Modellbildung in der Gasdynamik
-Ventilmodellbildung-

Bachelor Thesis
von
cand. aer. Ansgar Lechtenberg

durchgeführt am
Institut für Aerodynamik und Gasdynamik
der Universität Stuttgart
und beim DLR in Oberpfaffenhofen.

Stuttgart, im September 2014

www.DLR.de/jobs

Starten Sie Ihre Mission beim DLR.

Das DLR ist das Forschungszentrum für Luft- und Raumfahrt sowie die Raumfahrtagentur der Bundesrepublik Deutschland. Rund 7.300 Mitarbeiterinnen und Mitarbeiter forschen gemeinsam an einer einzigartigen Vielfalt von Themen in Luftfahrt, Weltraum, Energie, Verkehr und Sicherheit. Ihre Missionen reichen von der Grundlagenforschung bis hin zur Entwicklung von innovativen Anwendungen und Produkten von morgen. Wenn auch Sie sich für die Welt der Spitzenforschung in einem inspirierenden, wertschätzenden Umfeld begeistern, starten Sie Ihre Mission bei uns.

Für das **Institut für Systemdynamik und Regelungstechnik** in **Oberpfaffenhofen** vergeben wir eine

Bachelorarbeit

Im Bereich Modellbildung Gasdynamik

Ihre Mission:

Die Arbeitsgruppe Thermofluidsysteme befasst sich mit Systemen, deren Wirkungsweise sowohl auf der Thermodynamik als auch der Fluiddynamik, d.h. Strömungslehre, aufbaut. Um im Bereich der Flugzeugenergiesysteme Architekturen auslegen zu können, sind genaue Modelle von Thermofluidsystemen nötig. Das Team Energiesysteme beschäftigt sich mit der Erstellung und Verbesserung derartiger Berechnungsmodelle sowie darauf aufbauend mit der Auslegung und Optimierung von Systemarchitekturen. Verwendet wird hierfür die objektorientierte gleichungsbasierte Modellierungssprache Modelica.

Ihre Aufgaben umfassen die Weiterentwicklung einer Modellbibliothek zur eindimensionalen Simulation schneller kompressibler Strömungen sowie die anschließende Validierung Ihrer Arbeiten.

Aktuell wird bei systemdynamischen Simulationen mit Modelica vor allem die klassische Rohrhydraulik verwendet. In der öffentlichen Standardbibliothek sind hier ausgereifte Modelle für Rohre, Pumpen, Ventile etc vorhanden. In Bleedsystemen ist dieser Ansatz nicht ausreichend, da hier hohe Strömungsgeschwindigkeiten und potentiell Stöße auftreten können. Aus diesem Grund wurde am Institut eine Bibliothek entwickelt, welche auf konservativen Methoden aufbaut. Momentan enthält diese Bibliothek Randbedingungen, verschiedene Diskretisierungsmethoden und eindimensionale Rohrmodelle mit anpassbaren eindimensionalen Gittern, so dass z.B. Lavaldüsen abgebildet werden können. In Bleedsystemen treten allerdings nicht nur Rohre, sondern auch Butterflyventile auf. Ein solches Ventilmodell soll nun entwickelt werden. Dieses soll insbesondere für austauschbare Diskretisierungsverfahren unterschiedlicher Ordnung (u.a.: Upwind Muscl-Hancock method mit Minbee limiter und Roe monotone flux) in die bestehende Bibliothek integrierbar sein. Folgende Arbeitspakete wären entsprechend zu bearbeiten:

- Einlesen in Berechnungsverfahren für schnelle kompressible Strömungen
- analytische Beschreibung des Ventilverhaltens
- Implementierung in die bestehende Bibliothek
- Plausibilitätsbetrachtungen

DLR Deutsches Zentrum für Luft- und Raumfahrt

Zertifikat seit 2002 audit berufundfamilie

trendence Graduate Barometer 2012 DEUTSCHLANDS 100 Top-Arbeitgeber

Erklärung

Hiermit versichere ich, dass ich diese Bachelorarbeit selbstständig mit Unterstützung meiner Betreuer angefertigt und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.
Die Arbeit oder wesentliche Bestandteile davon sind weder an dieser noch an einer anderen Bildungseinrichtung bereits zur Erlangung eines Abschlusses eingereicht worden.

