-1/2} + A^-\Delta Q_{i+1/2}) - \frac{\Delta t}{\Delta x}(\widetilde{F}_{i+1/2} - \widetilde{F}_{i-1/2}) \tag{342}$$

As can be seen from this formula, what we need is $A^+\Delta Q_{i-1/2}$, $A^-\Delta Q_{i-1/2}$ and $\widetilde{F}_{i-1/2}$. All of these terms can be computed by using only $W^j_{i-1/2}$ and λ^j:

$$A^+\Delta Q_{i-1/2} = \sum_{j=1}^{m}(\lambda^j)^+ W^j_{i-1/2} \tag{343}$$

$$A^-\Delta Q_{i-1/2} = \sum_{j=1}^{m}(\lambda^j)^- W^j_{i-1/2} \tag{344}$$

$$\widetilde{F}_{i-1/2} = \frac{1}{2}\sum_{i=1}^{m}|\lambda^j|(1 - \frac{\Delta t}{\Delta x}|\lambda^j|)\widetilde{W}^j_{i-1/2} \tag{345}$$

Now we shall explain how everything is implemented in the package by following the bottom up approach. We start by computing ΔQ and then give it together with the eigenvector matrix R as input to the Riemann block, which will compute the α for each interface (Figure 68). Note that the eigenvector matrix R and the eigenvalues λ^j are

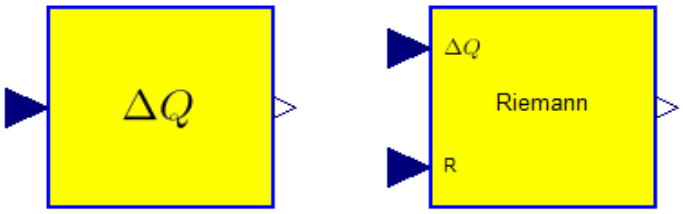

Figure 68: Blocks ΔQ and Riemann

given by the user. Once we have α, we can compute θ according to λ^j. This is done by the θ block (Figure 69). Once θ has been computed it remains to choose the method that we wish to use. If we wish to use the upwind method, we pass then θ to the *Upwind* block, which will give us $\phi(\theta)$ as output (Figure 70). The upwind method and many others are incorporated into a *FluxLimiterSolver* block (Figure 72). As in the *Derivatives* blocks in the *MOL* package, the choice of the method is triggered through the parameter variable, *fls*, in the *WorldModel* block which tells to this block which method we wish to use (Figure 71). Per default the upwind method is used. With this information it is possible to compute the limited alpha, $\widetilde{\alpha}$ and later the corresponding waves, which is done by the blocks *LimitedAlpha* and *WaveP*, respectively (Figure 73). Finally, we pass the waves and eigenvalues to the *FluxLimited* block and the output to the *Fluctuation* block (Figure 74). After summing the + outputs and the − outputs of

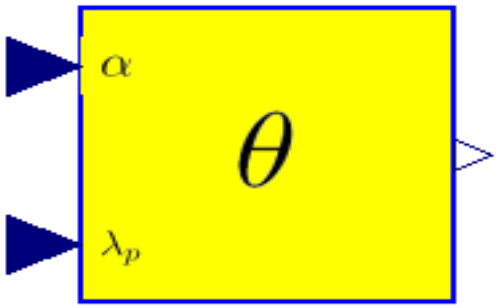

Figure 69: θ block

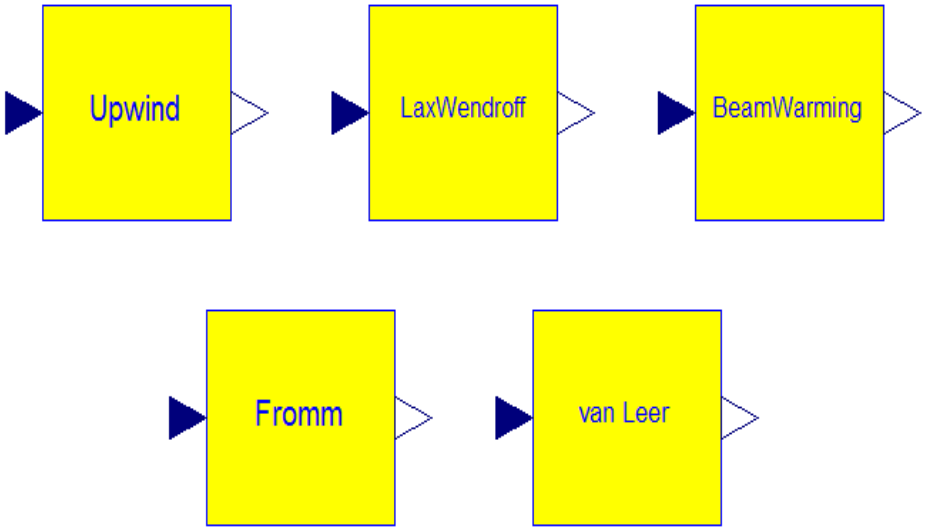

Figure 70: Upwind and other method blocks

```
LinearMethods.Upwind upwind if method == 1;
LinearMethods.LaxWendroff laxWendroff if method == 2;
LinearMethods.BeamWarming beamWarming if method == 3;
LinearMethods.Fromm fromm if method == 4;
HighResolutionMethods.vanLeer vanLeer if method == 5;
```

Figure 71: Portion of FluxLimiterSolver block code

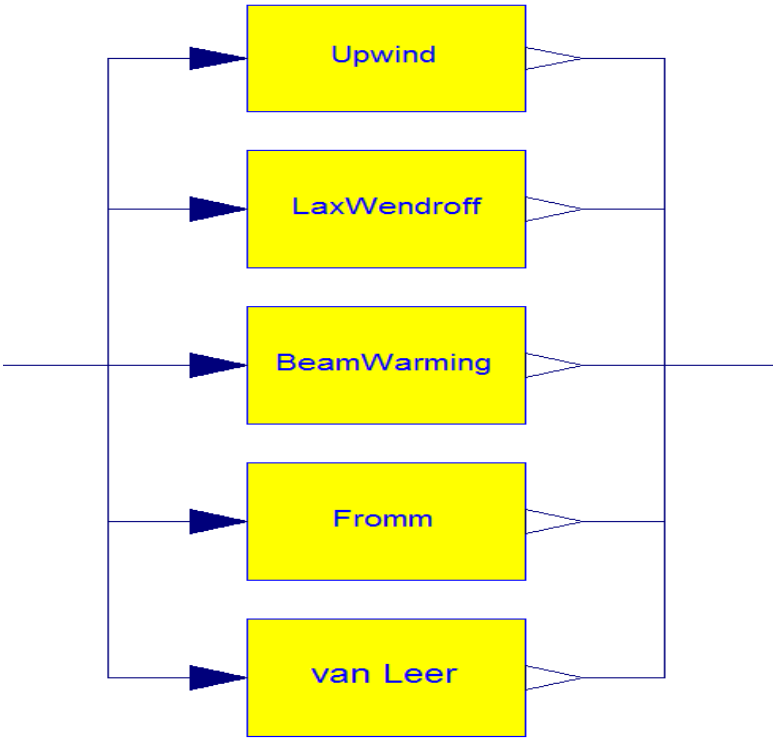

Figure 72: FluxSolver block

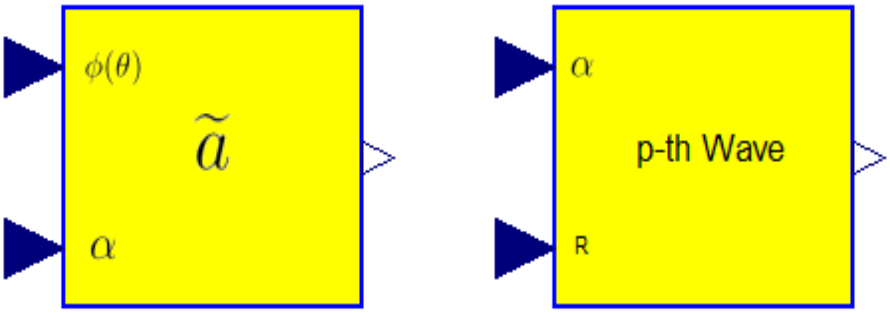

Figure 73: $\widetilde{\alpha}$ and Wave blocks

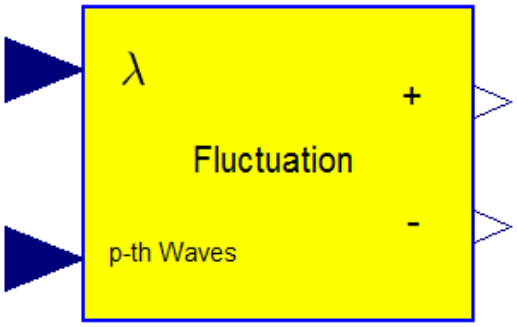

Figure 74: Fluctuation block

the *Fluctuation* blocks, we can pass them to the + and − input of the *FLIntegrator* block (Figure 75) respectively, together with the flux and initial and boundary conditions. To illustrate the use of the blocks, a complete example of the acoustic system is

Figure 75: FLIntegrator block

implemented (Figure 76). To understand better the example, three yellow regions are visualized in the figure. In region 1 the Riemann problem is solved. Region 2 computes the two fluctuations and finally, region 3 computes the flux. In this example the eigenvector matrix and eigenvalues are constant and provided by the user. In Euler system eigenvector matrix and eigenvalues changes with time and space. In order to use the *FluxLimiter* package we might try to provide new eigenvalues and eigenvectors at each time step . For the Euler system of equations the eigenvalues are

