Shock Wave Modeling for Modelica.Fluid library using Oscillation-free Logarithmic Reconstruction

José Díaz López
email: jose.diaz@dynasim.se
Dynasim AB
Research Park Ideon, SE-223 70 Lund, Sweden

Abstract

An oscillation-free discretisation of conservation laws has been implemented using Modelica.Fluid library [1] with a novel logarithmic reconstruction technique. Roe's approximation flux is used in combination with it. The results show that this logarithmic reconstruction has many advantages: limiter-free, absence of spurious oscillations near shock waves as well as third order of approximation to the solution.

Keywords: Finite Volume Methods, Shock waves, Roe's Flux approximation, Logarithmic reconstruction

1 Introduction

This paper presents an implementation of the logarithmic reconstruction method for hyperbolic partial differential equations in conservation law form, as presented in [7], using Modelica.Fluid [1].

The main objective is to resolve numerically shock waves avoiding spurious oscillations, limiters, artificial viscosity or wave velocity estimators.

2 Problem Formulation

Scalar hyperbolic partial differential equations (HPDE's) are usually formulated in the following compact way

$$u_t + f(u)_x = 0 \qquad (1)$$

where $u : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^m$ is the solution and $f : \mathbb{R}^m \to \mathbb{R}^m$ is the flux. The subindices represent partial derivation respect to time (t) and space (x) variables. Some classical case studies in HPDE's are

- **Advection Equation** where $x \in [a,b]$, $u : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$, $f \equiv \mathrm{I}$ with boundary condition $u(t,a) = g(t)$ and initial value $u(0,x) = h(x)$.

- **Burgers' Equation** where $x \in [a,b]$, $u : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$, $f(u) = \frac{1}{2}u^2$, boundary condition $u(t,a) = g(t)$ (or periodic) and initial value $u(0,x) = h(x)$.

- **Euler System**. If the section A of the pipe is constant we have that $x \in [a,b]$,

$$u = \begin{pmatrix} \rho A \\ \rho v A \\ \rho e_0 A \end{pmatrix},$$

$$f(u) = \begin{pmatrix} \rho v A \\ (\rho v^2 + p)A \\ \rho v h_0 A \end{pmatrix}$$

boundary conditions and initial values for ρ, v and p. The variables denote physical quantities as follows

ρ	mass dentity
v	fluid speed
e_0	stagnation internal energy
h_0	stagnation entalphy
p	pressure

The stagnation internal energy is related to the specific internal energy as

$$e_0 = e + \frac{1}{2}v^2,$$

and h_0 satisfies

$$h_0 = e_0 + \frac{p}{\rho}.$$

A source term $S(u)$ is included if the section A varies gradually. In this case, the flow is called *quasi-one-dimensional*. The conservation law takes the form

$$u_t + f(u)_x = S(u). \qquad (2)$$

This source term is defined as

$$S(u) = \begin{pmatrix} 0 \\ -p\frac{\mathrm{d}A}{\mathrm{d}x} + \rho G A \\ -\rho q A \end{pmatrix}$$

where q is the thermal conductivity constant and $G = \frac{1}{2}\rho v|v|k\frac{4}{D}$, k the wall friction coefficient and D the equivalent diameter of the duct. The source term can also include effects related to temperature

$$S(u) = \begin{pmatrix} 0 \\ -p\frac{\mathrm{d}A}{\mathrm{d}x} + \rho G A \\ -\rho q A + kA\frac{\partial^2 T}{\partial x^2} + \dot{Q} \end{pmatrix}$$

where T is temperature, k is the thermal conductivity (considered constant along the pipe) and $\dot{Q}$ is the heat flow. The system is completed by equations that are characteristic to the medium, that is

$$p = p(\rho, T),\ e = e(\rho, T).$$

The three equations of this system have physically relevant names: *mass*, *momentum* and *energy* balance equations. We will refer to them later on with those names.

3 Numerical flux and reconstruction

3.1 Finite volume methods for HPDE's

Consider now the semidiscretisation of (1) by means of finite volumes (FV). This method yields a system of ODE's that are numerically integrated in time. For a survey in FV-standard notation, see also [2].

The method can be briefly summarised as follows. Consider the domain $\Omega = [a,b]$, divided into n segments (x_i, x_{i+1}) with $a = x_0 \leq x_1 < ... < x_i < x_{i+1} < ... < x_{n+1} = b$. We use also as notation $\Delta x_i = x_{i+1} - x_i$ and

$$x_{i+\lambda} = \lambda x_i + (1-\lambda)x_{i+1} \qquad \lambda \in (0,1).$$

The so-called computing cells are therefore defined as

$$C_i = [x_{i-1/2}, x_{i+1/2}], \quad i = 1, ..., n.$$

For the numerical integration in time, consider as state variables the averages

$$\bar{u}_i(t) = \frac{1}{\Delta x_i} \int_{C_i} u(x,t)\mathrm{d}x, \tag{3}$$

where $\Delta x_i = x_{i+1/2} - x_{i-1/2}$, the length of C_i. The main objective now is to state the governing equations for this averages. This is the process of *semidiscretisation* of the HPDE. This method is inspired by the integral from of (1).

The *semidiscretised* ODE system is based on the construction of a flux approximation from the left and right states at $x_{i+1/2}$

$$f(u(x_{i+1/2}, t)) \approx \hat{f}_{i+1/2}(u^-(x_{i+1/2}, t), u^+(x_{i+1/2}, t)).$$

The function $\hat{f}$ is the so-called *numerical flux approximation*, and governs the evolution of $\bar{u}_i$ as follows

$$\frac{\mathrm{d}}{\mathrm{d}t}\bar{u}_i = \frac{1}{h}\left(\hat{f}_{i+1/2} - \hat{f}_{i-1/2}\right), \quad i = 1, ..., n \tag{4}$$

where the sign $\pm$ indicates the side of x_i taken from the reconstruction of u.

The focus now on lies exclusively on $\bar{u}_i$ and the numerical flux approximation $\hat{f}_{i+1/2}$. The reason is that different properties of the conservation law are inherited by the numerical simulation, depending on the reconstruction method for $u^{\pm}$ and $\hat{f}_{i+1/2}$.