Statement of Originality

This thesis has been performed independently with support by my supervisors. It contains no material that has been accepted for the award of a degree in this or any other university. To the best of the candidate's knowledge and belief, this thesis contains no material previously published or written by another person except where due reference is made in the text.

Stuttgart, den _____ Unterschrift: _____

Übersicht

In der nachfolgenden Bachelor-Thesis werden verschiedene Möglichkeiten der Realisierung eines Ventils in der Modellierungssprache Modelica vorgestellt und anschließend diskutiert. Diese Ventilmodelle sollen in der Vorauslegung des Bleedsystem von Passagierflugzeugen Anwendung finden. Das Hauptaugenmerk liegt dabei auf der Ventilmodellierung mit Hilfe der Eulergleichungen. Dabei wird auf verschiedene Varianten nach Godunov zur Lösung dieses streng hyperbolische Differenzialgleichungssystems eingegangen.
Zur Validierung wird ein einfaches Model eines Rohrs mit Ventil mit Hilfe des CFD-Programms OpenFOAM vorgestellt, wobei die Erstellung des Rechengitters kurz beleuchtet wird.
Aus den verschiedenen physikalischen Ansätzen der Modelle entstehen sich zum Teil deutlich unterscheidende Ergebnisse. Diese Ergebnisse werden abschließend miteinander vor dem Hintergrund, dass mit Hilfe der Modelle in Modelica ein Schmetterlingsventil simuliert werden soll, verglichen.

Abstract

In the following bachelor thesis different possibilities of realization of a valve in the modeling language Modelica are presented and discussed. These valvemodels will take part in the dimensioning of bleedsystems of passenger aircrafts. Here, the main focus is on the valvemodelling using the Eulerequations. Therefor this thesis will deal with various variants by Godunov to solve this strictly hyperbolic differential equations.
For validation purpose a simple pipe with a valve using the CFD-program OpenFOAM is presented, wherby the creation of the computational grid is briefly discussed.
These various physically approches of the models results in partially obvious different findings. The results are finally compared against the background that a butterflyvalve is to be simulated using these models in Modelica.

Inhaltsverzeichnis

Aufgabenstellung iii

Übersicht v

Inhaltsverzeichnis vii

Nomenklatur ix

Abbildungsverzeichnis xi

1 Einleitung 1
- 1.1 Motivation und Zielsetzung . . . 1
- 1.2 Grundlagen . . . 2
  - 1.2.1 Einführung Modelica . . . 2
  - 1.2.2 Die Eulergleichungen . . . 4
  - 1.2.3 Riemannlöser nach Godunov . . . 5
  - 1.2.4 Numerische Flussberechnung . . . 7
  - 1.2.5 Gittererstellung für CFD-Programme . . . 14
  - 1.2.6 OpenFOAM . . . 15

2 Modellbildung der Ventile 19
- 2.1 Versuchsaufbau in Modelica . . . 19
- 2.2 Das Ventil aus der Standardbibliothek . . . 20
- 2.3 Zwei Ventile in der Gasdynamikbibliothek . . . 21
  - 2.3.1 Ventilberechnung über Ausflussfunktion . . . 21
  - 2.3.2 Das diskretisierte Ventil . . . 24
- 2.4 Ventilerstellung mit OpenFOAM . . . 27
  - 2.4.1 Erstellung des 2D-Gitters . . . 27
  - 2.4.2 Vorgeben der Randbedingungen . . . 30
  - 2.4.3 Simulation starten und Ergebnisse ausgeben . . . 30

3 Ergebnisse und Auswertung 31
- 3.1 Vergleich des Standardventils mit dem Ventil mit Ausflussfunktion . . . 31
- 3.2 Vergleich des diskretisierten Ventil mit dem mit Ausflussfunktion . . . 34

3.3 Vergleich des diskretisierten Ventil mit den Computational Fluid Dynamics (CFD)-Simulationen 41
3.4 Vergleich der unterschiedlichen Berechnung des numerischen Fluss 46
3.5 Fazit 48