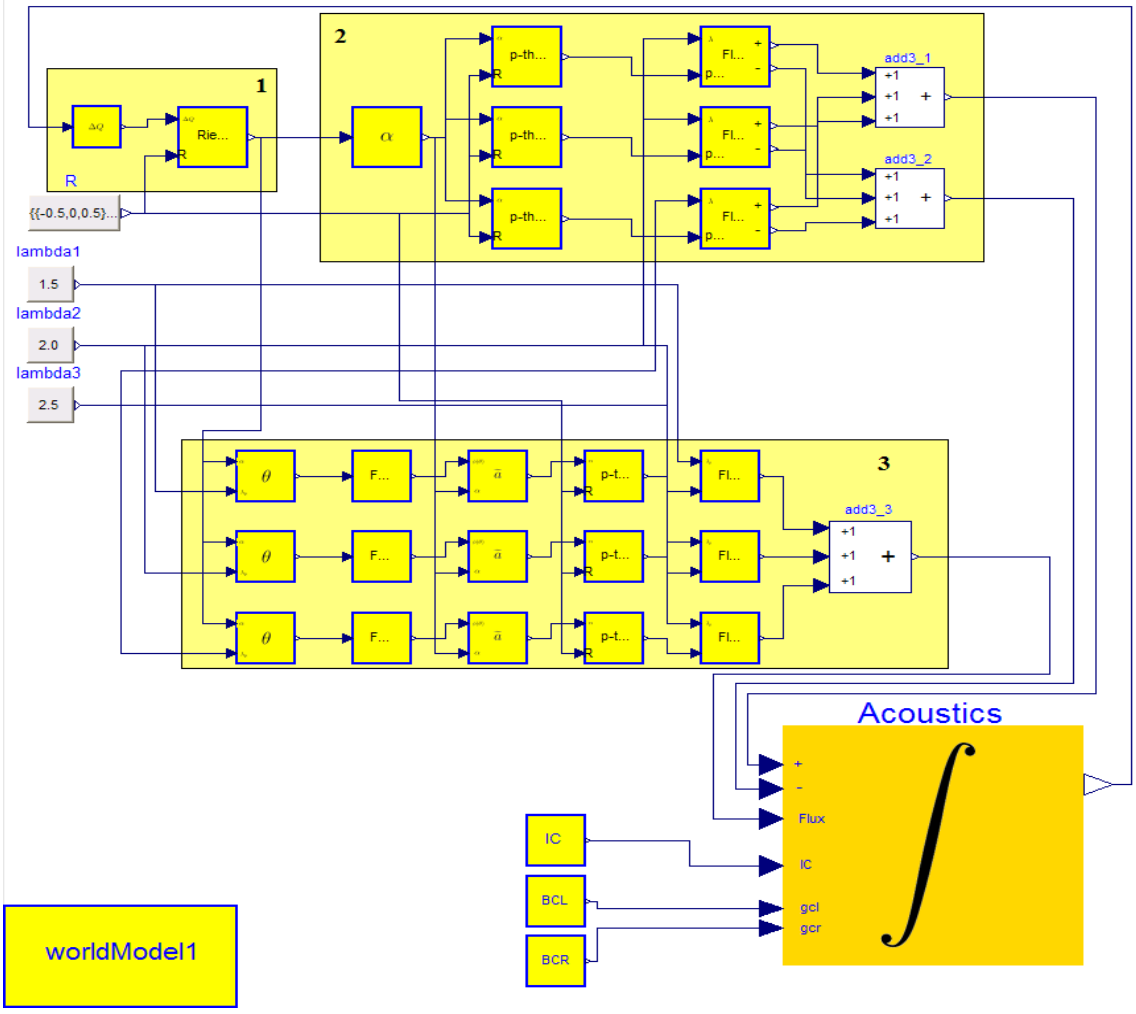

Figure 76: Acoustics example

$$\lambda_1 = v - c \tag{346}$$

$$\lambda_2 = v \tag{347}$$

$$\lambda_3 = v + c \tag{348}$$

and the eigenvector matrix R is

$$R = \begin{pmatrix} 1 & 1 & 1 \\ v-c & v & v+c \\ h - vc & \frac{1}{2}v^2 & h + vc \end{pmatrix} \tag{349}$$

where $c = \sqrt{\frac{\gamma p}{\rho}}$ and $h = \frac{E+p}{\rho}$. Unfortunately the blocks implemented in *FluxLimiter* package cannot be used, because the eigenvector matrix R changes also in space and we

have consequently a different matrix R for each interface that changes with time. For this purpose blocks implemented in *FluxLimiter* package are modified accordingly and can be found, together with Euler example, in $PDE \rightarrow FiniteVolume \rightarrow FluxLimiter \rightarrow Examples \rightarrow EulerSystem$. *FluxLimiter* package is not required for this master thesis and because of lack of time *Acoustics* and *EulerSystem* were not tested in depth.

6.11 Limiter-Free Third Order Logarithmic Reconstruction

In previous chapters we have approximated the cell averages by constant functions or by linear functions with nonzero slope (Figure 77). Other possibilities were to use polynomi-

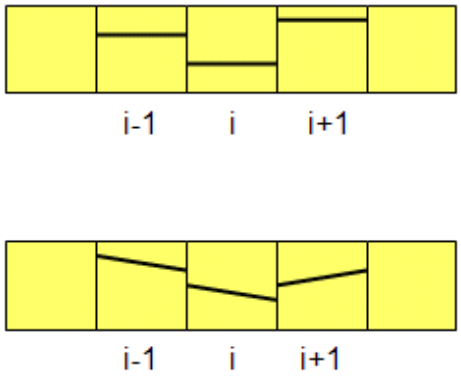

Figure 77: Constant and linear cell average reconstruction

als. However, polynomials are not an optimal choice for hyperbolic problems. As said in [7]: "polynomials are unable to take on very large slopes followed by a flat region, which frequently occur in the piecewise smooth solutions to hyperbolic conservation laws. It is thus reasonable to consider other approximating functions."
In this chapter we consider the reconstruction of an unknown smooth function with *local double logarithmic reconstruction* (LDLR) method. It is *local* because it uses information from only the central cell and its neighbors and it is *double logarithmic* because it uses two logarithmic functions to reconstruct the unknown function. In particular, we reconstruct the values of the flux at the interface $x_{i+1/2}$ in the following way:

$$u^-(x_{i+1/2}) = \overline{u}_i + c_3 h\eta^+(c_1) + c_4 h\eta^+(c_2) \tag{350}$$

$$u^+(x_{i+1/2}) = \overline{u}_{i+1} + c_3 h\eta^-(c_1) + c_4 h\eta^-(c_2) \tag{351}$$

where

$$\eta^+(x) = -\frac{log(1-x)+x}{x^2} \tag{352}$$

$$\eta^-(x) = \frac{(x-1)log(1-x)-x}{x^2} \tag{353}$$

$$c_1 = (1-tol)(1+tol-\frac{2|d_1|^q|d_2|^q+tol}{|d_1|^{2q}+|d_2|^{2q}+tol}) \tag{354}$$

$$c_2 = \frac{c_1}{c_1-1} \tag{355}$$

$$c_3 = \frac{(c_1-1)(d_2(1-c_2)-d_1)}{c_2-c_1} \tag{356}$$

$$c_4 = d_1 - c_3 \tag{357}$$

with $tol = 0.1h^q$ and typically $q = 1.4$. The lateral derivatives are obtained in the following way:

$$d_1 = \frac{\overline{u}_i - \overline{u}_{i-1}}{h},\ d_2 = \frac{\overline{u}_{i+1} - \overline{u_i}}{h} \quad for \quad u^-(x_{i+1/2}) \tag{358}$$

$$d_1 = \frac{\overline{u}_{i+1} - \overline{u}_i}{h},\ d_2 = \frac{\overline{u}_{i+2} - \overline{u}_{i+1}}{h} \quad for \quad u^+(x_{i+1/2}) \tag{359}$$

Once we have reconstructed the values at the interfaces, we can pass these values for example to the unstable numerical flux procedure

$$\widehat{f}_{i+1/2} = \frac{1}{2}(f(u^-(x_{i+1/2},t)) + f(u^+(x_{i+1/2},t))) \tag{360}$$

to obtain the flux at the boundaries and at the end use the average update formula

$$\frac{d\overline{u}_i(t)}{dt} = -\frac{1}{h}(\widehat{f}_{i+1/2} - \widehat{f}_{i-1/2}) \tag{361}$$

to compute the average $\overline{u}_i$ at the next time step. As already seen, the unstable numerical flux is unstable. For this reason, a modified version of it is used, Lax-Friedrichs flux, where some correction term with a parameter α is inserted in order to avoid spurious oscillations.

$$\widehat{f}_{i+1/2} = \frac{1}{2}(f(u^-(x_{i+1/2},t)) + f(u^+(x_{i+1/2},t))) - \frac{1}{2}\alpha(u^+(x_{i+1/2},t) - u^-(x_{i+1/2},t)) \quad (362)$$

The parameter α is not given and must be estimated, which turns out to be a difficult task. This additional term is the so called numerical diffusion and serves to dampen the instabilities that arise in the unstable flux. Although this numerical diffusion is used to avoid instabilities, it introduces much more diffusion than required. This has as a consequence smeared numerical results. However, this problem vanishes when the grid is sufficiently fine.

6.11.1 LDLR Implementation in Modelica

Now that we have seen how the LDLR method works, we can start with the description of the implementation done in Modelica. As already seen the starting point for the computation are the lateral derivatives d_1 and d_2. The lateral derivative $D1minus$ block for the average u^- is shown in Figure 78. As input the $D1minus$ block takes the average

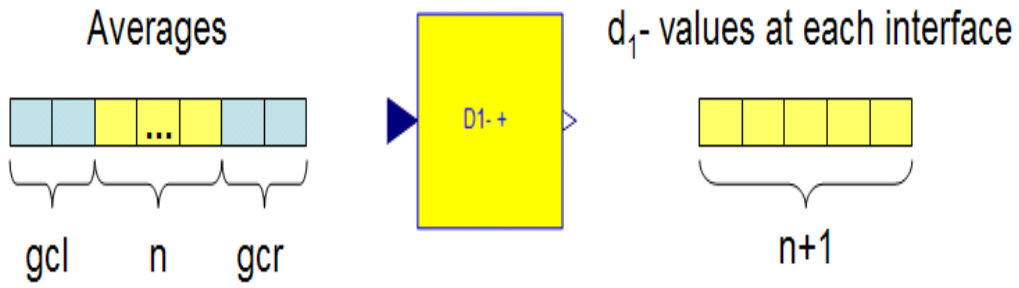

Figure 78: Computing lateral derivatives: D1minus block

vector along with the ghost cells and as output we obtain an $n+1$ vector of d_1 values at each interface. A similar procedure is applied to the blocks $D1plus$, $D2minus$ and $D2plus$. Once we have the lateral derivatives, we can compute the constants $c1$, $c2$, $c3$

and $c4$, which is done by the corresponding blocks (Figure 79). The functions η^- and

