Distributed Parameter Pneumatics

Felix Fischer[1] Katharina Schmitz[1]

[1]RWTH Aachen University – Institute for
Fluid Power Drives and Systems (ifas), Germany, {f.fischer,katharina.schmitz}@ifas.rwth-aachen.de

Abstract

Pneumatics is a branch of engineering that deals with the use of pressurized air or gases to create mechanical motion. It involves the study and application of systems and components such as air compressors, valves, cylinders, and actuators to control and transmit power through the use of compressed air. For highly dynamic events in pneumatic systems, such as fast switching processes in automation technology, lumped-parameter simulation is not sufficient to correctly calculate the pressure build-up in pipes. The propagation and reflections of different pressure waves and refraction waves cannot be accounted for by the zero-dimensional models provided by the `Modelica.Fluid` library. Therefore, this paper presents a method for calculating such events using the finite volume method. The library presented in this paper uses Godunov's scheme and an arbitrary Riemann solver and gas model to calculate the time evolution inside 1D or 2D discretized pneumatic components, as well as systems composed of these components.

Keywords: pneumatics, simulation, partial differential equation, distributed parameters, library

1 Introduction

The *Distributed Parameters Pneumatics* library allows for the calculating of transient events in pneumatic systems composed of pipes, valves, open and closed endings as well as connecting components such as T-connectors. These components are described in partial differential equations. These kinds of models consider the spatial distribution of parameters, such as temperature, density, and pressure; they are called *distributed parameter* models. This paper describes the theoretical foundation for the library as well as the core details of its implementation.

The theoretic background of the library discussed in this paper is introduced in section 2. The details of the implementation using Modelica is presented in section 3. The components of the library are validated using the analytical results of Sod-like tests, see section 4, as well as using experiments in section 5.

1.1 Motivation

While the *Modelica fluid* library is a suitable tool for calculating slow pressure changes in fluid systems used in hydraulics, pneumatics, and process engineering, it cannot be used to describe the fast fluctuations in pneumatic systems in the build-up phase of increasing or decreasing pressure. This is important, for example, when analyzing the interaction of different fast-moving actuators in an automation system. Another example of an application outside of pneumatics where the description of highly dynamic movements of gases is essential is gas transport in process engineering.

`Modelica.Fluid` uses zero-dimensional components, described by a single set of ordinary differential equations. These equations calculate changes in the quantities pressure p, temperature T, and density ρ only in terms of time and not of space. Therefore, these components can only describe the overall change of the averaged quantities inside of them over time. These types of models are called *lumped parameter* systems.

This is especially relevant for directional components like T- or X-pieces. The ideal T-connectors implemented in *Fluid* do not distinguish between the different directions. Therefore, a pressure wave entering an X-piece would be transmitted immediately to all three other sides without any difference and without any time delay between them. In a real X-piece, most of the pressure wave propagates to the opposite side.

A possible solution for this problem is the discretization of a single pipe component into a set of smaller pipes, which are described by the same underlying ordinary differential equations as the primitive component. This approach is called the finite volume method for solving partial differential equations. *Modelica Fluid* provides a discretized model with `Modelica.Fluid.Pipes.DynamicPipe`.

1.2 Sod Test

The standard method for the evaluation and test of finite volume method for gas dynamics is the Sod Test. This test is a fictional shock tube experiment. At time $t = 0$, the left half of the tube contains an ideal gas at high pressure, whereas the right half contains gas at a lower pressure. Both halves have an equal diameter and are directly connected. Initially, the gas in the whole tube is at the same temperature.

While the original paper by Sod uses a certain set of fixed and dimensionless initial pressures p, densities ρ and velocities $v = 0$, the analytical solution is known and well studied for every chosen set of start values (Sod 1978). Thus, can be used for the verification of gas dynamics simulations. The set-up of a Sod-like test and the pressure and rarefaction waves occurring in this test can be seen in Figure 1.

The standard method of initialization *Modelica Fluid* uses the *system* model. It includes the initial pressures as well as the initial temperature and the initial velocities. The

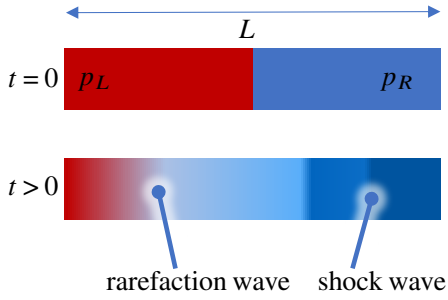

Figure 1. Set up and time evolution of the Sod test

density ρ can be calculated from the pressure and the temperature using the chosen gas law (e.g., the ideal gas law). In this work, the same method of initialization is used and therefore the systems are initialized using realistic values rather than abstract standard values. The start values are listed in Table 1. The Medium is Air. In the one-dimensional case, the diameter of the pipe has no influence on the results.

Table 1. Parameters of Sod-like test used in this work

Description	*Symbol*	*Start value*
Initial temperature	T	20 °C
Pressure in the left half	p_L	6 bar
Pressure in the right half	p_R	1 bar
Length of the pipe	L	2 m
Discretization	N	100
Modelica solver	DASSL	
Tolerance		1×10^{-6}

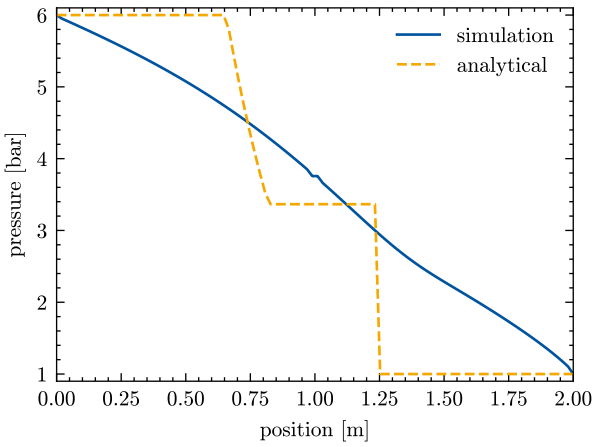

Figure 2. Sod test of staggered `DynamicPipe` at 1 ms

A sod test of the staggered model `DynamicPipe` from *Modelica Fluid* can be seen in Figure 2. There is a large discrepancy between the analytical result and the simulation (Isaac Backus 2017). Thus, this library is not suitable for highly dynamic gas simulations.