To show the reasons for requiring limiter-free, non-oscillatory reconstruction and viscous-free flux approximation, two simpler approaches are presented before. One of them is already implemented in Modelica.Fluid.

3.2 A primer approach

In [1] we find an approach to discretising the pipe example presented before using Modelica. The discretisation uses the following approximations

$$\bar{u}_i = u(x_i, t)\Delta x_i \tag{5a}$$

$$\hat{f}_{i+1/2} = f(u^-(x_{i+1/2}, t)) \tag{5b}$$

and piecewise constant reconstruction of u in C_i for the mass balance and energy balance equations. The same setup is used for the moment balance equation, except that the grid used here is then staggered, see [6]. The argumentation presented in [6] is that staggering results in a velocity and pressure fields that are physically meaningful (pp. 120). Spurious oscillations are not allowed as solutions in the velocity field. It is also stated that the implementation is cumbersome. This staggered-grid formulation is even more troublesome for object-oriented modeling.

The example model of a distributed pipe found in Modelica.Fluid.Components.Pipes, called `DistributedPipeFV`, implements a non-staggered grid, with the setup described above. The discretisation is found in Modelica.Fluid.BaseClasses.Pipes.Flow1d_FV.

3.3 Lax-Friedrichs numerical flux

The Lax-Friedrichs numerical flux is classically the easiest possibility of a numerical flux. This numerical flux is unstable and needs a damping factor α to avoid spurious oscillations. The numerical flux is defined as

$$\begin{aligned}\hat{f}^{LF}_{i+1/2} = &\frac{1}{2}(f(u^-(x_{i+1/2}, t)) + f(u^+(x_{i+1/2}, t))) \\ &- \frac{1}{2}\alpha(u^+(x_{i+1/2}, t) - u^-(x_{i+1/2}, t))\end{aligned} \tag{6}$$

The principal problem with this approach is that the parameter α has to be tuned to avoid unnecessary damping of shock waves and corners. This parameter is related to the traveling wave velocities and is in general difficult to estimate them *a priori*. Furthermore, there is no easy way to treat waves traveling in both left and right directions with this scheme.

We want to avoid those artificial factors as much as possible. The cause of such spurious oscillations is a too coarse estimations of u^+ and u^-.

3.4 Reconstruction of u

Another classical approach is to consider linear reconstruction of u in C_i, see [4, 12]. In this case,

$$u^+ = \bar{u} + \frac{\Delta x_i}{2} m^+$$

and

$$u^- = \bar{u} - \frac{\Delta x_i}{2} m^-$$

, where the slopes m^+ and m^- are estimated from the averages $\bar{u}_{i-1}, \bar{u}_i$ and $\bar{u}_{i+1}$. This approach needs the so called *limiter*, simply a bounded function $L(m)$. The reconstruction takes the form

$$u^+ = \bar{u} + \frac{\Delta x_i}{2} L(m^+)$$

and

$$u^- = \bar{u} - \frac{\Delta x_i}{2} L(m^-)$$

instead. The main advantage is that limiters avoid spurious oscillations, but the maximum approximation order of reconstruction is two, if the limiter is constructed in an appropiate way, see [4].
Higher order of approximation requires other techniques.

3.5 Oscillations in DistributedPipeFV

Consider the Sod problem, described in [12]. The Sod problem consists of a domain with a perfectly isolating membrane in the middle that separates a zone with high pressure and temperature of a zone with low pressure and temperature. The dynamics of the fluid in the whole domain are governed by Euler equations. At time $t = 0$, the membrane is removed instantaneously. Further details about the configuration are given later in Sec. 4.
The corresponding Modelica model is implemented with two DistributedPipeFV models connected with adequate initial conditions, depicted in Fig. 1. The reason for having two pipes is that the Sod problem has initially waves traveling in both left and right from the membrane. Since we want to experiment with Lax-Friedrichs numerical flux also, we need to specify two different α coefficients in two different domains.

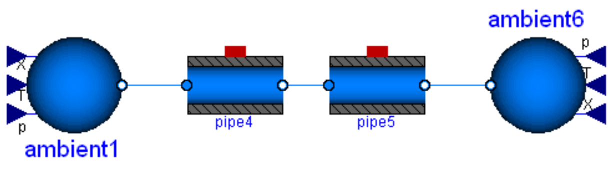

Figure 1: Sod problem model using two connected pipes

3.5.1 Original approach

The implementation of the original approach is in Modelica.Fluid.BaseClasses.Pipes.Flow1D_FV. We present only the dynamical equations of the model since this is the focus of this paper.
The implementation of equations (5) can be observed in the listings 1 (mass and energy balance) and 2 (momentum balance).

Listing 1: Mass and Energy Balance

```
//Mass and energy balance
for i in 1:n loop
  if static then
  ...
  else
    der(m[i]) = m_flow[i] - m_flow[i + 1] + ms_flow[i];
    der(mXi[i, :]) = mXi_flow[i, :] - mXi_flow[i + 1, :] + msXi_flow[i, :];
    der(U[i]) = H_flow[i] - H_flow[i + 1] + Qs_flow[i];
  end if;
end for;
```

Listing 2: Momentum Balance

```
// Momentum Balance
 for i in 2:n loop
   F_p[i] = (medium[i-1].p-medium[i].p)*A_inner;
   F_f[i] = -dp[i]*A_inner;
   (if dynamicTerm then der(m_flow[i])*length/n else 0) =
                         F_p[i] + F_f[i] + (I_flow[i-1]-I_flow[i+1])/2;
 end for;
```

In this approach, the boundary conditions are easily included: the ports `port_a` and `port_b` are considered the nodes 0 and $n+1$ of the discretisation, and included when the array index of the component `medium` is less than 1 or larger than n.
The result is depicted in Fig. 2, and the spurious oscillations appear clearly in the two shock waves.