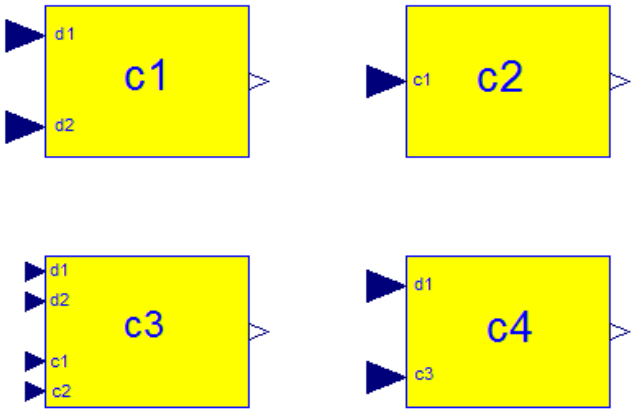

Figure 79: c_1, c_2, c_3 and c_4 blocks

η^+ are implemented by the blocks $n-$ and $n+$ respectively (Figure 80). With blocks

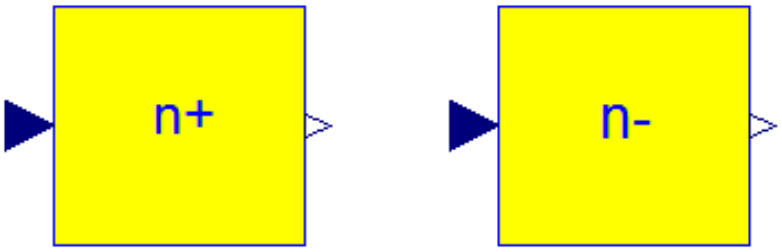

Figure 80: n^- and n^+ blocks

Rminus and *Rplus* we compute

$$c_3 h\eta^+(c_1) + c_4 h\eta^+(c_2) \tag{363}$$

and

$$c_3 h\eta^-(c_1) + c_4 h\eta^-(c_2) \tag{364}$$

respectively and the reconstruction is completed with the blocks u_minus and u_plus (Figure 81). In order to simplify the construction, additional blocks are implemented:

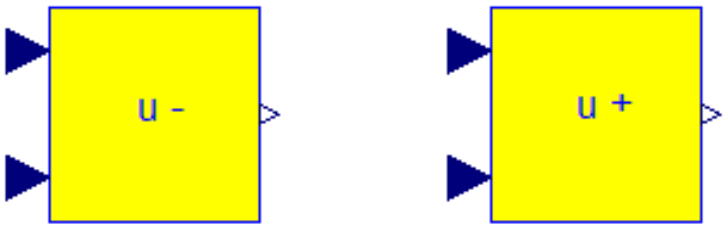

Figure 81: u^- and u^+ blocks

LDLRminus and LDLRplus. At the end we can pass the fluxes and reconstructed values to the Lax-Friedrichs flux block for example (see next section).

6.11.2 LDLR Implementation of the Euler System with Lax-Friedrichs Flux

Let us now look in detail how to implement the Euler equation with the LDLR method. We start with the first equation, the continuity equation. In the continuity equation the average of interest is density ρ and the flux is ρv. We first reconstruct by using LDLR the values ρ^- and ρ^+ by using the blocks u^- and u^+ respectively. We do the same for the momentum. After we have reconstructed the densities ρ^- and ρ^+ and velocities v^- and $v+$ at the interfarce $x_{i+1/2}$ we can compute the two fluxes $\rho^- v^-$ and $\rho^+ v^+$ at this interface. The reconstruction value process is shown in Figure 82. With

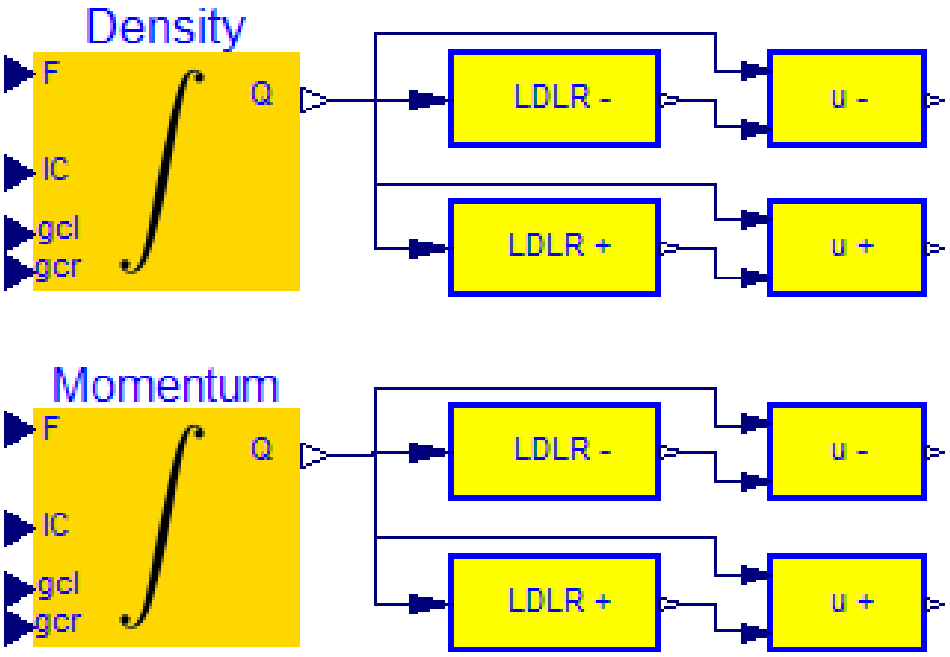

Figure 82: LDLR Reconstruction of density and momentum

the reconstructed value we can construct the fluxes, which are passed together with the reconstructed density values to the Lax-Friedrich block (Figure 83) For the momentum equation we have $\rho v^2 + p$ as flux. The reconstruction process of the momentum flux is shown in Figure 84. To compute the values of the pressure p^+ and p^- we use the formulae

$$p^+ = (\gamma - 1)(E^+ - \rho^+ \frac{(v^+)^2}{2}) \tag{365}$$

$$p^- = (\gamma - 1)(E^- - \rho^- \frac{(v^-)^2}{2}) \tag{366}$$

Finally, we reconstruct the energy flux $(E+p)v$ (Figure 85). Now that we have the fluxes, we pass them to the Lax-Friedrichs flux blocks along with the corresponding averages and connect the output of the Lax-Friedrichs flux blocks to the corresponding average

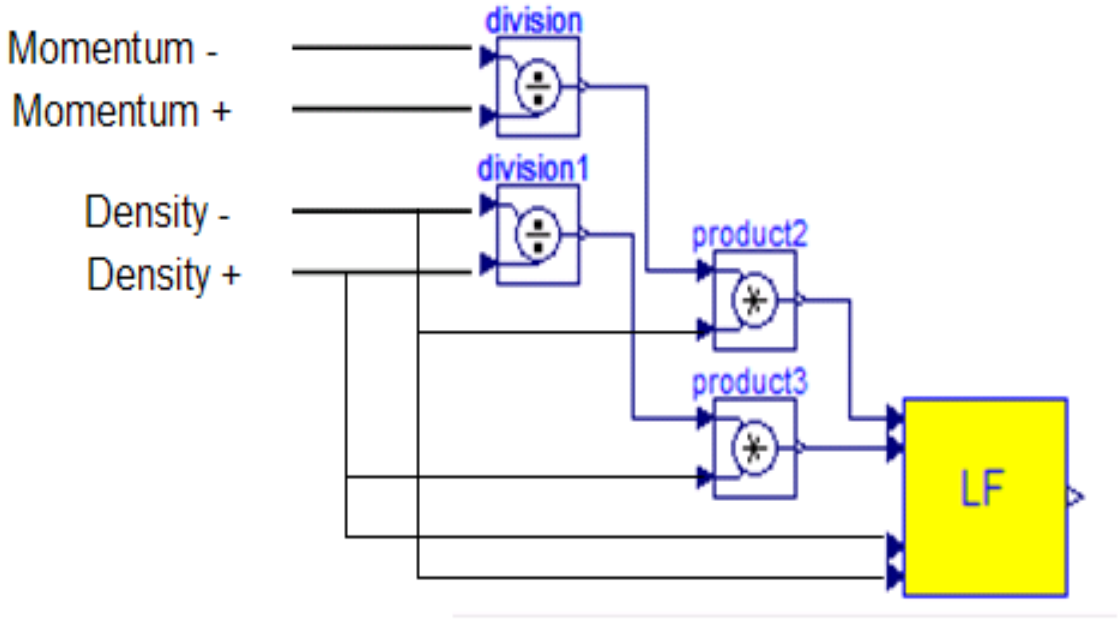

Figure 83: LDLR: Continuity flux

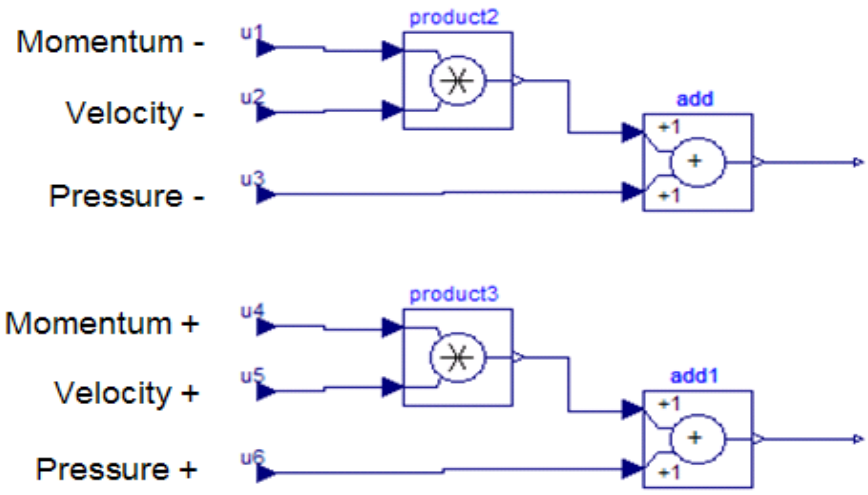

Figure 84: LDLR: Momentum flux

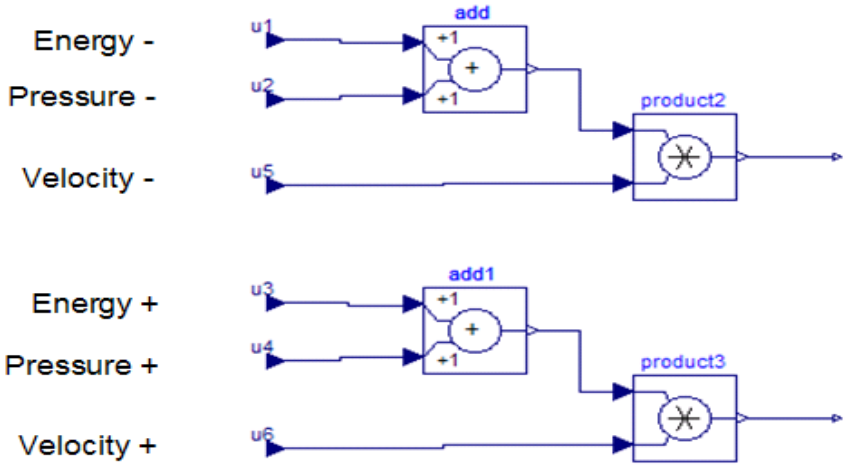

Figure 85: Energy flux

integrator. The construction of the Euler system is now complete (Figure 86) and we can start to simulate it.