2 Theoretic Background

This section contains a brief introduction into the theoretic background of the simulation library presented in this work.

2.1 Euler Equations

This library is based on the conservative formulation of the Euler equations. The Euler equations are a simplified version of the Navier-Stokes equations. They describe the flow of a fluid without considering the thermal conductivity and viscosity.

Conservative formulations of partial differential equations guarantee that the amount of conserved quantities entering a finite volume is equal to the amount leaving it. Even if the discretization is very broad, the total and the local balances of these quantities are conserved. In non-conservative schemes, the quantities are not conserved and therefore there can be erroneous sources and sinks in the calculated solution. Therefore, the error in the conserved quantities is only acceptable, if the grid is fine enough (Ferziger, Perić, and Street 2020).

The conservative formulation of the one-dimensional Euler equations takes the following form (Toro 2009, p. 30):

$$0 = \frac{\partial}{\partial t}\mathbf{U} + \nabla\mathbf{F}(\mathbf{U}) \tag{1}$$

$$\mathbf{U} := (\rho, \rho u, E)^\intercal \tag{2}$$

$$\mathbf{F} := (\rho u, \rho u^2 + p, u(E+p))^\intercal \tag{3}$$

$$E := \rho\left(\frac{1}{2}u^2 + e\right) \tag{4}$$

In these equations, ρ is the density of the gas, u is the velocity and p is the absolute pressure. The total energy E (see Equation 4) can be separated into a kinetic and an internal component. The specific internal energy $e(p,\rho)$ is a function of the density and the absolute pressure. The expression for e is dependent on the gas model used in the simulation (Toro 2009). The entries of $\mathbf{U}$ shown in Equation 2 are called conserved variables, whereas p, u, and ρ are called primitive variables. The vector $\mathbf{F}$ shown in Equation 3 is called the flux vector and is a function of $\mathbf{U}$.

2.2 Riemann Problem

In finite volume methods, each discrete volume has only one value for $\mathbf{U}$ (see Equation 2) at each time step. Therefore, if two neighboring cells differ in any primitive variable, there will be a discontinuity in the initial condition of the partial differential equation. Such a partial differential equation with an initial condition that is constant everywhere except for a single discontinuous jump is called a *Riemann problem*.

The Riemann problem can be imagined as a miniature version of the Sod test at every cell boundary, see Figure 1. Analogous to the shock tube, the analytical solution is known and can be used to calculate the time evolution of every single finite volume.

DOI
10.3384/ecp20485

The finite volume method used in this work is *Godunov's scheme*. In Godunov's scheme, the spatial derivative in Equation 1 is calculated numerically by evaluating the flux vector $\mathbf{F}$ at the intercell boundary. The one-dimensional Godunov's method is given by (Toro 2009, p. 177):

$$\frac{\partial}{\partial t}\mathbf{U}(t,x) = -\frac{1}{\Delta x}\left(\mathbf{F}_{i+\frac{1}{2}} - \mathbf{F}_{i-\frac{1}{2}}\right) \tag{5}$$

$$\mathbf{F}_{i\pm\frac{1}{2}} := \mathbf{F}\left(\mathbf{U}_{i\pm\frac{1}{2}}\right) \tag{6}$$

$$\mathbf{U}_{i\pm\frac{1}{2}} := \mathbf{U}\left(t, x_i \pm \frac{\Delta x}{2}\right) \tag{7}$$

$$\Delta x := x_{i+1} - x_i \tag{8}$$

Here, t is the continuous time and x_i the discrete position of a specific finite volume with index i. $x \pm \frac{\Delta x}{2}$ represents the position of the boundary between the volume at x_i and the neighboring volume at $x_i \pm \Delta x$, see Figure 3.

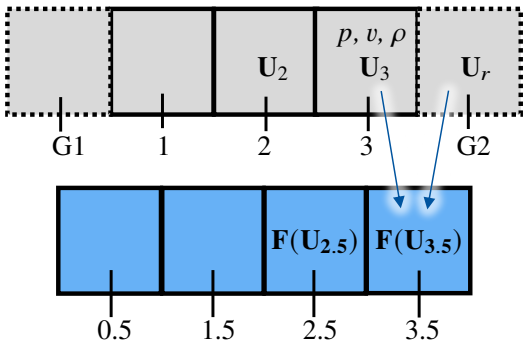

Figure 3. Discretizations used in Godunov's scheme, including ghost cells

The numerical method used to calculate $\mathbf{F}(\mathbf{U}(t, x \pm \Delta x))$ is called a *Riemann solver*. If the value $\mathbf{U}$ at the intercell boundary is calculated using the analytical solution, the solver is called an exact solver, otherwise, it is called an approximate solver. Compared to approximate Riemann solvers, exact solvers are more complex to construct and implement, and they are more computationally expensive. This work uses the approximate Harten-Lax-van-Leer-Contact (HLLC) and local Lax-Friedrichs solver (Toro 2016). These solvers take only the two neighboring volumes into account and are therefore called first-order methods, as shown in Equation 9.

$$\mathbf{F}_{i+\frac{1}{2}} = f(\mathbf{U}_i, \mathbf{U}_{i+1}) \tag{9}$$

The function f represents the first-order approximate Riemann solver. The time derivative in Equation 5 is solved by the Modelica solver.

2.3 Two Dimensional Formulation

To create a system that is more complicated than a single straight pipe with linear components connected to it, two-dimensional components are needed. These 2D components must be discretized in two dimensions, and a two-dimensional formulation of the finite volume method is needed. The main difference between the one- and two-dimensional algorithms is the additional component of the velocity in the y-direction which must be considered. Therefore, the state vector $\mathbf{U}$ needs an additional fourth component.