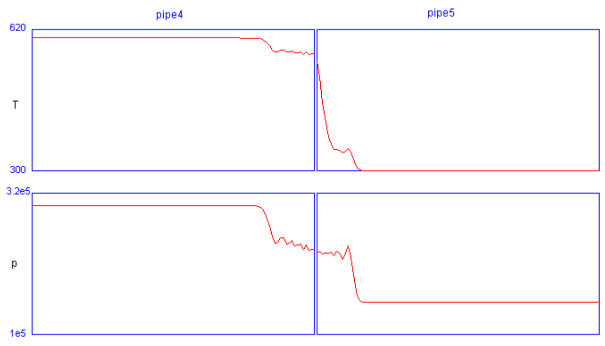

Figure 2: Spurious oscillations on top the shock waves in the Sod problem

3.5.2 Lax-Friedrichs Flux and upwinding

Lax-Friedrichs flux avoids oscillations and is easy to implement. With some tuning of the α factor, this flux can yield better results. The *sine qua non* condition of LF-flux is that the shock waves are traveling in one direction.
The implementation is presented in the listings 3 and 4.

Listing 3: Mass and Energy Balance (LF)

```
//Mass and energy balance

for i in 2:n-1 loop
    if static then
    ...
    else
      der(m[i]) = m_flow[i] - m_flow[i + 1] + ms_flow[i]
                + alpha*(-m_flow[i-1]+2*m_flow[i]-m_flow[i+1]);

      der(mXi[i, :]) = mXi_flow[i, :] - mXi_flow[i + 1, :] + msXi_flow[i, :]
                + alpha*(-msXi_flow[i-1, :]+2*msXi_flow[i, :]-msXi_flow[i+1, :]);

      der(U[i]) = H_flow[i] - H_flow[i + 1] + Qs_flow[i]
                + alpha*(-H_flow[i-1]+2*H_flow[i]-H_flow[i+1]);
    end if;
  end for;
```

Listing 4: Momentum Balance (LF)

```
for i in 2:n-1 loop
  F_p[i] = (medium[i-1].p-medium[i].p)*A_inner
         + alpha*(-medium[i-1].p+2*medium[i].p-medium[i+1].p)*A_inner;

  F_f[i] = -dp[i]*A_inner;
  (if dynamicTerm then
          der(m_flow[i])*length/n else 0) = F_p[i] + F_f[i]
                                          + (I_flow[i-1]-I_flow[i+1])/2;

end for;
```

Notice that the main difference is the added terms multiplied by α. The results are depicted in Fig. 3, before the left shock wave bounces agains `ambient1`, and in Fig.4 after the wave bouncing. As we already mentioned, the spurious oscillations appear since α is negative in pipe4.

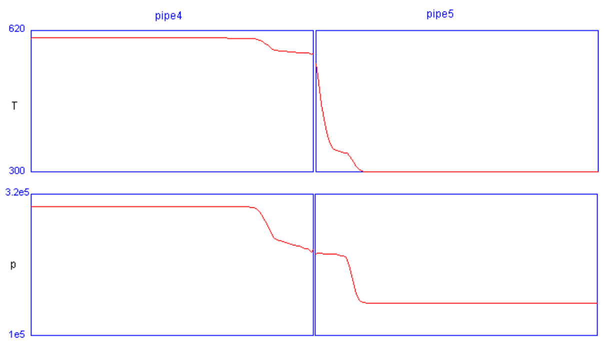

Figure 3: Lax-Friedrichs Flux with $\alpha = -0.1$ in pipe4 and $\alpha = 0.1$ in pipe5. (Before first bouncing.)

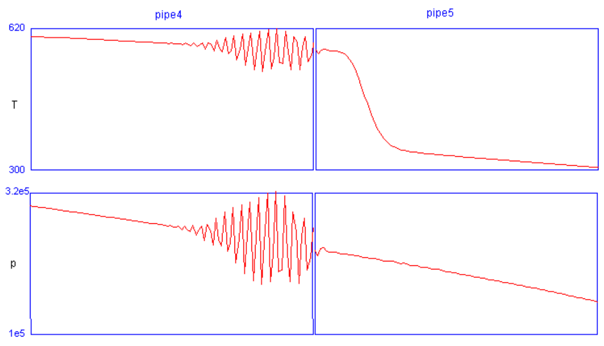

Figure 4: Lax-Friedrichs Flux with $\alpha = -0.1$ in pipe4 and $\alpha = 0.1$ in pipe5. Spurious oscillations appear in pipe4 after the reflexion of a small shock wave at the left boundary.

3.6 Logarithmic reconstruction

Logarithmic reconstruction is a technique recently developed mainly by A. Schroll in the papers [8, 9, 7] with examples in [10]. Schroll's scheme is called **LDLR**, standing for *Local Double Logarithmic Reconstruction*. LDLR is as a natural generalisation of A. Marquina's hyperbolic reconstruction presented in [5]. LDLR has also the advantage over other approximations of higher order (as those studied by S. Serna and A. Marquina in [11], based on their own variant of the weighted ENO schemes) that shock and rarefaction waves are resolved correctly without any estimates, thresholds or heuristics.
When using LDLR for reconstruction, u is represented as a piecewise logarithmic function. In particular, the left state u^+ and the right state u^- at $x_{i+1/2}$ are estimated as

$$u^{\pm} = u_i + c_3 \Delta x_i \eta^{\pm}(c_1) + c_4 \Delta x_i \eta^{\pm}(c_2) \tag{7}$$

where

$$\eta^+(t) = -\frac{\log(1-t)+t}{t^2}$$

and

$$\eta^-(t) = -\frac{(t-1)\log(1-t)-t}{t^2}$$

with appropriate constants c_1, c_2, c_3 and c_4 for third order convergence.
The computation of the constants is relatively simple. We describe the computation in the following algorithm

1. Calculate
$$d_1 = \frac{\bar{u}_i - \bar{u}_{i-1}}{\Delta x_{i-1}}, d_2 = \frac{\bar{u}_{i+1} - \bar{u}_i}{\Delta x_i}. \tag{8}$$

2. The coefficient c_1 is obtained from
$$c_1(d_1, d_2) = (1-\mathrm{tol})\left(1 + \mathrm{tol} - \frac{2|d_1|^q |d_2|^q + \mathrm{tol}}{|d_1|^{2q} + |d_2|^{2q} + \mathrm{tol}}\right)$$
where $\mathrm{tol} = 0.1\Delta x_i^q$ and typically $q = 1.4$. This factor q controls the local variation, and the larger q the smaller the local variation.