4 Zusammenfassung und Ausblick **49**
4.1 Zusammenfassung 49
4.2 Ausblick 51

Literaturverzeichnis **53**

Anhang **I**

Nomenklatur

Lateinische Symbole

Symbol	Bedeutung	Einheit
A	Jacobi-Matrix	-
A	Querschnittsfläche	m^2
Av	Durchflussfaktor in SI-Einheiten	m^2
A_V	Querschnittsfläche des Ventilhalses	m^2
a	Schallgeschwindigkeit	$\mathrm{m\,s^{-1}}$
Cv	angewandter Durchflussfaktor in USA	USG/min
c_V	spezifische Wärmekapazität bei konstanten Volumen	$\mathrm{J\,kg^{-1}\,K^{-1}}$
D, d	Durchmesser	m
E	Energie pro Volumen	$\mathrm{J\,m^{-3}}$
e	Innere Energie	J
F	Fluss der Eulergleichungen	-
g	Erdbeschleunigung	$\mathrm{m\,s^{-2}}$
g	(mit Indizierung) numerischer Fluss	-
H	totale Enthalpie	J
K	Eigenvektoren der Jacobi-Matrix	-
Kv	angewandter Durchflussfaktor in Europa	$\mathrm{m^3\,h^{-1}}$
$\dot{m}$	Massenstrom	$\mathrm{kg\,s^{-1}}$
p	Druck	Pa
Q	Volumenstrom	$\mathrm{m^3\,s^{-1}}$
R	spezifische Gaskonstante	$\mathrm{J\,kg^{-1}\,K^{-1}}$
r	Radius	m
S	Quellterm in Eulergleichungen	-
S	(mit Indizierung) Ausbreitungsgeschwindigkeit einer Wellenfront	$\mathrm{m\,s^{-1}}$
T	Temperatur	K
t	Zeit	s
U	konservative Eulervariablen	-
u, v, w	Geschwindigkeitsvektoren in X, Y, Z-Richtung	$\mathrm{m\,s^{-1}}$
V	Volumen	m^3

Griechische Symbole

Symbol	Bedeutung	Einheit
α	lokale Wellenstärke im numerischen Fluss nach Roe	-
β	limitierender Faktor in Limiterfunktion	-
Δ	Differenz	-
κ	Isentropenexponent	-
λ	Eigenwerte der Jacobi-Matrix	-
$\lambda_{\max}$	maximale Wellenausbreitungsgeschwindigkeit im Rusanovfluss	$\mathrm{m\,s^{-1}}$
π	Kreiszahl	-
ρ	Dichte	$\mathrm{kg\,m^{-3}}$
φ	Winkelauslenkung	rad
Ψ	Ausflussfunktion	-
ω	Winkelbeschleunigung	$\mathrm{rad\,s^{-1}}$

Indizes

Index	Bedeutung
i	i-te Zelle
$i + \frac{1}{2}$	Wert am rechten Rand der i-ten Zelle
$i - \frac{1}{2}$	Wert am linken Rand der i-ten Zelle
L, R	linker, bzw. rechter Wert
n	(hochgestellt) n-ter Zeitschritt
t	partielle Ableitung nach der Zeit
x, y, z	partielle Ableitung nach der Raumrichtung

Abkürzungen

Kürzel	Bedeutung
CFD	Computational Fluid Dynamics
CFL	Courant-Friedrichs-Lewy
DLR	Deutsche Zentrum für Luft- und Raumfahrt
HLL	Harten-Lax-van Leer
MUSCL	Monotonic Upstream-Centered Scheme for Conservation Laws
TVD	Total-Variation-Diminishing

Abbildungsverzeichnis

1.1 Modelica Simulation eines Fadenpendels 3
1.2 Unterteilung einer DGL in lokale Riemannprobleme nach Godunov 6
1.3 Wellenausbreitung um einen Fixpunkt der Eulergleichungen 8
1.4 Wellenausbreitung um einen Fixpunkt des HLL-Verfahren 11
1.5 Wellenausbreitung um einen Fixpunkt nach Lax und Friedrichs 12
1.6 Schematische Funktionsweise einer Limiter-Funktion 13
1.7 Einfache Diskretisierung von Kreisen 15
1.8 O-Grid Diskretisierung eines Kreises 16

2.1 Versuchsaufbau für die Betrachtung der Ventile in Modelica 19
2.2 Funktionsprinzip der Konnektoren in der Gasdynamikbibliothek 22
2.3 Entstehung des Quellterms bei Querschnittsveränderung 25
2.4 Schaubild zur Flussberechnung im Ventil 26
2.5 Gegenüberstellung des komplexen Rohrgitters und des quasi 2D-Gitters . 27
2.6 Gegenüberstellung von Druck und Geschwindigkeit in OpenFOAM in 3D und 2D 28
2.7 Gitter einer Rohrleitung mit einem 40% geöffnetem Ventil 29