The conservative formulation of the two-dimensional Euler equations can be described as the following (Toro 2009, p. 104):

$$0 = \frac{\partial \mathbf{U}}{\partial t} + \frac{\partial \mathbf{F}_x}{\partial x} + \frac{\partial \mathbf{F}_y}{\partial y} \tag{10}$$

$$\mathbf{U} := \left(\rho, \rho u_x, \rho u_y, E\right)^\intercal \tag{11}$$

$$\mathbf{F}_x := \left(\rho u_x, \rho u_x^2 + p, \rho u_x u_y, u_x(E+p)\right)^\intercal \tag{12}$$

$$\mathbf{F}_y := \left(\rho u_y, \rho u_x u_y, \rho u_y^2 + p, u_y(E+p)\right)^\intercal \tag{13}$$

Here, u_y is the gas velocity in y-direction (Schulz-Rinne, Collins, and Glaz 1993).

This differential equation is solved numerically using a two-dimensional Godunov's scheme, as described in Equation 5. In this case, the flux vectors at all four intercell boundaries have to be calculated using a 2D version of a Riemann solver. In this work, an adapted version of the local Lax-Friedrichs solver has been used for two 2D discretized meshes.

3 Implementation in Modelica

This section describes how this library is structured and how the finite volume method described in section 2 is implemented in Modelica.

3.1 General Structure

The general structure closely follows the structure of the `Modelica.Fluid` library, to keep the library as compatible with it as possible. The models make use of replaceable packages and inheritance to keep the code reusable. Due to this structure and the use of a sub-package, it is possible to quickly exchange the selected gas model or the Riemann solver. The internal structure of the models is based on the finite volume library presented by Sielemann (Sielemann 2012b).

The package `TransientPneumatics` is separated into three sub-packages:

1. `Parts`: This package contains the useable components in this library, such as pipes and valves. Furthermore, it contains the sub-package `Base` in which the one- or two-dimensional discretized pipe sections are located. These discretized sections are used by the regular parts as attributes.
2. `Solvers`: This package contains the different selectable Riemann-Solvers (`OneD`, `TwoD`) and gas models (`Media`).
3. `Systems`: This package contains the base template for new systems as well as simulation models for testing this library.

3.2 Components

The different components contain the initialization, the border conditions, and the connectors. The discretization and the solutions of the partial differential equations are delegated to the reusable models in the sub-package `TransientPneumatics.Parts.Base`. The general structure of a component is shown in Listing 1.

Listing 1. Basic structure of a component in this library

```
within TransientPneumatics.Parts;
model Part "Example of a part"
  import Modelica.Units.SI;
  replaceable package Medium
    = TransientPneumatics
        .Solvers.Media.Ideal.Example
    constrainedby TransientPneumatics
        .Solvers.Media.Base
    "Medium";
  replaceable package Solver
    = TransientPneumatics.Solvers.HLLC(
    redeclare package Medium = Medium)
    constrainedby
         TransientPneumatics.Solvers.OneD
    "Solver";
  Modelica
      .Fluid.Interfaces.FluidPort_a left(
    redeclare package Medium=Medium)
    "Left connector";
  Modelica
      .Fluid.Interfaces.FluidPort_b right(
    redeclare package Medium=Medium)
    "Right connector";
  parameter Integer N = 20
  "Number of finite volumes";
  public
    SI.AbsolutePressure pressure[N]
    "Absolute pressure in component";
  protected
    Base.PipeBase discretization(
      redeclare package Medium=Medium,
      redeclare package Solver=Solver,
      N=N)
      "Reusable base model";
end Part;
```

The base models in `Parts.Base` have the following structure:

Listing 2. Basic structure of the base model containing the logic of every pipe section

```
within TransientPneumatics.Parts.Base;
model PipeBase
  "Base class for pipe sections"
  import Modelica.Units.SI;
  replaceable package Medium
    = TransientPneumatics
        .Solvers.Media.Ideal.Example
    constrainedby TransientPneumatics
        .Solver.Media.Base "Medium";
  replaceable package Solver
    = TransientPneumatics.Solvers.HLLC(
    redeclare package Medium = Medium)
    constrainedby TransientPneumatics
        .Solvers.OneD "Solver";
  parameter Integer N = 20
    "Number of finite volumes";
  parameter SI.Length L = 1
    "Length of the pipe";
  parameter SI.Diameter diameter = 0.001
    "Diameter of pipe";
  protected
    final SI.Area cross_section
      = Modelica
          .Constants.pi * diameter^2 / 4
      "Cross section of pipe";
    final SI.Length delta_x = L/N;
    Real U[
        N, 3] "Conserved variable vectors";
  public
    Medium.ThermodynamicState
      volume_left, volume_right;
    SI.MassFlowRate
      flow_rate_left, flow_rate_right;
    // finite volumes at the border
    Medium.ThermodynamicState volume[N]
        "Records containing
             the primitive variables";
    SI.MassFlowRate flow_rate[N]
        "Mass flow rate in each volume";
    // kept public for initialization
  equation
    // the
        differential equations are put here
end PipeBase;
```

The primitive variables pressure and density are contained in the record `ThermodynamicState`. Additionally, the gas velocity u can be calculated using the mass flow rate $\dot{m}$ and the cross-section a:

$$u = \frac{\dot{m}}{\rho a} \tag{14}$$

Based on the primitive variables, the conserved variables $\mathbf{U}$ can be calculated according to Equation 2 in the function `Solver.primitiveToConserved`. The set of equations in Listing 2 is chosen to avoid the implementation of a function for the conversion from the conserved variables to the primitive variables, because there is no function to set a `thermodynamicState` record from the pressure and the density in the Modelica standard library (Sielemann 2012a). The function `Solver.monotoneFlux` calculates the flux vector as a function of the neighboring cells according to the solver, as seen in Equation 9.

The implementation of the sets of differential equations into Modelica is shown in Listing 3.