6.11.3 Roe´s Flux

We have seen in the previous section how to solve the Euler system by using the Lax-Friedrich flux. Many other flux methods could be used instead.
In the following we will use another numerical flux: Roe´s flux. We have already seen that when decomposing a hyperbolic system of equations with eigenvalue decomposition, we obtain a system of decoupled advection equations each with the velocity λ_i. This information can be used to implement upwind schemes. In Roe´s flux method the flux difference at $x_{i+1/2}$ is decomposed in the following way

$$\widehat{f}^{+} - \widehat{f}^{-} = W^{+} + W^{-} \tag{367}$$

where the traveling waves are decomposed according to velocities λ_i as

$$W^{+} = \sum_{\lambda_i > 0} a_i W_i \tag{368}$$

$$W^{-} = \sum_{\lambda_i \leq 0} a_i W_i \tag{369}$$

for i = 1, ..., 3 and the strengths a_i together with waves W_i are computed as

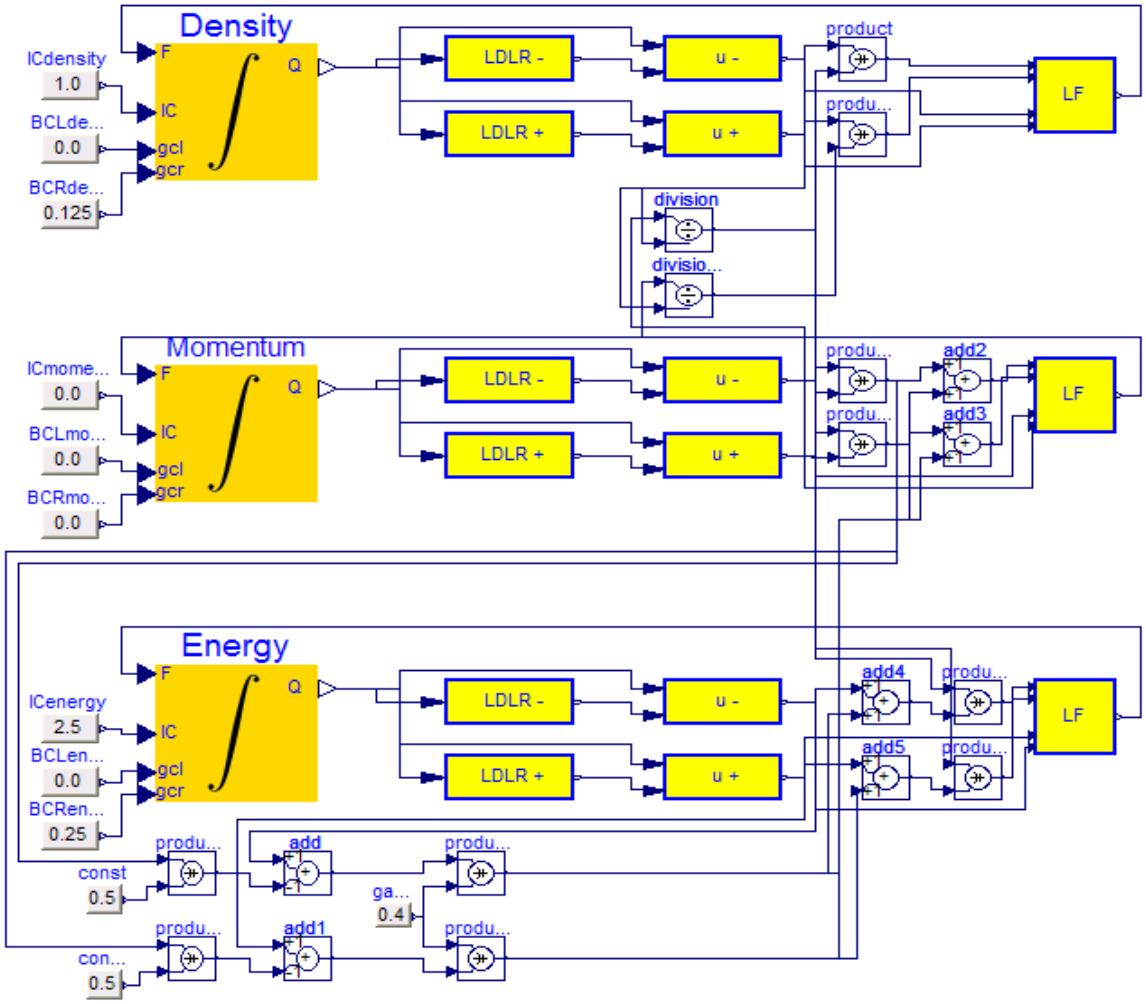

Figure 86: Euler system with LDLR reconstruction and Lax-Friedrichs flux

$$a_1 = \frac{1}{2\widetilde{a}^2}(\triangle p - \widetilde{\rho}\widetilde{a}\triangle u) \tag{370}$$

$$a_2 = \frac{1}{\widetilde{a}^2}(\widetilde{a}^2\triangle\rho - \triangle p) \tag{371}$$

$$a_3 = \frac{1}{2\widetilde{a}^2}(\triangle p + \widetilde{\rho}\widetilde{a}\triangle u) \tag{372}$$

$$W_1 = \begin{pmatrix} 1 \\ \widetilde{v} - \widetilde{a} \\ \widetilde{h} - \widetilde{v}\widetilde{a} \end{pmatrix} \tag{373}$$

$$W_2 = \begin{pmatrix} 1 \\ \widetilde{v} \\ \frac{1}{2}\widetilde{v}^2 \end{pmatrix} \tag{374}$$

$$W_3 = \begin{pmatrix} 1 \\ \widetilde{v} + \widetilde{a} \\ \widetilde{h} + \widetilde{v}\widetilde{a} \end{pmatrix} \tag{375}$$

where

$$\triangle p = p^+ - p^- \tag{376}$$

$$\triangle \rho = \rho^+ - \rho^- \tag{377}$$

$$\triangle v = v^+ - v^- \tag{378}$$

and

$$\lambda_1 = \widetilde{v} - \widetilde{a} \tag{379}$$

$$\lambda_2 = \widetilde{v} \tag{380}$$

$$\lambda_3 = \widetilde{v} + \widetilde{a} \tag{381}$$

and the Roe´s averages $\tilde{\rho}$, $\tilde{v}$, $\tilde{h}$ and $\tilde{a}$ are found by

$$\widetilde{\rho} = \sqrt{\rho^+ \rho^-} \tag{382}$$

$$\widetilde{v} = \frac{\sqrt{\rho^+}v^+ + \sqrt{\rho^-}v^-}{\sqrt{\rho^+} + \sqrt{\rho^-}} \tag{383}$$

$$\widetilde{h} = \frac{\sqrt{\rho^+}h^+ + \sqrt{\rho^-}h^-}{\sqrt{\rho^+} + \sqrt{\rho^-}} \tag{384}$$

$$\widetilde{a} = \sqrt{(\gamma - 1)(\widetilde{h} - \frac{1}{2}\widetilde{v}^2)} \tag{385}$$

The logarithmic reconstruction procedure can be summarized in Figure 87.

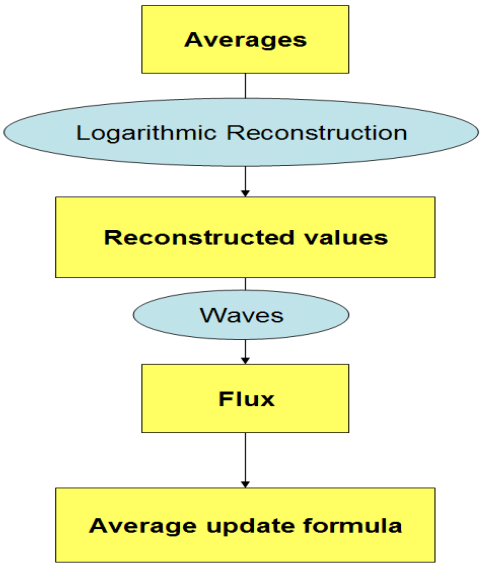

Figure 87: Logarithmic Reconstruction

6.11.4 LDLR Implementation of the Euler System with Roe´s Flux

In the Roe´s flux we start by computing the averages $\widetilde{\rho}$, $\widetilde{v}$, $\widetilde{h}$ and $\widetilde{a}$. This is done by the *Daverage*, *Vaverage*, *Haverage* and *Aaverage* blocks respectively (Figure 88). These

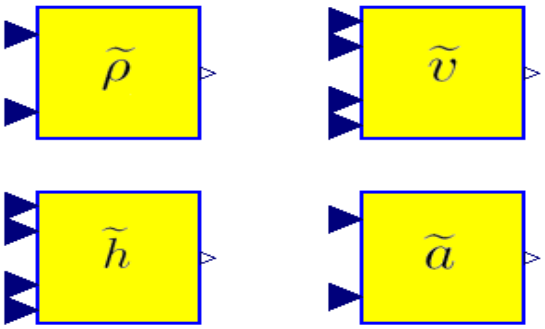

Figure 88: Average blocks

blocks receive as inputs the required information, for instance, the *Daverage* block receives as input densities ρ^- and ρ^+ at each interface, and gives as output the averages at each interface, $\sqrt{\rho^+\rho^-}$ in *Daverage* case. By using this average we can compute the wave velocities λ_i and the waves W_i. Blocks *Lambdas* and *Waves* achieve this task (Figure 89). The output of the *Waves* block are three $mxn+1$ matrices. The first matrix contains the W_1 waves for each interface, the second the W_2 waves for each interface, and the third the W_3 waves for each interface. Each wave is an m-component vector.