3.1 Zeitvergleich des Standardventils mit dem Ventil mit Ausflussfunktion bei geringer Druckdifferenz 32
3.2 Zeitvergleich des Standardventils mit dem Ventil mit Ausflussfunktion bei hoher Druckdifferenz 33
3.3 Gegenüberstellung der Massenströme des diskretisierten Ventil und des Ventils mit Ausflussfunktion 35
3.4 Gegenüberstellung der Ventilcharakteristiken eines realen Schmetterlingsventil [14] und denen aus Modelica 36
3.5 Gegenüberstellung des Druckverlaufs des Ventilmodel mit Ausflussfunktion und des diskretisierten Ventilmodels 37
3.6 Geschwindigkeit als Fläche über Inlet- und Outlet-Drücken für die Ventilstellung 26° 38
3.7 Geschwindigkeit als Fläche über Inlet- und Outlet-Drücken für die Ventilstellung 37° 38
3.8 Geschwindigkeit als Fläche über Inlet- und Outlet-Drücken für die Ventilstellung 53° 39

3.9 Geschwindigkeit als Fläche über Inlet- und Outlet-Drücken für die Ventilstellung 90° 39
3.10 Geschwindigkeitsprofil des vollständig geöffnetes Ventils aus OpenFOAM . 41
3.11 Vergleich der Ventilcharakteristiken berechnet durch OpenFOAM und Modelica bei einem Druck von 1.1bar auf 1bar 43
3.12 Vergleich der Ventilcharakteristiken berechnet durch OpenFOAM und Modelica bei einem Druck von 5.5bar auf 3.5bar 44
3.13 Vergleich der Druckkurven simuliert in Modelica und OpenFOAM 45
3.14 Vergleich der berechneten Massenströme der unterschiedlichen Diskretisierungen 47

1 Einleitung

1.1 Motivation und Zielsetzung

Im Zuge der Auslegung und Planung eines Flugzeuges wurde rechnergestützt die Entnahme der Zapfluft (Bleedair) aus den Triebwerken zur Bereitstellung der Frischluft in der Kabine modelliert und simuliert. Es stellte sich heraus, dass die berechnete Rohrleitungssystem (Bleedsysteme), welche am Computer einwandfrei arbeiteten, im Realen Stöße und Oszillationen aufweisen. Aus diesem Grund wurde das Deutsche Zentrum für Luft- und Raumfahrt (DLR) damit beauftragt, ein verbessertes Modell zur Vorhersage dieser Stöße zu entwickeln, um diese in zukünftigen Baureihen zu vermeiden. In diesem Zusammenhang entstand von Sielemann [13] eine Gasdynamikbibliothek für die Modellierungssprache Modelica, welche bereits Rohrleitungen und Randbedingungen numerisch diskretisiert und über die eindimensionale Eulergleichung berechnet. Obwohl die Standardbibliothek von Modelica bereits solche Rohrleitungsmodelle sowie diverse Ventilmodelle enthält, war dies notwendig, da die Rohrleitungssysteme der Standardbibliothek schon bei einfachen Stoßproblemen (Stoßrohr) zu unphysikalischen Oszillationen an den Stoßgrenzen neigt. Mit Hilfe der neu entwickelten Bibliothek und eines einfachen Ventilmodells ist es am DLR gelungen, die Stöße im Rohrleitungssystem erstmals am Computer nachzubilden.
Um dieses Modell nun zu erweitern und zu verfeinern, entstand zusammen mit dieser Bachelorarbeit ein neues Ventilmodell. Während das oben genannte Ventilmodell den Massenstrom über die Ausflussfunktion ψ, somit also über die Druckdifferenz zwischen der Ein- und Ausflussseite des Ventils, berechnet, soll das neue Ventilmodell gleich dem Rohrleitungsmodell über die eindimensionalen Eulergleichungen diskretisiert und berechnet werden, sodass es sich ebenso in die verschiedenen bereits vorhandenen Diskretisierungsverfahren einfügt und damit auch in das Gitter des Rohrleitungsmodells. Da hier die Güte des Ventilmodells von entscheidender Bedeutung ist, beschränkt sich diese Bachelorarbeit nicht nur auf die Erstellung eines einzelnen Ventilmodells, sondern verfolgt auch den Ansatz eines Ventilmodells in einem CFD-Programm, welches dementsprechend auch unterschiedliche Ergebnisse liefert. Somit ist ein wichtiger Punkt dieser Arbeit die Bewertung und Abgrenzung der einzelnen Modelle mit besonderer Betrachtung auf die Anwendbarkeit des diskretisierten Ventilmodells auch in Hinblick darauf, dass im Realen ein Schmetterlingsventil verbaut wurde.
Der weitere Aufbau dieser Bachelorarbeit gliedert sich wie folgt: Zunächst werden einige Grundlagen beschrieben. Dazu gehört eine Einführung in Modelica sowie in die Eulergleichungen. Außerdem wird ausführlich auf die verschiedenen Riemannlöser bzw.