Listing 3. Set of differential equation to be solved in each pipe section

```
equation
for i in 1:N loop
  U[i] = Solver.primitiveToConserved(
    volume[i], flow_rate[i],
    cross_section);
end for;
// Godunov's method
// left border
der(U[1]) = 1 / delta_x * (
  Solver.monotoneFlux(
```


DOI
10.3384/ecp20485

```
    volume_left, flow_rate_left,
    volume[1], flow_rate[1],
    cross_section)
  - Solver.monotoneFlux(
    volume[1], flow_rate[1],
    volume[2], flow_rate[2],
    cross_section));
// center section
for i in 2:N-1 loop
  der(U[i]) = 1 / delta_x * (
    Solver.monotoneFlux(
      volume[i-1], flow_rate[i-1],
      volume[i], flow_rate[i],
      cross_section)
    - Solver.monotoneFlux(
      volume[i], flow_rate[i],
      volume[i+1], flow_rate[i],
      cross_section));
end for;
// right border
// analogous to left border
```

3.2.1 2D Components

The two-dimensional base models are implemented analog to Listing 2; in this case, the conserved variables are stored in an $N \times M$ array: `U[N, M, 3]`. For 2D components, it needs to be differentiated between pipe section with round or rectangular cross-sections. When implementing rectangular cross-sections, it must be considered, that depending on the discretization the cross-section for flow in the x-direction can differ. In the case of circular cross-sections, the cross-section $a(y)$ for flow in the y-direction (perpendicular to the orientation of the pipe) depends on the y-position, as indicated in Figure 4.

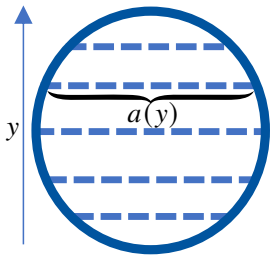

Figure 4. Cross-section of circular 2D pipes

3.3 Connectors

Each component is connected to the neighboring components using connectors. The connectors are the same connectors used in `Modelica.Fluid`. In this work, the ghost cell method is used to connect the connectors to the finite volumes. This approach is based on the work presented by López (López 2006). Due to the connectors being identical to the default ports, it is possible to connect the distributed parameter components developed in this work to the concentrated parameter components contained in `Modelica.Fluid`. This is demonstrated in Figure 5, where a graphical representation of a system in OpenModelica's OMEdit can be seen.

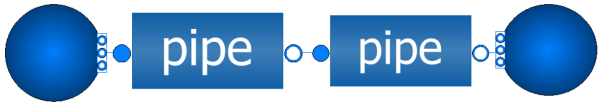

Figure 5. Two distributed parameter pipes connected to two concentrated parameter endings from `Modelica.Fluid` in OMEdit using connectors

The ghost cells on the left side of the base pipe are defined by the additional values `volume_left` and `flow_rate_left`. The variables for the right ghost cells are accordingly named `volume_right` and `flow_rate_right`. These values are related to the connectors by the following equations in the model of the parts shown in Listing 1:

Listing 4. Relation between the ghost cells and the connectors

```
discretization.volume_left
  = Medium.setState_ph(
    left.p, actualStream(left.h_outflow));
discretization.volume_right
  = Medium.setState_ph(
    right.p, actualStream(
      right.h_outflow));
discretization.flow_rate_left
  = left.m_flow;
discretization.flow_rate_right
  = - right.m_flow;
left.m_flow = discretizaion.flow_rate[1];
right.m_flow = -discretiation.flow_rate[N];
left.h_outflow = Medium.specificEnthalpy(
  discretization.volume[1]);
right.h_outflow = Medium.specificEnthalpy(
  discretization.volume[N])
```

In this work, the direction of flow is defined as left to right. A negative mass flow rate would therefore indicate a flow from right to left. Therefore, the sign for flow entering from the right connector must be inverted.

3.4 Pipe Endings

After enough time, every wave propagating in a finite pipe will hit either a closed or an open pipe ending. In both cases, the wave will be reflected on the ending and another wave will travel in the opposite direction. These open and closed boundary conditions are implemented into this library as separate models using the ghost cell approach shown in (Kratschun 2020).

3.4.1 Closed Ending

In the case of the reflection at a closed ending, the reflective boundary condition states that there is no velocity component in the x-direction at the boundary. This can be implemented using with the ghost cell approach by adding another volume with equal, but opposite velocity to the neighboring volume (LeVeque 2012, Chapter 7). Analogous to the connectors, a reflection on the left border of a part (Listing 1) can be implemented by adding the following lines:

Listing 5. Reflection on a closed ending at the left border of a part

```
discretization.volume_left
```

```
  = discretization.volume[1];
discretization.flow_rate_left
  = -discretization.flow_rate[1];
```

3.4.2 Open Ending

In the case of reflection at an open end, the reflective boundary condition states that there can be any velocity at the boundary, but there are conditions on the intensive variables on the volume next to the open end. The ghost cell at this end has the same velocity as the connected end, but depending on the direction of the flow, the temperature, and the pressure are fixed. Due to conservation of energy, the gas velocity at the end will overshoot compared to the wave packets inside the tube. Thus, the reflection at an open tube end can be intuitively understood to be caused by the discontinuity in acoustic impedance. The open end can be implemented by using the following additional lines:

Listing 6. Reflection on an open ending at the left border of a part

```
parameter SI.AbsolutePressure p_start
  = system.p_ambient
  "Environmental pressure";
parameter SI.Temperature T_start
  = system.T_ambient
  "Environmental temperature";
equation
  if discretization.flow_rate[1] > 0
  then
    discretization.volume_left
      = Medium.setState_pT(
        p_start, T_start);
  else
    discretization.volume_left
      = Medium.setState_pT(
        p_start, discretization.T[1]);
  end if;
  discretization.flow_rate_left
    = discretization.flow_rate[1];
```

3.5 Valve

Sod-like tests can be used for the analytical validation of a model. When trying to replicate a shock tube experiment on a test rig, a rapid 2/2-valve is needed, preferable switch times below 1 ms. But even in that case, the valve will have an influence on the propagation of the pressure waves, which cannot be neglected. Therefore, a model for valves is needed for the experimental validation of this library.