3. The constants c_2, c_3 and c_4 are calculated from
$$\begin{aligned} c_2 &= \frac{c_1}{c_1 - 1} \\ c_3 &= \frac{(c_1-1)(d_2(1-c_2)-d_1)}{c_2 - c_1} \\ c_4 &= d_1 - c_3. \end{aligned}$$

4. Using (7) we can now calculate u^-.

The procedure for u^+ is very similar. The only difference is that d_1 and d_2 are defined instead as following

$$d_1 = \frac{\bar{u}_{i+1} - \bar{u}_i}{\Delta x_i}, \ d_2 = \frac{\bar{u}_{i+2} - \bar{u}_{i+1}}{\Delta x_{i+1}}.$$

3.7 Roe's numerical flux and upwind schemes

Upstream and downstream propagation is used in upwind schemes. This is intrinsic and physically meaningful information contained in the HPDE. The cornerstone is to decompose the flux difference at $x_{i+1/2}$ as $\hat{f}^+ - \hat{f}^- = W^+ + W^-$, according to the physics of the Euler equations.
Consider the left and right states of u (as defined for **Euler system**) at $x_{i+1/2}$. Define Roe's averages as (we drop the subindex for simplicity in this case)

$$\tilde{\rho}^2 = \rho^+ \rho^- \tag{9a}$$

$$\tilde{v} = \frac{\sqrt{\rho^+}v^+ + \sqrt{\rho^-}v^-}{\sqrt{\rho^+} + \sqrt{\rho^-}} \tag{9b}$$

$$\tilde{h} = \frac{\sqrt{\rho^+}h_0^+ + \sqrt{\rho^-}h_0^-}{\sqrt{\rho^+} + \sqrt{\rho^-}} \tag{9c}$$

$$\tilde{a}^2 = (\gamma - 1)(\tilde{h} - 1/2\tilde{v}^2). \tag{9d}$$

Using Roe's averages and the differences

$$\Delta p = p^+ - p^-,\ \Delta\rho = \rho^+ - \rho^-,\ \Delta v = v^+ - v^-,$$

we can estimate the traveling wave velocities as follows

$$\begin{aligned} \lambda_1 &= \tilde{u} - \tilde{a} && (10a)\\ \lambda_2 &= \tilde{u} && (10b)\\ \lambda_3 &= \tilde{u} + \tilde{a}. && (10c) \end{aligned}$$

These travel velocities have associated wave forms, estimated as

$$W_1 = \begin{pmatrix} 1 \\ \tilde{u} - \tilde{a} \\ \tilde{h} - \tilde{u}\tilde{a} \end{pmatrix} \qquad (11a)$$

$$W_2 = \begin{pmatrix} 1 \\ \tilde{u} \\ \frac{1}{2}\tilde{h} \end{pmatrix} \qquad (11b)$$

$$W_3 = \begin{pmatrix} 1 \\ \tilde{u} + \tilde{a} \\ \tilde{h} + \tilde{u}\tilde{a} \end{pmatrix} \qquad (11c)$$

and associated strengths, estimated as

$$\begin{aligned} a_1 &= \frac{1}{2\tilde{a}^2}(\Delta p - \tilde{\rho}\tilde{a}\Delta u) && (12a)\\ a_2 &= \frac{1}{\tilde{a}^2}(\tilde{a}^2\Delta\rho - \Delta p) && (12b)\\ a_3 &= \frac{1}{2\tilde{a}^2}(\Delta p + \tilde{\rho}\tilde{a}\Delta u). && (12c)\\ & && (12d) \end{aligned}$$

With all this information, we can decompose the traveling waves, according to velocities λ_i. In a compact notation we can write

$$\begin{aligned} W^+ &= \sum_{\lambda_i > 0} a_i W_i && (13a)\\ W^- &= \sum_{\lambda_i \leq 0} a_i W_i && (13b) \end{aligned}$$

with $i = 1,2,3$. For more complex schemes considering contact traveling waves, see [4, 12].

3.8 LDLR Stencil and Roe's Flux

The LDLR is symmetric as long as the numerical flux keeps symmetry. The following dependency graph shows the dependency of $\frac{d\bar{u}_i}{dx}$ on $\bar{u}_i$, $i = 1,2,...,n$.

$$\begin{tikzcd}
 & & & \frac{d\bar{u}_i}{dx} \arrow[dll] \arrow[drr] & & & \\
 & \hat{f}_{i+1/2} \arrow[dl] \arrow[dr] & & & & \hat{f}_{i-1/2} \arrow[dl] \arrow[dr] & \\
u^+_{i+1/2} \arrow[d] & & u^-_{i+1/2} \arrow[d] & & u^+_{i-1/2} \arrow[d] & & u^-_{i-1/2} \arrow[d] \\
\bar{u}_{\{i+2,i+1,i\}} & & \bar{u}_{\{i+1,i,i-1\}} & & \bar{u}_{\{i+1,i,i-1\}} & & \bar{u}_{\{i,i-1,i-2\}}
\end{tikzcd}$$

We observe that $\frac{d\bar{u}_i}{dx}$ depends on $\bar{u}_{i+2}, \bar{u}_{i+1}, \bar{u}_i, \bar{u}_{i-1}$ and $\bar{u}_{i-2}$. Thus, treatment of the boundary conditions is needed for $\frac{d\bar{u}_1}{dx}$ and $\frac{d\bar{u}_n}{dx}$

Boundary Conditions Consider $i = 1$ in Eq. (4). To compute $\frac{d\bar{u}_1}{dx}$ we need the values of $\hat{f}_{1+1/2}$ and $\hat{f}_{1-1/2}$, constructed from the available data and the boundary conditions. We use the *ghost cell* approach to solve this problem. The ghost cell technique is applied by extending the domain beyond the boundary with new cells. The amount of those new cells is as many as needed to use the same central numerical approximation.
In our case, to compute $\frac{d\bar{u}_1}{dx}$ we would need u_3, u_2, u_1, u_0 and u_{-1} because of the size of the stencil. We have added two ghost cells u_0 and u_{-1}. Their value has to be design to meet the boundary conditions. For simplicity, we just consider $u_0 = u_{-1} = u(a,t)$ for this study.
The same reasoning is also valid for $i = n$, where two ghost cells u_{n+1} and u_{n+2} are to be introduced. In the same fashion, we consider $u(b,t) = u_{n+1} = u_{n+2}$.
Clearly, this approximation means a loss of accuracy at the boundaries, where the order of approximation drops. Of course, more sophisticated numerical schemes may be derived to meet higher approximation orders, using the two new degrees of freedom introduced.