die unterschiedlichen Berechnungen der numerischen Flüsse eingegangen. Besonderes Augenmerk wird hierbei auf vier verschiedene Verfahren zur Berechnung des numerischen Fluss gelegt: Die Flussberechnung nach Roe, das Harten-Lax-van Leer (HLL)-Verfahren, das Verfahren nach Lax und Friedrichs, sowie das Monotonic Upstream-Centered Scheme for Conservation Laws (MUSCL)-Hancock-Verfahren. Ebenso folgt ein kurzer Abriss zum CFD-Programm OpenFOAM, mit dessen Hilfe das Ventilmodell außerhalb von Modelica entstanden ist. An den Grundlagenteil schließt sich die exakte Realisierung der verschiedenen Ventilmodelle mit der Hervorhebung ihrer einzelnen Besonderheiten an. Im Anschluss werden die Ergebnissen der unterschiedlichen Modelle dargestellt und diskutiert, was zusammen mit der Realisierung den Hauptteil dieser Arbeit darstellt.

1.2 Grundlagen

1.2.1 Einführung Modelica

Modelica ist eine objektorientierte, aber gleichzeitig auch gleichungsbasierte Modellierungssprache. Es gibt sie unter anderem in der OpenSource-Version OpenModelica. Diese Bachelorarbeit wurde mit der lizenzpflichtigen Version Dymola Modelica, welche von Dassault Systèmes entwickelt wurde, erstellt. Modelica findet heutzutage in der Technik verbreitet Anwendung und wird besonders in der Automobilbranche, wie auch in der Luft- und Raumfahrt benutzt. Der Einsatzzweck von Modelica ist in der Vorentwicklung und -auslegung von komplexen Systemen, welche mit Hilfe von Modelica am Computer modelliert und anschließend simuliert werden können. Grundlegendes Prinzip hierbei ist es, die physikalischen Gleichungen (so wie in diesem Fall die Eulergleichungen) dem Computer vorzugeben und anschließend den Zeitverlauf der einzelnen Größen berechnen zu lassen. Die besondere Stärke von Modelica liegt darin, dass es nicht nur algebraische Gleichungen, sondern auch gewöhnliche Differentialgleichungen mit der Variablen Zeit berechnen kann. Dabei können diese Gleichungen sowohl analytisch als auch numerisch gelöst werden. Hierfür sind bereits verschiedene numerische Löser implementiert.
Mit diesen Werkzeugen lassen sich komplexe physikalische Sachverhalte ohne größeren Aufwand modellieren und berechnen. So benötigt man zur Berechnung eines einfachen Pendels nur folgende zwei Gleichungen:

$$\dot{\varphi} = \omega \tag{1.1}$$

$$\dot{\omega} = -\frac{g \sin \varphi}{r} \tag{1.2}$$

Letztlich muss für φ nur noch ein Startwert vorgegeben werden und Modelica kann dann den Zeitverlauf des Pendels berechnen. Dieser kann dann in einem Diagramm zur Auswertung ausgegeben werden. Der Quellcode sieht wie folgt aus:

```
Real phi;  //angle
Real omega;  //angular acceleration
parameter Real g = 9.81;  //acceleration of gravity [m/s
   ^2]
parameter Real r = 1;  //length of pendlum [m]

equation
  der(phi) = omega;
  der(omega) = -(g*sin(phi))/r;
initial equation
  phi = Pi/2;
```