In this work, the valve is modeled as a plate orifice with changing diameter (Kratschun 2020). An orifice can be implemented into a finite volume method by calculating the flux vectors F based on the orifice equation (Schmitz 2022). The orifice equation is only valid for ideal gases, and it depends on the isentropic exponent κ.

$$\kappa = \frac{C_p}{C_v} \tag{15}$$

C_p is the heat capacity at constant pressure and C_v is the heat capacity at constant volume. For ideal gases, κ is constant. If the pressures are not very high, the ideal gas law is usually a good approximation. Therefore, κ is calculated as the fraction of heat capacities for all gas laws and is potentially not constant.

The orifice equation is only valid for stationary gases. To include dynamic effects, this work uses the total pressure instead of the static absolute pressure if the velocity is directed towards the orifice:

$$p_{\text{tot.}} = p + \rho u^2/2 \tag{16}$$

According to the orifice equation, gas will only ever flow from the side with higher pressure to the side with lower pressure. In the following, it is assumed that the pressure to the left of the orifice p_l is higher than the pressure to the right p_r. These equations can be implemented bidirectionally with Modelica by using conditional statements.

The velocity of the gas flowing through the orifice u_v increases when the pressure ratio Π decreases up to a critical ratio $\Pi_{\text{crit.}}$. For lower ratios, the velocity remains constant (Schmitz 2022, p. 45):

$$\Pi := \frac{p_r}{p_l} \tag{17}$$

$$\Pi_{\text{crit.}} = \left(\frac{2}{\kappa+1}\right)^{\frac{\kappa}{\kappa-1}} \tag{18}$$

$$u_v = c_d a_o \sqrt{p_l \rho_l} \cdot \begin{cases} \psi(\Pi_{\text{crit.}}) & \Pi \le \Pi_{\text{crit.}} \\ \psi(\Pi) & \Pi > \Pi_{\text{crit.}} \end{cases} \tag{19}$$

$$\psi(\Pi) := \sqrt{\frac{2\kappa}{\kappa-1}\left(\Pi^{\frac{2}{\kappa}} - \Pi^{\frac{\kappa+1}{\kappa}}\right)} \tag{20}$$

Here, ρ_l is the pressure to the left of the orifice and a_o is the area of the opening of the orifice. c_d denotes the discharge coefficient and is dependent on the geometry of the orifice. The components of the new flux vector can thus be calculated depending on the condition of the flow:

$$F_1 = u_v \rho_l \tag{21}$$

$$F_2 = \begin{cases} p_l \Pi_{\text{crit.}} & \Pi \le \Pi_{\text{crit.}} \\ p_r & \Pi > \Pi_{\text{crit.}} \end{cases} \tag{22}$$

$$F_3 = u_v \left(\rho_l \left(\frac{1}{2} u_v^2 + e(p_l, \rho_l)\right) + F_2\right) \tag{23}$$

In Equation 23, $e(p,\rho)$ denotes the specific internal energy as a function of the absolute pressure and the density of the gas.

When a pressure wave reaches a partially closed valve or an orifice from either direction, some fraction of the wave will be reflected at the orifice while another fraction will be transmitted according to the orifice equation. To include this effect into the model of the orifice, this work uses the combined flux approach presented in (Kratschun 2020):

$$\mathbf{F} = \frac{a_o}{a}\mathbf{F}_c + \left(1 - \frac{a_o}{a}\right)\mathbf{F}_o \tag{24}$$

Where $\mathbf{F}_c$ is the flux created by the reflection on a closed ending, (see subsection 3.4) and $\mathbf{F}_o$ orifice flux presented in this section.

DOI
10.3384/ecp20485

3.6 Stability of the Simulation

The simulations using the staggered components in this library are stable. Usually, when using explicit schemes for hyperbolic differential equations, the Courant-Friedrichs-Lewy (CFL) condition must be obeyed as a necessary condition for stability (Courant, Friedrichs, and Lewy 1928, Page 61):

$$\frac{|u|\Delta t}{\Delta x} \leq 1 \tag{25}$$

To enforce or check this condition, it would be necessary to check the current simulation time step Δt in run time, which is not possible in Modelica. Therefore, the finite volume method should not be used with explicit Modelica solvers like the Euler method. In this work, the implicit DASSL solver has been used (Petzold 1982), which is unconditionally stable, although slower than typical explicit solvers. In case of an event in Modelica, DASSL will reduce to the Euler method. Therefore, the stability of the simulation can only be guaranteed by avoiding any events, or by using an a-stable solver. An example for an a-stable solver are implicit Runge-Kutta methods.

4 Analytical Validation

The different components, solvers, and gas laws are validated using the analytical results of the sod test described in subsection 1.2. A graphical representation of the system can be seen in Figure 5. In this system, two pipes with equal lengths are initialized with different pressures and connected using a connector in the center. The simulation parameters are listed in Table 1.

4.1 Solvers

A comparison of the different solvers can be seen in Figure 6. The results with both solvers follow the analytical

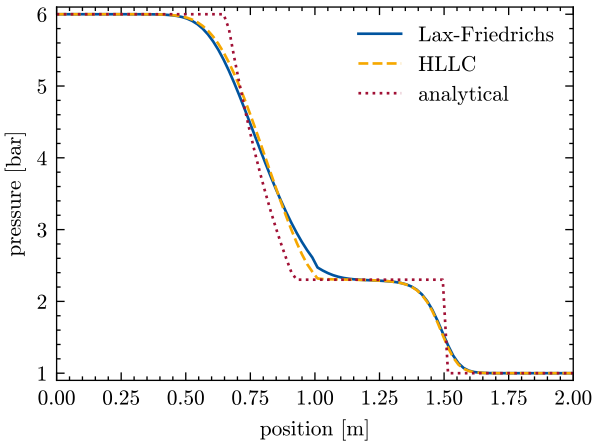

Figure 6. Comparison of Sod tests with the local Lax-Friedrichs Solver and the HLLC solver at 1 ms

solution. The simulated solutions are approaching the analytical result with increasing degree of discretization. Thus, the solvers have been implemented successfully.