3.9 Source term discretisation

Even though we don't explore the source term in this paper, we provide a way of handling the source term for completeness.
The semidiscretised system considering source term S takes the form

$$\frac{d}{dt}\bar{u}_i = \frac{1}{h}\left(\hat{f}_{i+1/2} - \hat{f}_{i-1/2}\right) + S_i, \quad i = 1,...,n. \qquad (14)$$

To model the term S_i, we use the speeds obtained by Roe's numerical flux. The decision is then taken using λ_1 and λ_3 from Eq. (10)

$$S_i = \begin{cases} S(u^+_{i-1/2}), & \lambda_1 > 0 \\ \dfrac{\lambda_1 S(u^-_{i-1/2}) - \lambda_3 S(u^+_{i+1/2})}{\lambda_1 - \lambda_3}, & \lambda_1 \leq 0 \leq \lambda_3 \\ S(u^-_{i+1/2}), & \lambda_3 < 0 \end{cases} . \qquad (15)$$

In case we include thermodynamical effects in S_i, we have to estimate the temperature gradient. It is important then to use a scheme with at least *third order*. This is important to keep the convergence order, since LDLR approximation is of third order, [7].

4 Applications

We will explore now the results of the LDLR implementation with Roe's flux using Modelica.Fluid. For details of the implementation, see section 5. We come back first to the

two pipe problem presented in Fig. 1, and then some other examples showing different aspects of the approach.
In all experiments, the chosen medium was `IdealGases.SingleGases.CH3COOH` from `Modelica.Media`. The length of the pipes is 10 cm and 20 cm the larger ones. The discretisation used is 10 cells/cm, that is, 100 cells for the short pipes and 200 for the large pipes. As before, the high pressure and temperature zone has $p_h = 3 \cdot 10^5$ Pa and $T_h = 600$ K. The low pressure and temperature zone has $p_l = 1.5 \cdot 10^5$ Pa and $T_l = 300$ K.

4.1 Two pipe Sod problem

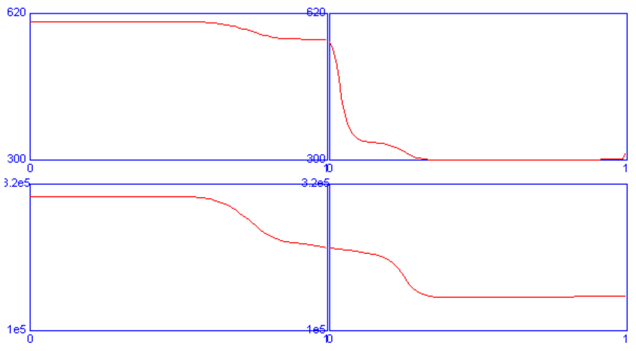

Figure 5: Oscillation-free LDLR solution and Roe's flux of the two pipe Sod problem

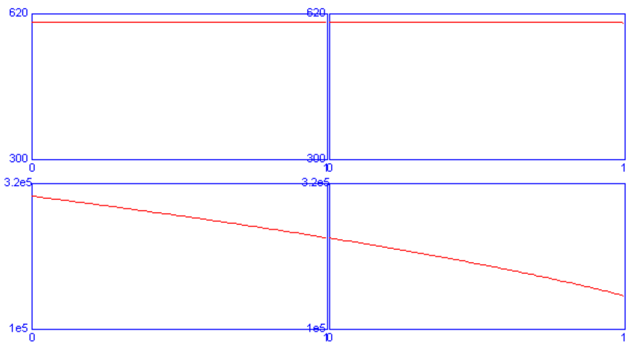

Figure 6: Steady State of the Two pipe Sod problem with LDLR and Roe's Flux.

With this scheme, the right upwinding and the logarithmic reconstruction makes the solution oscillation-free. We observe in particular that steady state is reached without any trouble, depicted in Fig. 6. This is not possible with the original approach or the Lax-Friedrichs flux in the general case. Fig. 5 depicts the solution given by the new scheme at the same time point depicted in Fig. 2 and Fig. 3.

4.2 One pipe Sod problem

We want to establish now how is the behavior of the ghost cell implementation of the boundary conditions and connectors. For that, we simulate the model depicted in Fig. 7.

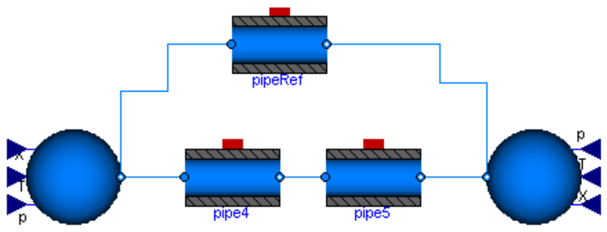

Figure 7: Model for boundary condition testing.

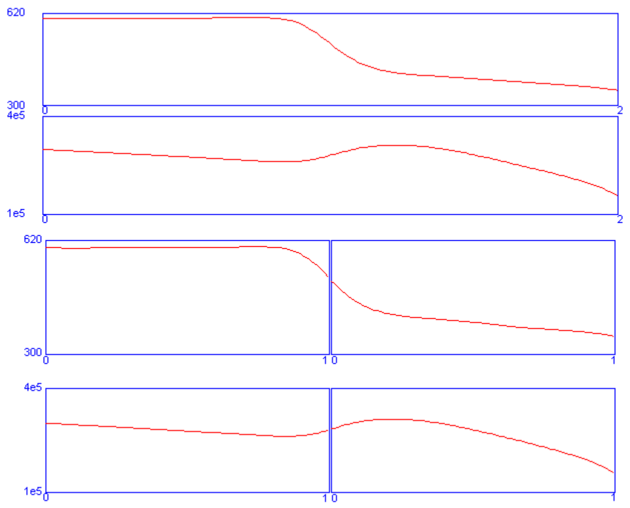

Figure 8: Simulation result of the comparison model

The model has a large pipe that undergoes the same conditions as two series connected pipes. The pressure and temperature profiles are then compared. Optically, the solution is not distinguishable, as depicted in Fig. 8.
When plotting the difference, depicted in Fig. 9, we observe the differences. In both temperature and pressure, the difference is around two orders of magnitude less than the variables. This difference causes traveling waves that does not influence very much the behavior of the system. Furthermore, this difference goes to zero as the number of discretisation intervals is increased.