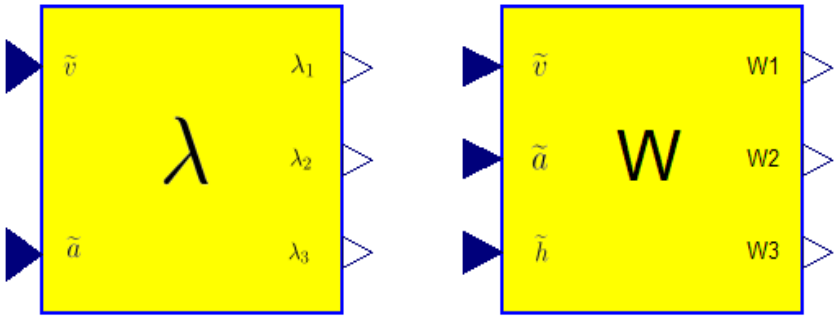

Figure 89: Lambdas and Waves blocks

The strengths a_1, a_2 and a_3 of the waves are computed by the a block (Figure 90). With

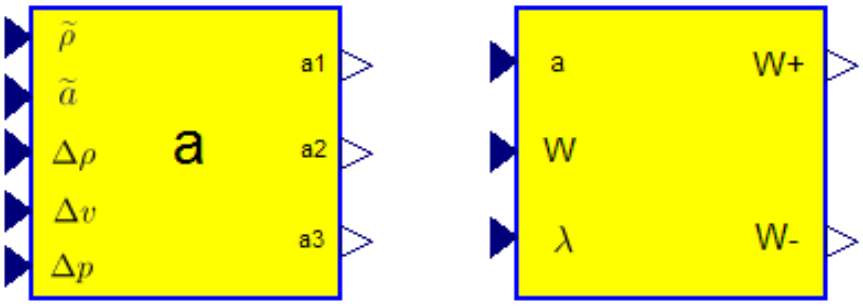

Figure 90: a and FluxDiff blocks

λ_i, a_i and W_i we can, as already explained, decompose the traveling waves:

$$W^+ = \sum_{\lambda_i > 0} a_i W_i \tag{386}$$

$$W^- = \sum_{\lambda_i \leq 0} a_i W_i \tag{387}$$

This task is achieved by the $FluxDiff$ block. Each $FluxDiff$ block produces one wave: either W^+ or W^-. In the Euler system we have three waves, that means that we need three $FluxDiff$ blocks, each of which will give us W^+ or W^- according to the λ. At the end we must sum the outputs in order to obtain

$$W^+ + W^- \tag{388}$$

Finally we give this sum to the integrator which evaluate the update rule. The complete

construction of the Euler system with the LDLR reconstruction and Roe´s numerical flux is illustrated in Figure 91.

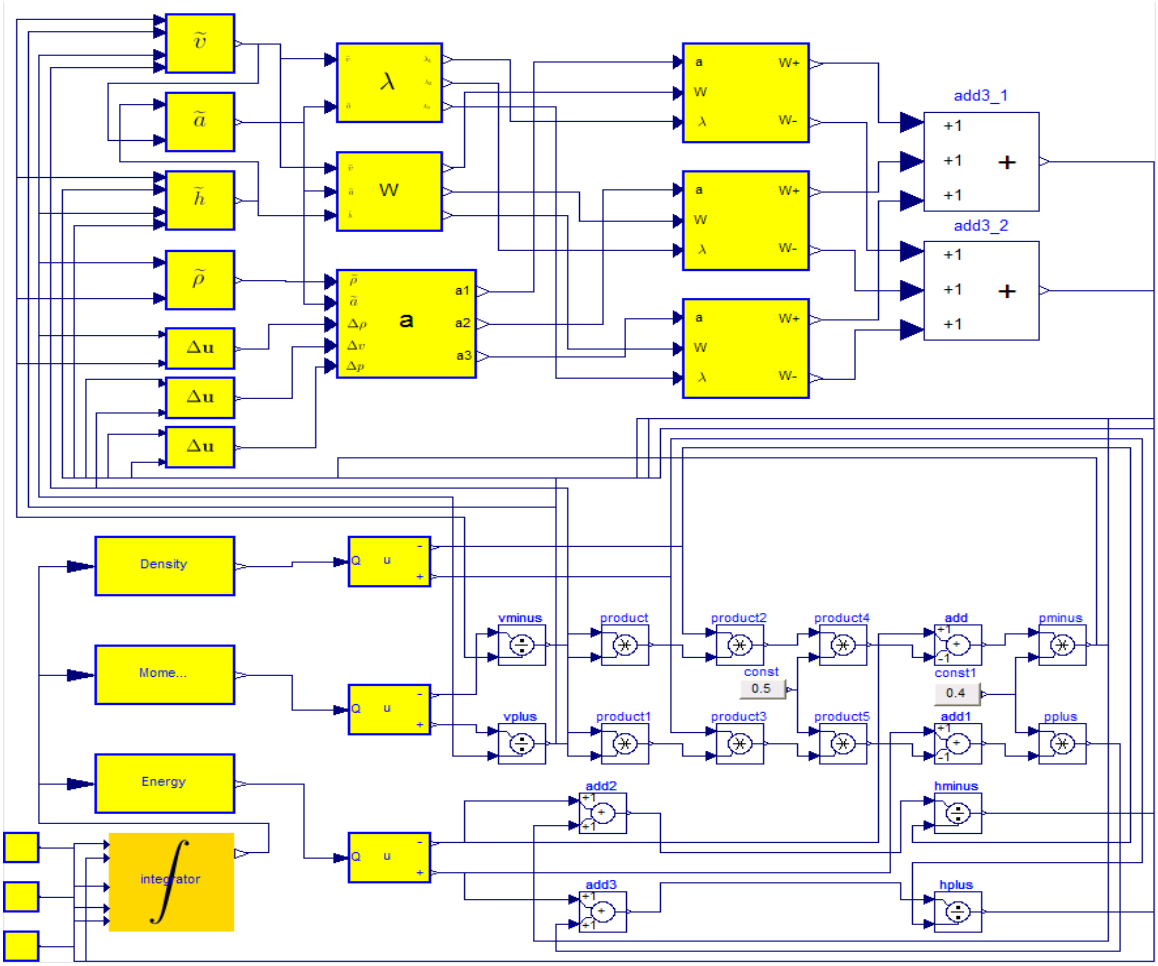

Figure 91: Euler system with LDLR reconstruction and Roe´s flux

6.12 Examples

The LDLR implementation of the Euler equations by using Lax-Friedrichs and Roe´s fluxes was already shown in the previous chapters. We illustrate in the following some other examples implemented by using the logarithmic reconstruction method. All examples can be found under $FiniteVolume \longrightarrow Examples$. As in the MOL chapter, we start with the simplest PDE implemented, the advection equation.

6.12.1 Advection Equation

The advection PDE was explained in the MOL chapter. Its implementation with the Finite Volume Method by using LDLR can be seen in Figure 92. Lax-Friedrichs flux

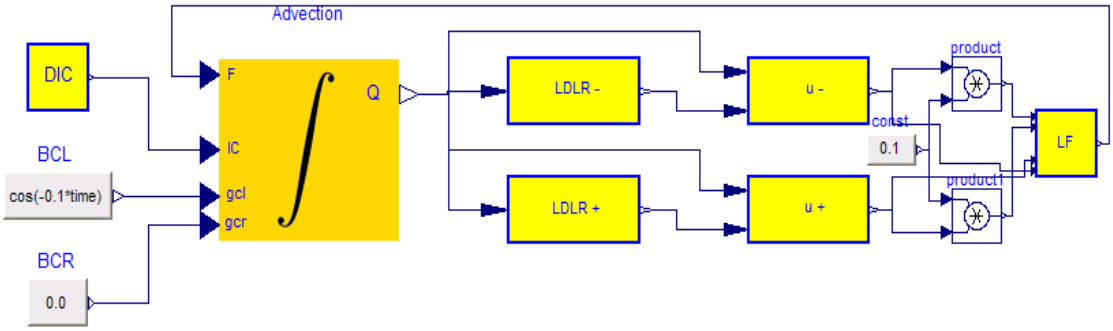

Figure 92: Advection PDE implementation by using FVM with LDLR and Lax-Friedrich´s flux

was used, with $\alpha = 0.2$. Other values of α lead to unsatisfactory results. In general, the results are not as good as in the MOL implementation. Forty cells were used to achieve good results (Figure 93). And the last cells presented quite accentuated dissipation.

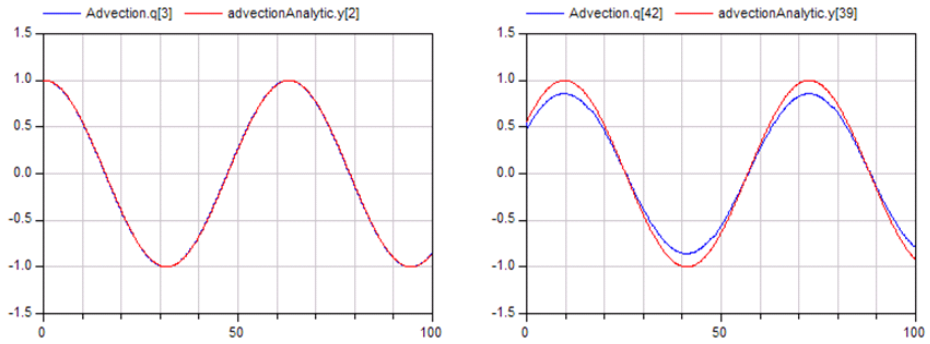

Figure 93: Analytical (red) and FVM (blue) solution of the advection equation for the first and last cells (40 cells were used)

This could be caused by the Lax-Friedrichs flux. Conclusion: Advection equation is better simulated with the method of lines. The finite volume method with LDLR and Lax-Friedrichs flux needs considerably more cells in order to achieve comparable results. However, if we implement the advection equation with the upwind method (Figure 94), the results are better. In Figure 95 the analytical and numerical solution for the first and last cells (10 cells were used) is shown. The accuracy is better than in the case of the Lax-Friedrichs method, moreover, there is less dissipation in the solution of the last cell. Additionally, we must not search for an optimal parameter when using the upwind