The results by the HLLC solver are closer to the analytical solution at every point. The difference in calculation time between both solvers is negligible. For this reason, the HLLC solver is preferred for all simulations in this library.

4.2 Gas Laws

In this work, multiple gas laws have been implemented. Besides the ideal gas law, the Van der Waals gas law has been implemented as an example of a real gas law. The main difference between the different gas laws is the relation between the primitive variables and the specific internal energy $e(p,\rho)$. A comparison of the gas laws at high pressure can be seen in Figure 7. The starting pressure

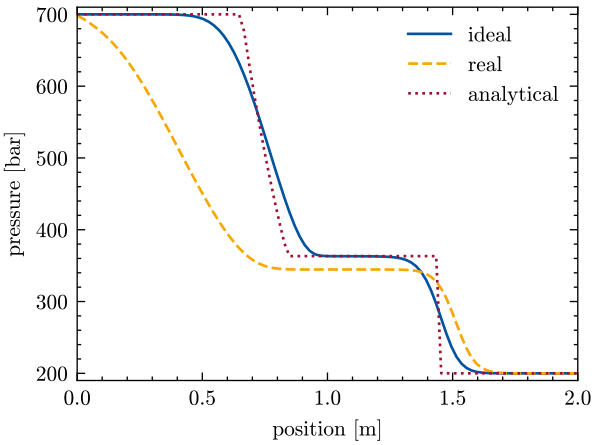

Figure 7. Comparison of Sod tests with real and ideal gas law at a very high-pressure drop at 1 ms

in the left half is 700 bar and on the right half 200 bar. At the initial conditions shown in Table 1, there is no visual difference between both solvers. For the high pressures shown in Figure 7, the results differ significantly. The analytical solution is only based on the ideal gas law and agrees therefore with the calculation based on the same law.

The default record `ThermodynamicState` contained in `Modelica.Fluid` contains the pressure and the temperature as basic variables. This leads to instabilities when solving a system using the DASSL solver of OpenModelica. The *Distributed Parameter Pneumatics* simulations library thus contains an alternative implementation of both `Media` and `ThermodynamicState` with the pressure and the density as internal variables, as seen in Listing 7. With this alternative record, the simulation runs flawlessly.

Listing 7. "Replacement for thermodynamic state record"

```
record ThermodynamicState
    "Custom thermodynamic state model"
  public
    SI.AbsolutePressure p(start = 1e5);
    SI.Density rho(start = 1.2);
end ThermodynamicState;
```

The two needed functions `Media.setState_ph` and `Media.setState_pT` return this record as a function of the pressure and the specific entropy or the pressure and the

temperature respectively. In the case of the Van-der-Waals gas equation, there is no closed-form expression for these functions, and they must be computed iteratively. This increases the computation time, which is about a factor of 3 larger compared to the ideal gas law. Thus, for low pressures, which are the most relevant for pneumatic applications, it is preferred to use the ideal gas law with this library.

4.3 2D Components

A sod test for a two-dimensional component with a circular cross-section can be seen in Figure 8. The simulated

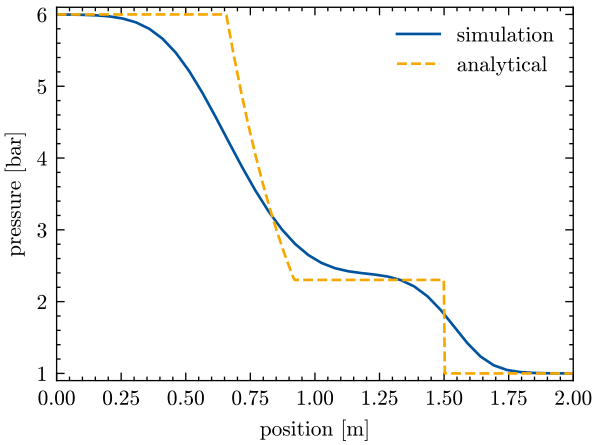

Figure 8. Sod test of a 2D pipe section with circular cross-section at 1 ms

pipe is circular, so the pressure curve is calculated in the x-direction, parallel to the pipe.

The pipe is discretized into 20 segments in the x-direction and 7 segments in the y-direction. Due to the smaller discretization compared to the simulation shown in subsection 4.1, there is a greater difference between the simulation and the analytical results. In the two-dimensional case, the calculation time is multiple times larger compared to the one-dimensional case. Therefore, simulations with a very high two-dimensional discretization are not feasible on regular hardware. Another reason for the decreased accuracy in this simulation is the use of a 2D-Lax-Friedrichs solver compared to the HLLC solver used in the one-dimensional case. Nevertheless, the simulated result still approaches the analytical result with increasing refinement of the grid. Due to the changing cross-section, it is not possible to perform a useful Sod test in the y-direction.

In the case of the rectangular cross-section, the same validation has been performed in both the x-direction and y-direction and in both cases, the simulation reaches a similar agreement with the analytical results as for the circular cross-section.

The implementation of branched connectors like T-connectors is the principal use of two-dimensional components. These components can be constructed by the combination of several two-dimensional sections with either circular or rectangular cross-sections. The rectangular cross-sections are needed to attach a pipe section to another pipe with an angle of 90° because the pipes with round cross-section taper towards the edge in the y-direction.

5 Experimental Validation

The simulation library presented in this work has been experimentally validated using a test rig for shock test experiments. The experiment allows the validation of components in this library, even if there is no known analytical solution. This is especially relevant for the open and closed pipe endings, as well as for the valve.

A pneumatic circuit diagram of the test rig can be seen in Figure 9. The test rig uses two absolute pressure sensors, one in the high-pressure section and one in the low-pressure section, to measure the environmental pressure and the exact pressure of supplied air. Additionally, there are five highly sensitive piezoelectric relative pressure sensors along the pipe. These sensors measure the exact curve of the pressure at their position.