4.3 Symmetry

For testing symmetry of the LDLR with Roe's approximation, we simulate the model depicted in Fig. 10. Simply two pipes in parallel and one of them with interchanged connections.
Fig. 11 shows the result of the simulation. Notice that the index of the left graph goes from 0 to 1 while the right graph goes from 1 to 0.

4.4 Three pipe problem

Three pipe problem is a very interesting application for boundary condition testing. The connection of three pipes at a point is challenging because of the mass balance in an infinitesimal volume at the connector.

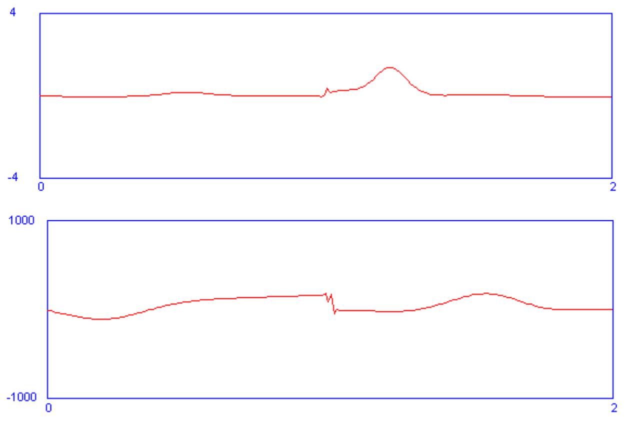

Figure 9: Difference between single pipe and two series pipe.

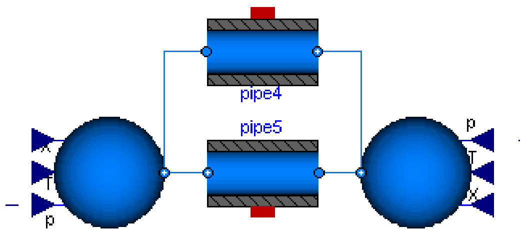

Figure 10: Model used for symmetry testing.

In this particular application, the wave interference can be viewed for inviscid or low viscosity fluids, if the length of the pipes are different. In fig. 12 is depicted the system modeled. Fig. 13 shows the pressure and temperature profiles when the traveling wave bounces in the short pipe. We observe in the larger pipe how the shock pressure wave is still traveling farther.
The profiles after the shock wave bounces at the end of the larger pipe is presented in Fig. 14.

5 Implementation details

5.1 Structure

The structure of the Modelica code follows the same dependency shown before. The parts of the code can be identified with the following diagram (from reconstruction to derivative)

$$\bar{u}_i \xrightarrow{\log} u^+_{i+1/2}, u^-_{i+1/2} \xrightarrow{W^+,W^-} \hat{f}_{i+1/2} \longrightarrow \frac{\mathrm{d}}{\mathrm{d}t}\bar{u}_i$$

The function `RoeWaves` computes the traveling waves W^+, W^- from the reconstruction scheme. The function `logarithmicminus` computes the approximation $u^-_{i+1/2}$ from the averages u_i and its companion `logarithmicplus` computes $u^+_{i+1/2}$ from the averages u_i, see listing 5.

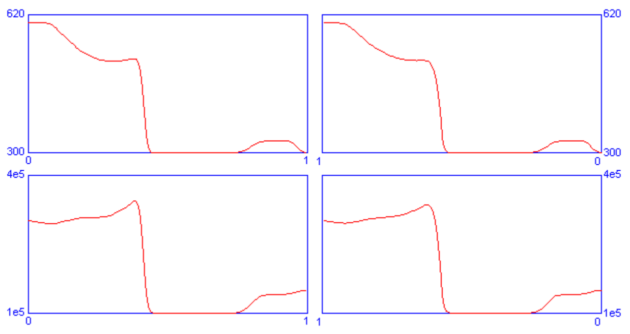

Figure 11: Solution of the symmetry test.

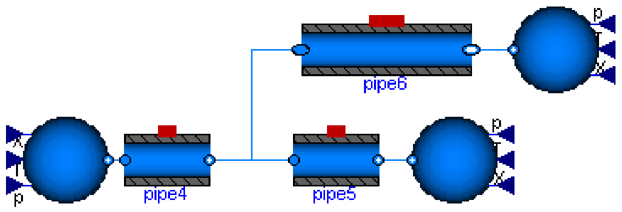

Figure 12: Three pipe model for testing of boundary conditions for several connections.

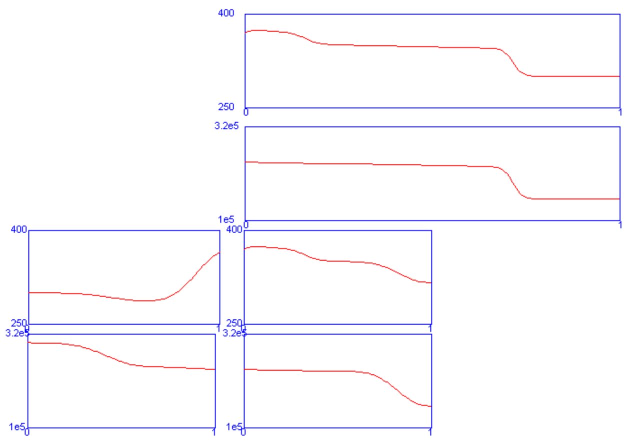

Figure 13: Solution after the traveling wave in the short pipe bounces.