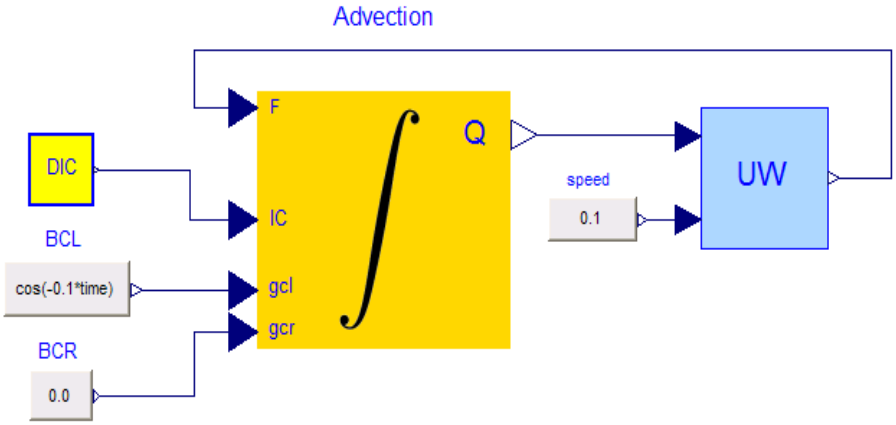

Figure 94: Advection PDE implementation by using FVM with LDLR and Upwind flux

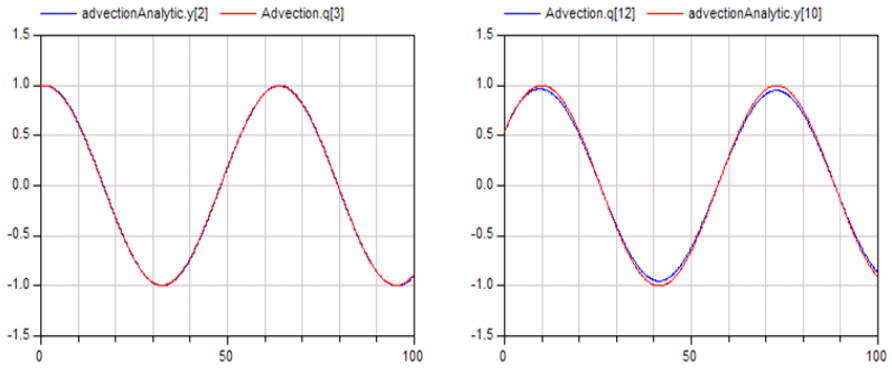

Figure 95: Advection PDE implementation by using FVM with LDLR and Upwind flux

method.
A third implementation of the advection equation uses Lax-Wendroff flux. The results are however not as satisfactory as in the case of the upwind flux.

6.12.2 Diffusion Equation

The diffusion problem was already discussed in the Method of Lines Chapter. In Figure 96 the FVM implementation of the diffusion equation can be seen. The Diffusion Flux

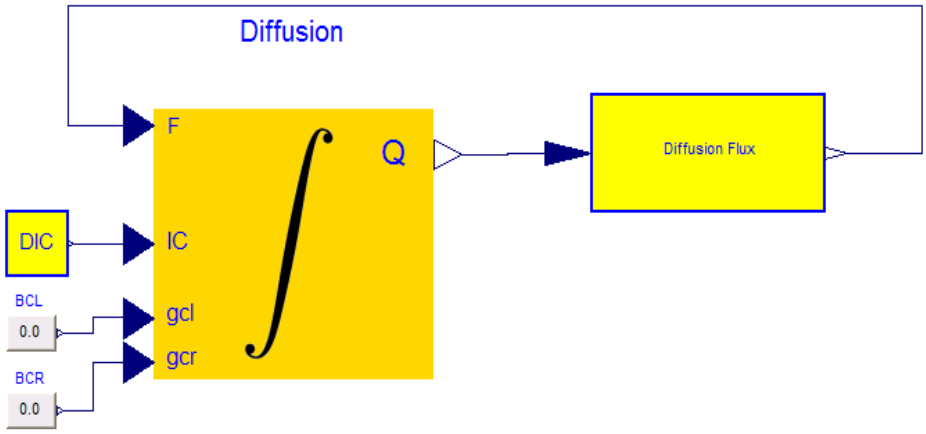

Figure 96: Diffusion PDE implementation with FVM

block implements the numerical diffusion flux discussed in the beginning of the Finite Volume Method Chapter. In Figure 97 we can see the comparison of the analytical solution and the FVM solution of the diffusion PDE for the second and middle cells of the domain, where 10 cells were used. As can be seen, ten cells are already enough to reach a good approximation. The situation with 40 cells is illustrated in Figure 98.

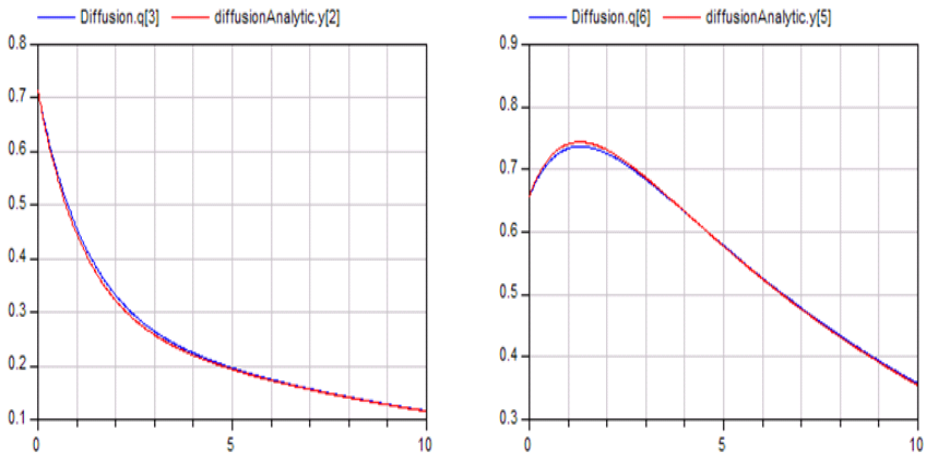

Figure 97: Comparison of the analytical and FVM solution of the diffusion PDE for the first and middle cells of the domain (10 cells were used)

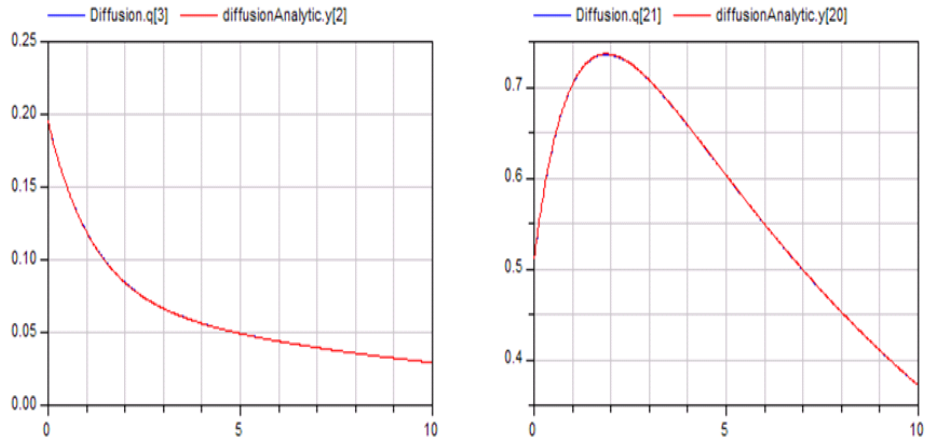

Figure 98: Comparison of the analytical and FVM solution of the diffusion PDE for the first and middle cells of the domain (10 cells were used)

6.12.3 Burger´s Equation

Burger´s equation was also implemented with LDLR reconstruction and Lax-Friedrichs flux ($\alpha = 0.9$). The corresponding implementation can be seen in Figure 99. The Finite

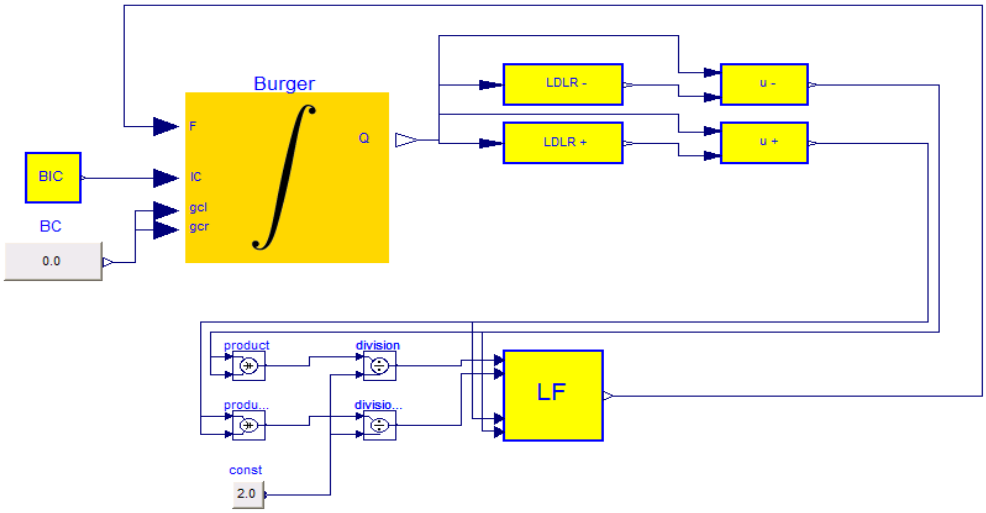

Figure 99: Burger PDE implementation by using FVM with LDLR and Lax-Friedrichs flux

Volume Methods solution of the first and middle cells compared with the respective analytical solution in the case of 10 and 40 cells is shown in Figure 100. As could be

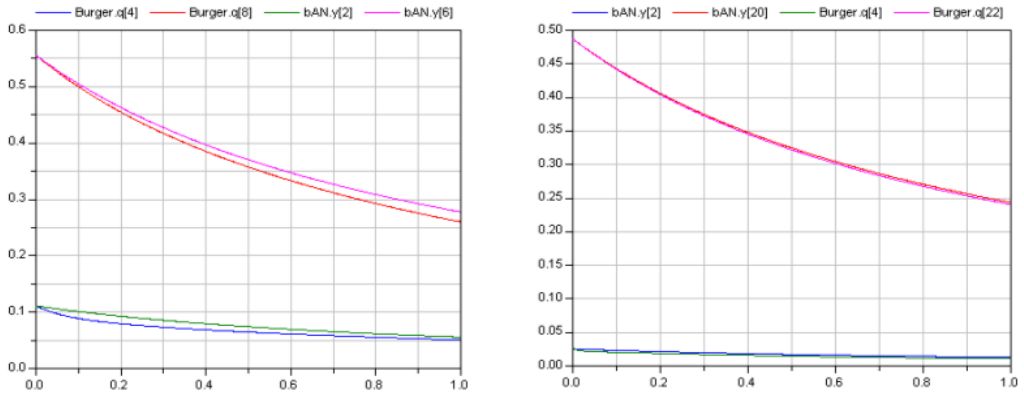

Figure 100: Numerical versus analytical solution of the Burger´s equation for the first and middle cells with 10 (left) and 40 (right) cells

expected, the accuracy increases when more cells are used.