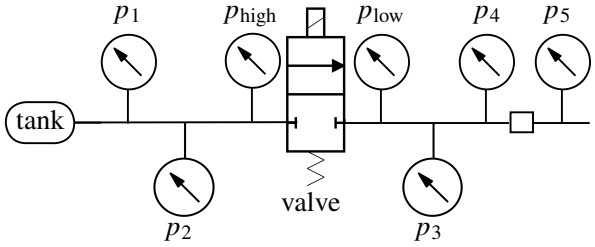

Figure 9. Pneumatic scheme of the test rig used in this work

The low-pressure half is connected to the high-pressure half by a quick-acting pneumatic valve. According to the manufacturer, the valve has a cycle time from closed to open to closed of about 1 ms. In the experiment discussed in this work, the valve starts in its closed state and is then opened and closed as fast as possible. No analytical solution exists for this system. The experimental conditions and the geometry of the pipes are listed in Table 2.

Table 2. Experimental Parameters

Variable	*Value*
Environmental temperature	21 °C
Environmental pressure	1.01 bar
Pressure in the tank	3.15 bar
Length of the left pipe	0.39 m
Length of the right pipe	1.985 m
Diameter of the pipe	0.7 cm
Distance between the valve and p_4	0.59 m
Fluid used in experiment	air

A model of the test rig shown in Figure 9 has been set up using `TransientPneumatics`. The ending of the tank to the left, as well as the connection to the environment to the right, have been modeled using open endings, which have been presented in subsection 3.4.

DOI
10.3384/ecp20485

When implementing the valve, the following parameters can be set in the simulation, see subsection 3.5:

1. The maximal inner diameter of the orifice a_o
2. The discharge coefficient c_d

There is no obvious way to map these parameters to properties of the valve which can be found in its datasheet. The same is true for the signal chosen in Modelica, which controls the degree of opening of the valve. The simulation parameters selected for this model can be found in Table 3.

Table 3. Simulation Parameters

Variable	*Value*
Discharge coefficient c_d	0.68
Inner diameter a_o	5.7 mm
Valve signal	trapezoidal
Opening time	0.4 ms
Open time	0.2 ms
Closing time	0.4 ms
Discretization of the left pipe	40
Discretization of the right pipe	110
Solver	DASSL
Tolerance	1×10^{-6}

A comparison between the measured and the simulated pressure curve at the fourth pressure sensor p_4 can be seen in Figure 10.

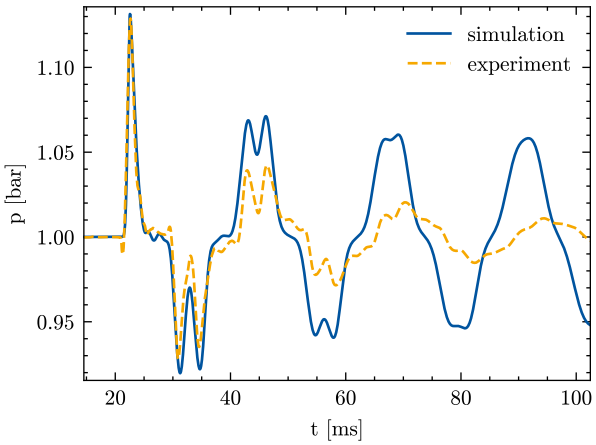

Figure 10. Comparison between the experiment and the simulation shown in Figure 9 measured at sensor p_4

The general shape of the first five double peaks match each other, and therefore it can be concluded, that the reflection at the open ending has been successfully implemented. In the last two peaks, the experiment shows a large dilation of the reflected pressure waves, which is not represented by the simulation. In the experiment, the amplitude of the oscillation decreases at a larger rate compared to the simulation. This is probably due to wall friction and, to a lesser degree, thermal transport through the walls of the pipe. Both effects cannot be yet simulated using this library.

A frequency analysis of the experiment and the simulation is depicted in Figure 11. The first peak corresponds

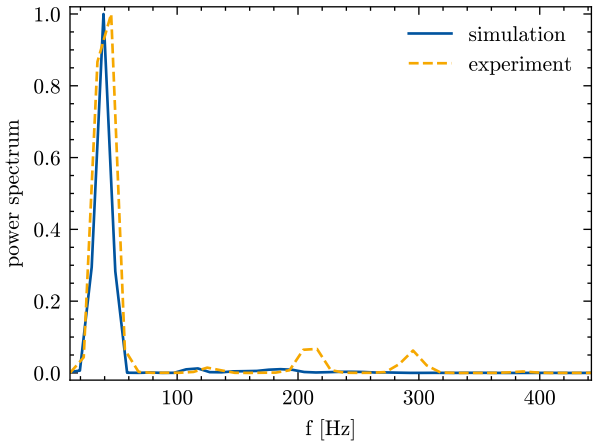

Figure 11. Comparison between the experiment and the simulation of the spectral power spectrum of the signal shown in Figure 10

to the principal eigenfrequency of the air column in the right pipe of the test rig. The fundamental tone of a tube of air, which is open at one and closed at the other, can be found approximately by the following equation (Rienstra and Hirschberg 2004):

$$\nu = \frac{V}{4L} \tag{26}$$

Where ν is the frequency of the fundamental tone, $V \approx 343\,\mathrm{m\,s^{-1}}$ is the speed of sound and L is the length of the pipe. This equation yields about 43.2 Hz for this experiment, which is in good agreement with the first peaks seen in Figure 11.

There are secondary peaks at higher frequencies seen in the experimental data, which are not present in the simulation. These peaks are probably due to partial reflection at the drilling holes of the sensors or at short cross-section jumps at the screw fittings. The frequency for two reflections at a closed ending is:

$$\nu = \frac{V}{2L} \tag{27}$$

According to this formula, the frequency for a reflection between the valve and p_4 is approximately 291 Hz; the distance between the valve and the connector in between $p4$ and $p5$ (see Figure 9) is 1.82 m which corresponds to 209 Hz.

Considering the assumed simplifications, this library has been successfully validated experimentally.

6 Summary, Evaluation, and Outlook

In this section, this paper is concluded by a summary of the results, as well as a critical evaluation of the presented method.