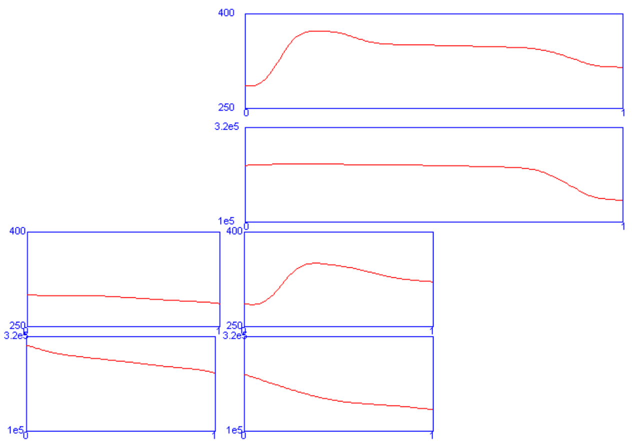

Figure 14: Solution after the traveling wave bounces at the large pipe.

At last, the approximations contained in the arrays `f1p,f2p,f3p` and `f1m,f2m,f3m`, that are equated to the derivatives $\frac{\mathrm{d}}{\mathrm{d}t}\bar{u}_i$, see listing 6 and 7

Listing 5: Roe's Flux with logarithmic reconstruction

```
(f1p[i],f2p[i],f3p[i],f1m[i],f2m[i],f3m[i]) =
  RoeWaves(
  {logarithmicminus(medium[i+1].d,
                    medium[i].d,
                    medium[i-1].d, length/n,q),
   logarithmicminus(medium[i+1].d*v[i+1],
                    medium[i].d*v[i],
                    medium[i-1].d*v[i-1],
                    length/n,q),
   logarithmicminus(medium[i+1].d*medium[i+1].u + medium[i+1].d*(v[i+1])^2/2,
                    medium[i].d*medium[i].u + medium[i].d*(v[i])^2/2,
                    medium[i-1].d*medium[i-1].u + medium[i-1].d*(v[i-1])^2/2,
                    length/n,q)},
  {logarithmicplus(medium[i+2].d,
                   medium[i+1].d,
                   medium[i].d,length/n,q),
   logarithmicplus(medium[i+2].d*v[i+2],
                   medium[i+1].d*v[i+1],
                   medium[i].d*v[i],
                   length/n,q),
   logarithmicplus(medium[i+2].d*medium[i+2].u + medium[i+2].d*(v[i+2])^2/2,
                   medium[i+1].d*medium[i+1].u + medium[i+1].d*(v[i+1])^2/2,
                   medium[i].d*medium[i].u + medium[i].d*(v[i])^2/2,
                   length/n,q)},
  1.4);
```

Listing 6: Mass Balance with Roe's Flux

```
der(m[i]) = - (f1p[i-1] + f1m[i]) *A_inner + ms_flow[i];
der(mXi[i, :]) = mXi_flow[i-1,  :] - mXi_flow[i, :] + msXi_flow[i, :];
der(U[i]) = - (f3p[i-1] + f3m[i]) *A_inner + Qs_flow[i];
```

Listing 7: Momentum Balance with Roe's Flux

```
(if dynamicTerm then
      der(m_flow[i])*length/n else 0) = -(f2m[i]+f2p[i-1])*A_inner
                                     + F_f[i] + (I_flow[i-1]-I_flow[i+1])/2;
```

5.2 Including connectors and boundary conditions

The ghost cells $u_{-1} = u_0$ are implemented in `medium_a`, at the right boundary. At the left boundary, the ghost cells $u_{n+1} = u_{n+2}$ are implemented in `medium_b`. Then, they are connected to the discretisation using the `semiLinear` operator and adequate relations for `m_flow` variables, see listing 8.

Listing 8: Ports and Ghost cells

```
//Port medium models for ghost cells
port_a.p = medium_a.p;
port_b.p = medium_b.p;
port_a.h = medium_a.h;
port_b.h = medium_b.h;
port_a.Xi = medium_a.Xi;
port_b.Xi = medium_b.Xi;

// Boundary conditions
port_a.H_flow = semiLinear(port_a.m_flow, port_a.h, medium[1].h);
port_b.H_flow = semiLinear(port_b.m_flow, port_b.h, medium[n].h);
port_a.mXi_flow = semiLinear(port_a.m_flow, port_a.Xi, medium[1].Xi);
port_b.mXi_flow = semiLinear(port_b.m_flow, port_b.Xi, medium[n].Xi);
port_a.m_flow = m_flow[1];
port_b.m_flow = -m_flow[n];
```

The traveling waves at $x = 1 - 1/2$ are called `f1p_a, f2p_a, f3p_a, f1m_a, f2m_a, f3m_a` and those at $x = n + 1/2$ are called `f1p_b, f2p_b, f3p_b, f1m_b, f2m_b, f3m_b`. Their implementation of their computation is in listing 5. Their incorporation into the mass, momentum and energy balance is in listing 10.