6.12.4 Euler with Lax-Friedrichs and Roe´s Fluxes

Simulation of the Euler equations requires considerably more cells than other examples. To compare the two implementations, Euler with Lax-Friedrichs flux and Euler with Roe´s flux, sixty cells were used. In Figure 101 the comparison of the density for the 59-th cell and 61-th cell can be seen. In Figure 102 the same comparison in the case

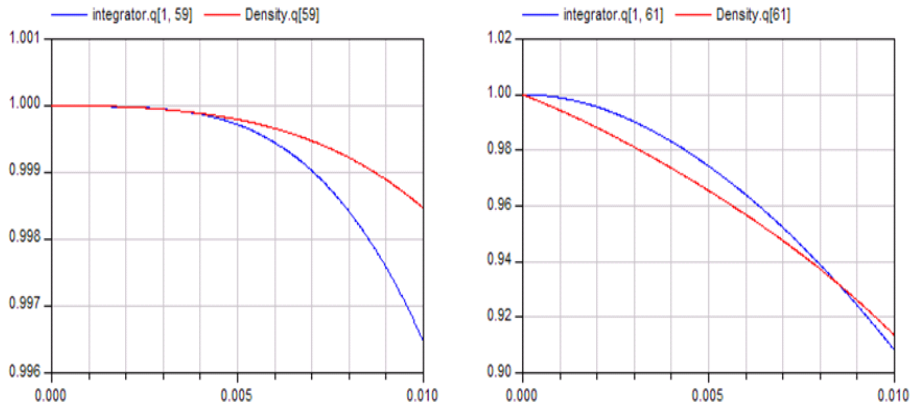

Figure 101: Comparison of density with Euler Lax-Friedrichs (red) and Euler Roe´s (blue) fluxes (60 cells used)

of momentum for the 59-th cell is done. By using more cells, for example hundred, the

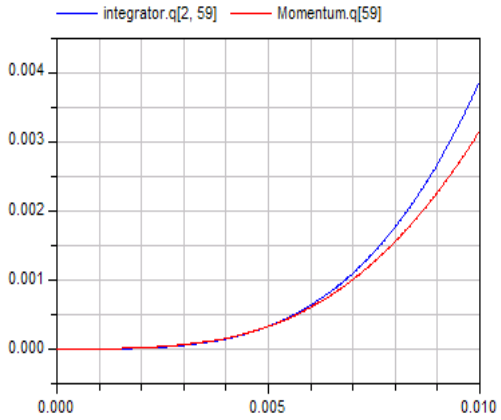

Figure 102: Comparison of momentum with Euler Lax-Friedrichs (red) and Euler Roe´s (blue) fluxes (60 cells used)

situation is almost the same. Another problem are the big oscillations in the Roe´s implementation. A possible cause might be the violation of CFL condition (see Section 6.14).

6.12.5 Buckley-Leverett Equation

Buckley-Leverett equation is implemented with LDLR method in FiniteVolume→Examples. The initial condition used is

$$u(x, 0) = 1.0 \tag{389}$$

and boundary conditions are

$$u(x_1, t) = 0.0 \tag{390}$$

$$u(x_n, t) = 0.0 \tag{391}$$

The FVM implementation with LDLR and Lax-Friedrich´s flux is illustrated in Figure 103. In figure 104 we can see the MOL and FVM solution of the Buckley-Leverett PDE

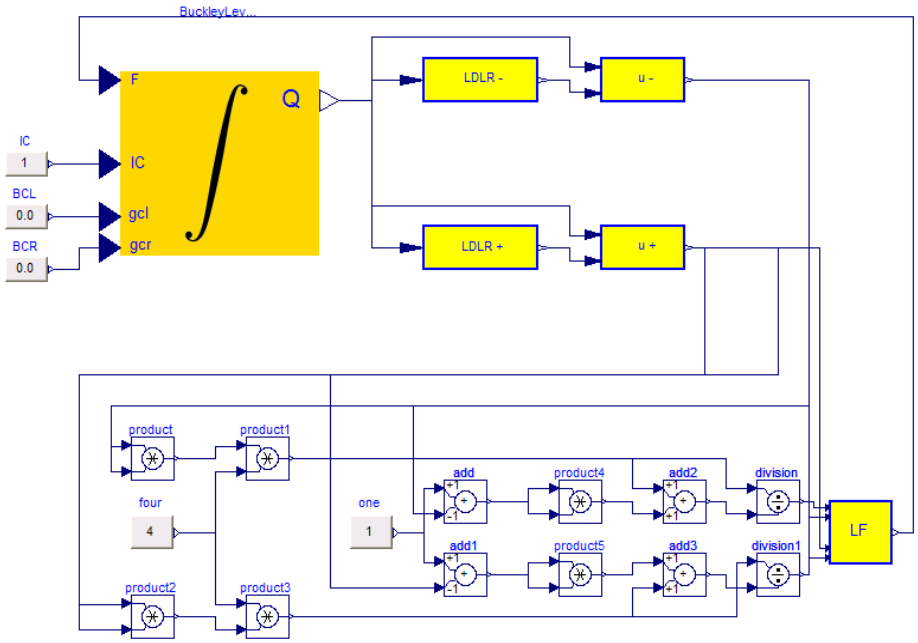

Figure 103: Buckley-Leverett PDE implementation with FVM (with LDLR and Lax-Friedrich´s flux)

for the first and middle cells of the domain (10 cells were used). The difference is big and by refining the grid the situation is not better. One reason could be the wrong α parameter in Lax-Friedrich´s flux.

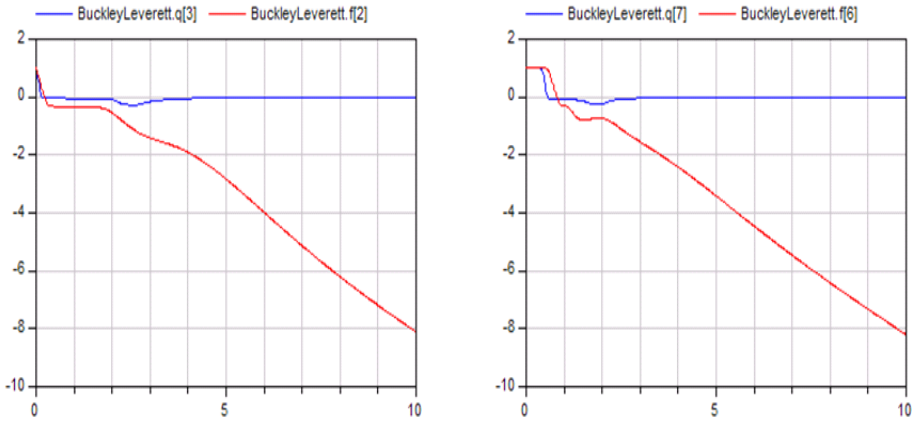

Figure 104: MOL and FVM solution of Buckley-Leverett PDE with LDLR and Lax-Friedrich´s flux

6.13 General Block

The idea of the *GeneralBlock* (Figure 105) is to incorporate two numerical methods, MOL and FVM, into a single block. This way we assure more transparence to the user, that can now just use one single block and choose the desired numerical method, without caring about details of each method. For this purpose, *GeneralBlock* contains

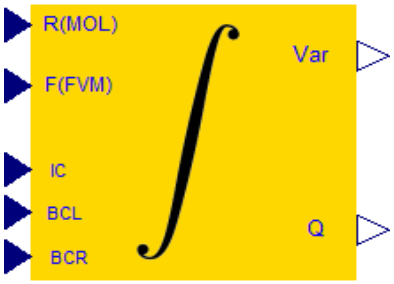

Figure 105: General Block Integrator

two integrators, the MOL and FVM integrators. Which one must be used is specified by the user through the parameter *integrator*: integrator = 1 for MOL and integrator = 2 for FVM. In Figure 106 the implementation of the advection equation with the *GeneralBlock* is illustrated. In this example we chose MOL method. For this purpose the R input and Var output of the block is used, just as in the MOL Integrator. If we wish to solve advection equation with the FVM method (Figure 107), we just set the *integrator* parameter to 2 and use the F input and Q output of the block.