The compilation and simulation time for systems with two-dimensional components is considerable. Therefore, this library cannot be used for complex systems with many

two-dimensional connections. The presented method allows calculating events in systems with arbitrary gas laws. Therefore, this library can be helpful for calculating gas transport in process engineering. Such systems are smaller and have a simpler topology compared to pneumatic systems, which allows performing the simulation in a short time.

Realistic systems will contain cross-section jumps and elastic tubes, which still need to be implemented. The presented method can further be improved by including wall friction and thermal conduction. It is possible to include wall friction, thermal conduction, and elastic tubing by adding a source term to the differential equations shown in Listing 2. A cross-section jump can be included by using the analytical solution of the Riemann problem at a cross-section jump, similar to the orifice presented in subsection 3.5 (Han, Hantke, and Warnecke 2012).

The library presented in this work allows for the calculation of highly dynamic transient events in pneumatic systems consisting of pipes, valves, connections, and open or closed endings. The parts have been validated using the analytic solution of the Sod tests, as well as experimentally. In general, there is a good agreement between the simulation and the analytical and experimental results.

Acknowledgements

The authors thank the Research Association for Fluid Power of the German Engineering Federation VDMA for its financial support. Special gratitude is expressed to the participating companies and their representatives in the accompanying industrial committee for their advisory and technical support.

Nomenclature

Sym.	Meaning	Sym.	Meaning
a	cross-section	a_o	opening of orifice
C_p, C_v	heat capacity	c_d	disch. coefficient
E	total energy	e	sp. internal energy
$\mathbf{F}$	flux vector	f	general function
L	length of pipe	$\dot{m}$	mass flow rate
p	abs. pressure	t	time
$\mathbf{U}$	conserved var.	u	gas velocity
V	speed of sound	x, y	position
κ	heat cap. ratio	ν	frequency
Π	pressure ratio	ρ	density of gas
ψ	orifice function		

References

Courant, R., K. Friedrichs, and H. Lewy (1928). "Über die partiellen Differenzengleichungen der mathematischen Physik". In: *Mathematische Annalen* 100.1, pp. 32–74. ISSN: 0025-5831. DOI: 10.1007/BF01448839.

Ferziger, Joel H., Milovan Perić, and Robert L. Street (2020). *Computational methods for fluid dynamics*. Fourth edition. Cham: Springer. ISBN: 978-3-319-99693-6.

Han, Ee, Maren Hantke, and Gerald Warnecke (2012). "Exact Riemann Solutions to Compressible Euler Equations in Ducts with discontinuous Cross-Section". In: *Journal of Hyperbolic Differential Equations* 09.03, pp. 403–449. ISSN: 0219-8916. DOI: 10.1142/S0219891612500130.

Isaac Backus (2017). *Sod shock tube calculator*. URL: https://github.com/ibackus/sod-shocktube (visited on 2023-05-16).

Kratschun, Filipp (2020). "Transient Pneumatic System Simulation: Transiente Simulation pneumatischer Systeme". Dissertation. Aachen: RWTH. DOI: 10.2370/9783844073997.

LeVeque, Randall J. (2012). *Finite Volume Methods for Hyperbolic Problems*. Cambridge University Press. ISBN: 9780521810876. DOI: 10.1017/CBO9780511791253.

López, José Díaz (2006). "Shock Wave Modeling for Modelica.Fluid Library using Oscillation-free Logarithmic Reconstruction". In: *Proceedings of the 5th International MODELICA Conference*. Ed. by Martin Otter and Dirk Zimmer. Vienna: The Modelica Association, pp. 641–649. URL: https://modelica.org/events/modelica2006/Proceedings/sessions/Session6b2.pdf.

Petzold, Linda R (1982). *Description of DASSL: a differential/algebraic system solver*. Tech. rep. Sandia National Labs., Livermore, CA (USA).

Rienstra, Sjoerd W and Avraham Hirschberg (2004). "An introduction to acoustics". In: *Eindhoven University of Technology* 18, p. 19. URL: https://ayeghsoti.com/wp-content/uploads/2019/09/boek.pdf (visited on 2023-06-20).

Schmitz, Katharina (2022). *Fluidtechnik – Systeme und Komponenten*. 1. Auflage. Düren: Shaker Verlag. ISBN: 978-3-8440-8801-4.

Schulz-Rinne, Carsten W., James P. Collins, and Harland M. Glaz (1993). "Numerical Solution of the Riemann Problem for Two-Dimensional Gas Dynamics". In: *SIAM Journal on Scientific Computing* 14.6, pp. 1394–1414. ISSN: 1064-8275. DOI: 10.1137/0914082.

Sielemann, Michael (2012a). "Device-Oriented Modeling and Simulation in Aircraft Energy Systems Design". Dissertation. Hamburg: TU Hamburg-Harburg. DOI: 10.15480/882.1111.

Sielemann, Michael (2012b). "High-Speed Compressible Flow and Gas Dynamics". In: *Proceedings of the 9th International MODELICA Conference*. Ed. by Martin Otter and Dirk Zimmer. Linköping Electronic Conference Proceedings. Linköping: Linköping University Electronic Press, pp. 81–100. ISBN: 978-91-7519-826-2. DOI: 10.3384/ecp1207681.

Sod, Gary A. (1978). "A survey of several finite difference methods for systems of nonlinear hyperbolic conservation laws". In: *Journal of Computational Physics* 27.1, pp. 1–31. ISSN: 00219991. DOI: 10.1016/0021-9991(78)90023-2.

Toro, Elewterio F. (2009). *Riemann solvers and numerical methods for fluid dynamics: A practical introduction*. 3. ed. Berlin and Heidelberg: Springer. ISBN: 978-3-540-49834-6. URL: http://www.loc.gov/catdir/enhancements/fy1109/2009921818-d.html.

Toro, Elewterio F. (2016). "The Riemann Problem". In: *Handbook of Numerical Methods for Hyperbolic Problems - Basic and Fundamental Issues*. Vol. 17. Handbook of Numerical Analysis. Elsevier, pp. 19–54. ISBN: 9780444637895. DOI: 10.1016/bs.hna.2016.09.015.