Listing 9: Connectors in Roe approximation

```
 ...
// port_a
    (f1p_a,f2p_a,f3p_a,f1m_a,f2m_a,f3m_a) =
      RoeWaves(
     {logarithmicminus(medium[1].d,
                       medium_a.d,
                       medium_a.d,
                       length/n,q),
      logarithmicminus(medium[1].d*v[1],
                       medium_a.d*v[1],
                       medium_a.d*v[1],
                       length/n,q),
      logarithmicminus(medium[1].d*medium[1].u + medium[1].d*(v[1])^2/2,
                       medium_a.d*medium_a.u + medium_a.d*(v[1])^2/2,
                       medium_a.d*medium_a.u + medium_a.d*(v[1])^2/2,
                       length/n,q)},
     {logarithmicplus(medium[2].d,
                      medium[1].d,
                      medium_a.d,length/n,q),
      logarithmicplus(medium[2].d*v[2],
                      medium[1].d*v[1],
                      medium_a.d*v[1]  ,
                      length/n,q),
      logarithmicplus(medium[2].d*medium[2].u + medium[2].d*(v[2])^2/2,
                      medium[1].d*medium[1].u + medium[1].d*(v[1])^2/2,
                      medium_a.d*medium_a.u + medium_a.d*(v[1])^2/2,
                      length/n,q)},
     1.4);
 ...

// port_b
    (f1p_b,f2p_b,f3p_b,f1m_b,f2m_b,f3m_b) =
     RoeWaves(
     {logarithmicminus(medium_b.d,
                       medium[n].d,
                       medium[n-1].d,
                       length/n,q),
      logarithmicminus(medium_b.d*v[n],
                       medium[n].d*v[n],
                       medium[n-1].d*v[n-1],
                       length/n,q),
      logarithmicminus(medium_b.d*medium_b.u + medium_b.d*(v[n])^2/2,
                       medium[n].d*medium[n].u + medium[n].d*(v[n])^2/2,
                       medium[n-1].d*medium[n-1].u + medium[n-1].d*(v[n-1])^2/2,
                       length/n,q)},
     {logarithmicplus(medium_b.d,
                      medium_b.d,
                      medium[n].d,
                      length/n,q),
      logarithmicplus(medium_b.d*v[n],
                      medium_b.d*v[n],
                      medium[n].d*v[n],
                      length/n,q),
      logarithmicplus(medium_b.d*medium_b.u + medium_b.d*(v[n])^2/2,
                      medium_b.d*medium_b.u + medium_b.d*(v[n])^2/2,
                      medium[n].d*medium[n].u + medium[n].d*(v[n])^2/2,
                      length/n,q)},
     1.4);

 ...
```

Listing 10: Boundary Conditions

```
// Mass and Energy Balance
...
// side a
    if static then
    ...
    else
      der(m[1]) = - (f1p_a+f1m[1]) *A_inner + ms_flow[1];
      der(mXi[1, :]) = mXi_flow[1, :] - mXi_flow[2, :] + msXi_flow[1, :];
      der(U[1]) = - (f3p_a+f3m[1]) *A_inner + Qs_flow[1];
    end if;
    ...
//side b
     if static then
     ...
     else
       der(m[n]) = - (f1p[n-1] + f1m_b)* A_inner + ms_flow[n];
       der(mXi[n, :]) = mXi_flow[n, :] - mXi_flow[n + 1, :] + msXi_flow[n, :];
       der(U[n]) = - (f3p[n-1] + f3m_b) *A_inner + Qs_flow[n];
     end if;
     ...
//Momentum Balance
if lumped_dp then
   ...
   else
    ...
    (if dynamicTerm then der(m_flow[1])*length/n/2 else 0) =
                 -(f2m[1]+f2p_a)*A_inner + F_f[1] + (I_flow[1]-I_flow[2])/2;
    ...
    (if dynamicTerm then der(m_flow[n])*length/n else 0) =
         (f2m_b+f2p[n-1])*A_inner + F_f[n] + (I_flow[n-1]-I_flow[n+1])/2;
    ...
  end if;
```

Finally, the numerical scheme is incorporated to `DistributedPipeFV` by extending from `Flow1D_FV_Log` instead of `Flow1D_FV`.

Listing 11: LDLR in DistributedPipeFV

```
model DistributedPipeFV "Distributed_pipe_model_with_optional_wall"

extends Flow1D_FV_Log(
  Qs_flow=heat.Q_flow,
  ms_flow=zeros(n),
  msXi_flow=zeros(n, Medium.nXi));
...

end DistributedPipeFV;
```

6 Conclusions and Future Work

We have shown how the logarithmic reconstruction and appropriate upwinding captures shock waves without spurious oscillations. The numerical performance of these FV schemes is not more complicated than the originals used in Modelica.Fluid library, in terms of implementation and are more reliable. The CPU time observed was similar.
Some further investigations have to be done for time integration. The control of the Courant-Friedrichs-Lax number is critical for an even better capture of shock waves. This CFL number has to be controlled using special numerical integration schemes. Special Runge-Kutta methods have been developed and investigated, with successful results, in [3].

References

[1] Elmqvist, H.,Tummescheit, H. and Otter,M. *Object-Oriented Modeling of Thermo-Fluid Systems* Proceedings of Modelica'2003 conference. Linköping, Sweden. pp.269

[2] Gallouët, T. and Hérard, J.-M. and Seguin, N. *Some recent finite volume schemes to compute Euler equations using real gas EOS* Internat. J. Numer. Methods Fluids, 2002, pp. 1073–1138.

[3] Gottlieb S., Shu C. and Tadmor E. *Strong stability-preserving high-order time discretization methods* SIAM Rev., 43, 2001:1 pp. 89-112.

[4] LeVeque, R., *Numerical Methods for Conservation Laws*, Birkhauser-Verlag, 1990

[5] Marquina, Antonio. *Local piecewise hyperbolic reconstruction of numerical fluxes for nonlinear scalar conservation laws*, SIAM J. Sci. Comput., 15:4, 1994, pp. 892–915

[6] Patankar, S. *Numerical heat Transfer and Fluid Flow*, Series in Computational Methods in Mechanics and Thermanl Sciences, Hemisphere Publishing Corporation 1980, USA.

[7] Artebrant, Robert and Schroll, H. Joachim. *Limiter-free Third Order Logarithmic Reconstruction* To appear.

[8] Artebrant, Robert and Schroll, Hans Joachim. *High-resolution Riemann-solver-free methods for conservation laws* Hyperbolic problems: theory, numerics, applications, Springer Verlag 2003, pp. 305–314

[9] Schroll, H. Joachim. *Relaxed high resolution schemes for hyperbolic conservation laws* , J. Sci. Comput., vol. 21:2, 2004, pp. 251–279

[10] Hall, Oskar and Schroll, Hans Joachim and Svensson, Fredrik. *High-resolution simulation of inviscid flow in general domains*, Internat. J. Numer. Methods Fluids, 47:10-11, pp. 1061–1067

[11] Serna, Susana and Marquina, Antonio. *Power ENO methods: a fifth-order accurate weighted power ENO method* , J. Comput. Phys., 194:2, pp. 632–658

[12] Winterbone, D., Pearson, R., *Theory of engine manifold design*, Professional Engineering Publishing, 2000