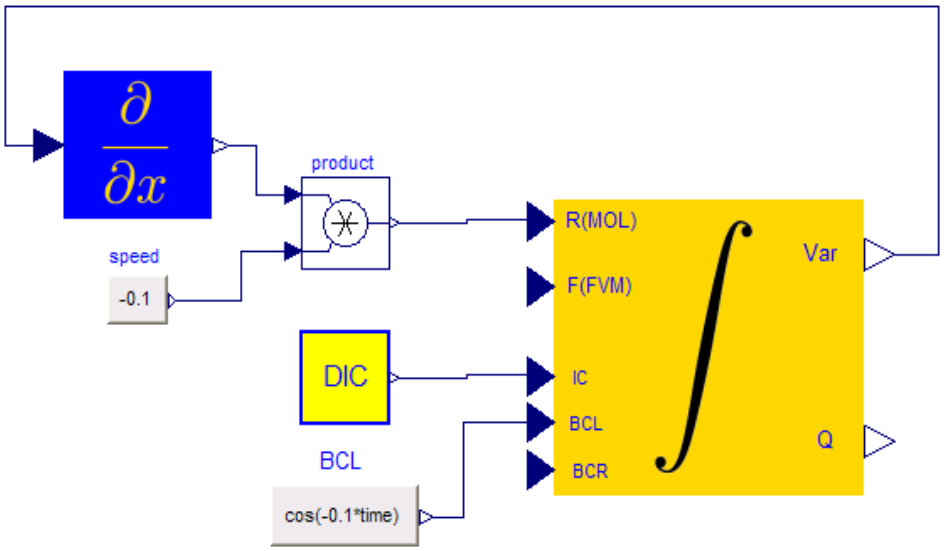

Figure 106: MOL implementation of the advection equation with *GeneralBlock*

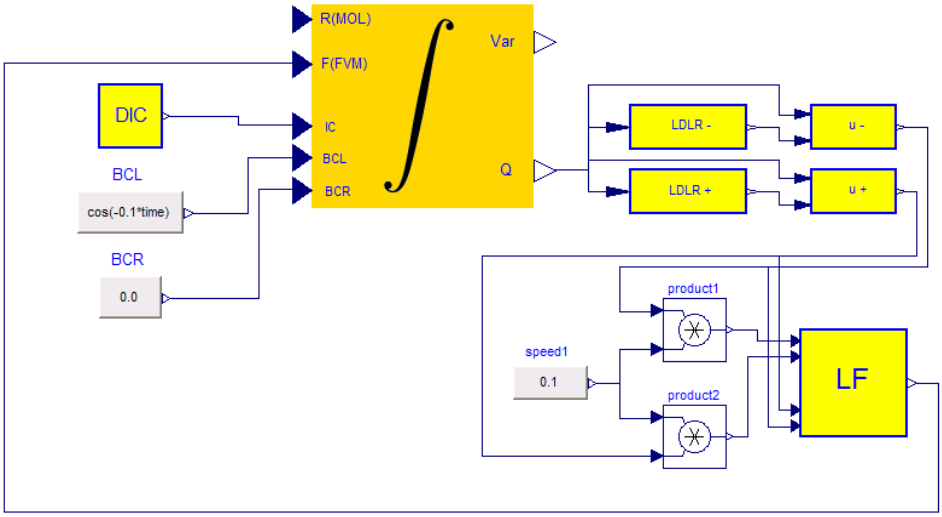

Figure 107: FVM implementation of the advection equation with *GeneralBlock*

6.14 Courant-Friedrich-Lewy Condition

When studying numerical methods we must consider many criteria, such as [8]:

- **Consistency**

  The discretization of a PDE should become exact as the mesh size tends to zero (truncation error should vanish).

- **Stability**

  Numerical errors which are generated during the solution of discretized equations should not be magnified.

- **Convergence**

  The numerical solution should approach the exact solution of the PDE and converge to it as the mesh size tends to zero.

- **Conservation**

  Underlying conservation laws should be respected at the discrete level (artificial sources/sinks are to be avoided).

- **Boundedness**

  Quantities like densities, temperatures, concentrations etc. should remain non-negative and free of spurious wiggles.

The theory is very complicated and we will not analyze all these criteria in this thesis. We will however look at the Courant-Friedrich-Lewy (CFL) condition.
The CFL condition is a necessary condition that must be satisfied by the finite volume method if we expect it to be stable and converge to the solution of the differential equation as the grid is refined [2].
Consider the scalar advection equation

$$q_t + \overline{v} q_x = 0 \tag{392}$$

with the positive velocity $\overline{v}$ and say we are using the finite volume method where we need only three values Q_{i-1}^t, Q_i^t and Q_{i+1}^t in order to compute the cell average $Q_i^{t+\Delta t}$ at the next time step. In this case the exact solution moves a distance $\overline{v}\Delta t$ over one time step. In Figure 108 we see two cases: one in which this distance is less than the cell size and one in which it is greater than the cell size. If $\overline{v}\Delta t < \Delta x$, that is if the distance moved by the exact solution is less than the cell size, then it makes sense to use the values Q_{i-1}^t and Q_i^t to compute the flux at the interface $x_{i-1/2}$. If however, $\overline{v}\Delta t > \Delta x$, that is if the distance moved by the exact solution is greater than the cell size, then we

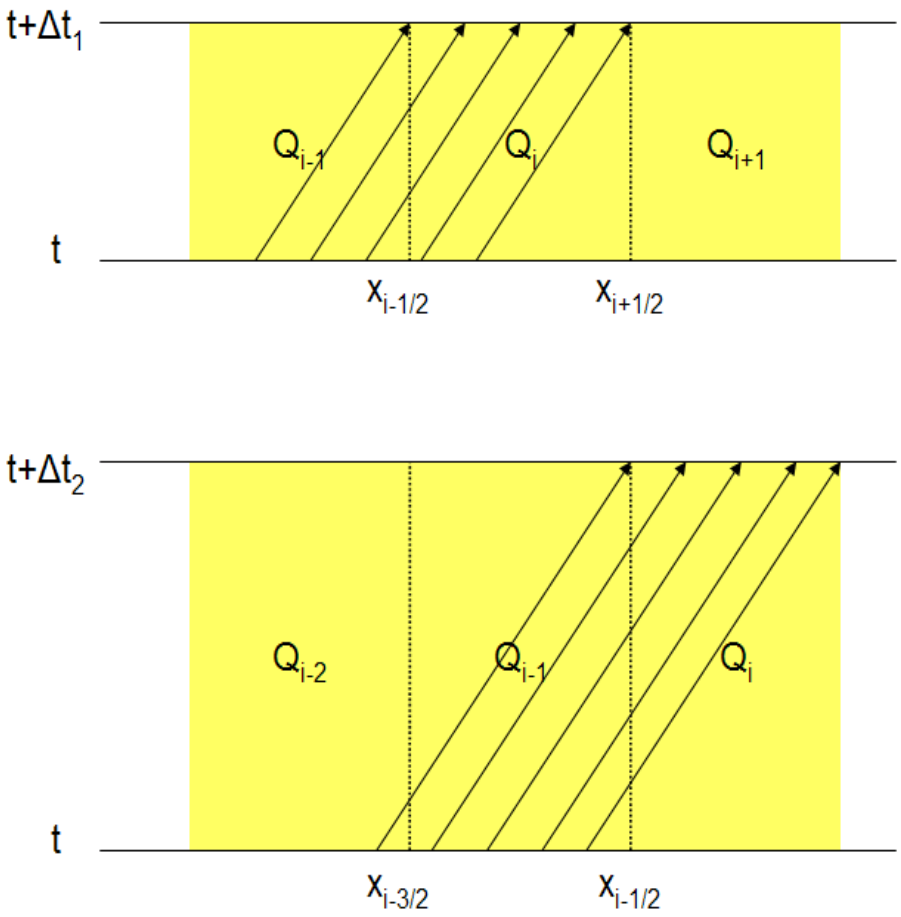

Figure 108: CFL: stable (above) and unstable (below) cases

see that the value Q^t_{i-2} comes into play. The flux at the interface $x_{i-1/2}$ depends now on the value Q^t_{i-2}.
The finite volume method would certainly be unstable when applied with such a large time step, no matter how the flux was specified, if this numerical flux depended only on Q^t_{i-1} and Q^t_i. This is a consequence of the CFL condition.

CFL Condition: *A numerical method can be convergent only if its numerical domain of dependence contains the true domain of dependence of the PDE, at least in the limit as Δt and Δx go to zero.*

Note: The CFL condition is only a necessary condition for stability. It is not always sufficient to guarantee stability [2].
When using DASSL, CFL condition cannot be always satisfied because adaptive time step procedure is used. We can use instead methods that use a fixed time step proce-

dure, like $Runge-Kutta$ methods of different orders in Dymola. This way we can adjust width grid used in our implementation by looking, after the simulation, the maximum time step used by Runge-Kutta.

7 Conclusion

In general Method of Lines gives better results than Finite Volume Methods and it is simple to use. First-order Finite Volume Methods methods are not as accurate as first-order Method of Lines methods. Only conservation laws, like the Euler system, are not simulated well enough. For these methods, Method of Lines introduces too much spurious oscillations. It is however possible that this is due to the particular implementation done in the MOL Package.
Finite Volume Methods is very complicated to understand and to implement. In order to use it, you have to know something about this method. For example, in the case of *FluxLimiter* Library, the Riemann theory must be understood in order to use correctly the blocks. Moreover, equations must be rewritten in conservation form. Finite Volume Methods is mainly used for conservation laws and does not introduce much oscillations. Euler system of equations for example contains three conservation laws: conservation of mass, conservation of momentum and conservation of energy. This system of equations is implemented in three different ways. One implementation uses the Lax-Friedrichs flux and works quite well. Another implementation uses Roe´s flux and is instable. The third implementation modified the *FluxLimiter* Library to solve the Euler equations. Because of lack of time it was not possible to test and debug the last two implementations, that are for now instable. One possible cause might be the violation of the CFL condition, that cannot be assured at all time steps, when using methods such as DASSL. Another disadvantage of Finite Volume Methods are the fluxes. In the case of Lax-Friedrichs flux an optimal parameter, α, must be chosen. This is very cumbersome, especially for time expensive simulations.
A third method, Adaptive Method of Lines, was originally planed, but dropped because of the problem in Dymola of creating/destroying instances at run time that is required in adaptive methods.
General block combines both methods, Method of Lines and Finite Volume Methods, into a single method. Only one equation, the advection equation, is implemented by using this block and works well.
Although originally only time dependent PDEs were considered, an example of time independent PDE, Poisson equation, is also implemented and works well. Consequently PDE Package can be theoretically used to solve time dependent and time independent PDEs.
In the present thesis all examples were implemented in one dimension. It is however possible to extend the package to more dimensions. Maybe future works could achieve this task.

8 Bibliography

References

[1] Francois E. Cellier, Ernesto Kofman: Continuous System Simulation, Springer

[2] Randall J. Leveque: Finite Volume Methods for Hyperbolic Problems, Cambridge Texts in Applied Mathematics

[3] Bruce R. Munson, Donald F. Young, Theodore H. Okiishi: Fundamentals of Fluid Mechanics, Wiley International edition

[4] Michael Heath: Scientific Computing, McGraw-Hill

[5] Stanley J. Farlow: Partial Differential Equations for Scientists and Engineers

[6] Jose Diaz Lopez: Shock Wave Modeling for Modelica.Fluid library using Oscillation-free Logarithmic Reconstruction, The Modelica Association, 2006, September

[7] Robert Artebrant, H. Joachim Schroll: Limiter-Free Third Order Logarithmic Reconstruction, SIAM, Vol.28, No. 1, pp. 359-381

[8] Dmitri Kuzmin: Introduction to Computational Fluid Dynamics Lecture, www.mathematik.uni-dortmund.de/ kuzmin/cfdintro/cfd.html

[9] John D. Anderson: Computational Fluid Dynamics, McGraw-Hill

[10] Mack J. Hyman: Moving Mesh Methods for Partial Differential Equations, Mathematics Applied to Science: In Memoriam Edward D. Conway, pages 129-153. Academic Press, Boston, Mass., 1988