Listing 1.1: *Implementierung eines Fadenpendels in Modelica*

Wenn man dieses Modell für fünf Sekunden ausführen lässt, ergibt sich Diagramm (1.1). Zum weiteren Verständnis werden nun noch die in Modelica verwendeten Typen vorge-

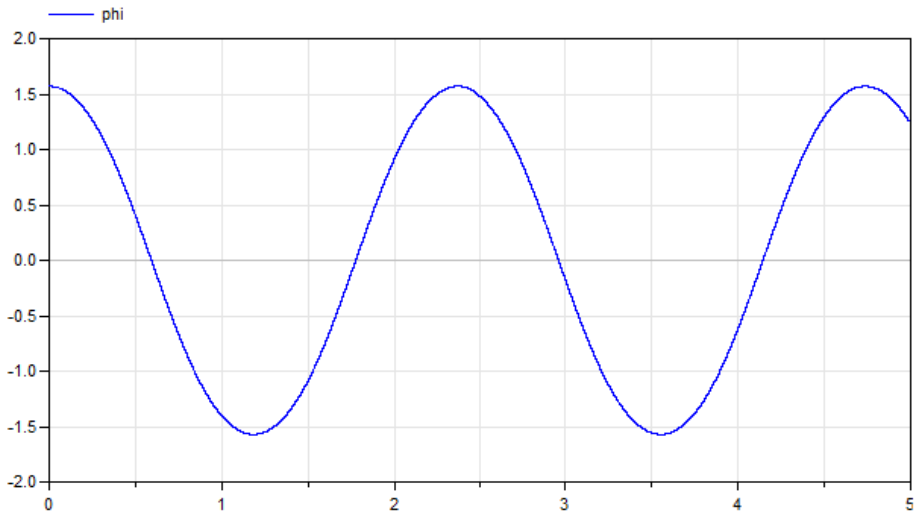

Abbildung 1.1: *Abhängigkeit des Winkels von der Zeit eines idealen Fadenpendels simuliert für 5s mit Modelica. Aufgrund der Startauslenkung von $\pi/2$ gilt hier die Kleinwinkelnäherung nicht.*

stellt:

Konnektoren Wie der Name „Konnektor“ schon andeutet, stellen diese Klassen die Verbindungselemente der einzelnen Untermodelle da. Sie bestehen lediglich aus Größen, die nur von einem zum anderen Modell weiter gegeben werden – Im allgemeinen Zustandsgrößen –, und so genannten Flüssen (wie Massenstrom, etc.). Zu beachten gilt,

dass der Fluss, der aus einem Modell heraus fließt, ein negatives Vorzeichen hat, und der, der hinein fließt, ein positives.

Funktionen Gleich den Funktionen anderer gängiger Programmiersprachen, werden Funktionen in Modelica mit einem oder mehreren Eingabewerten aufgerufen und geben einen Ausgabewert zurück. Auch die oft benötigten Ableitung von Funktionen kann man mit dem Operator der(f) darstellen. Bei geschlossenen Funktionstermen wird die Ableitung von Modelica berechnet. Ist dies nicht der Fall, zum Beispiel durch If-Verzweigungen, muss der Anwender die Ableitung explizit vorgeben.

Modelle Die Modelle sind die meist gebrauchten Klassen in Modelica, da diese die einzigen sind, die Modelica ausführen bzw. simulieren kann. Es kann, wie oben im Beispiel des Pendels, nur aus Gleichungen bestehen, oder aus diversen Konnektoren, Funktionen und weiteren Modellen aufgebaut werden. Ein komplexes System wird aus Untermodellen mittels Konnektoren zusammengesetzt. Das in dieser Arbeit betrachtete Modell wird zum Beispiel aus Rohren, Ventilen und Randbedingungen, welche alle einfache Modelle sind, zusammengesetzt.

Wie in objektorientierten Sprachen üblich, können Klassen von Basisklassen erben. Modelle, welche durch konnektierte Untermodelle aufgebaut werden, sind naturgemäß keine von besagten Untermodellen abgeleiteten Klassen.

1.2.2 Die Eulergleichungen

Grundlagen der Beschreibung der Fluiddynamik sind die Eulergleichungen. Im folgenden wird ein kleiner Umriss dieser Gleichungen dargelegt, um für die spätere numerische Berechnung in Abschnitt 2.3.2 die nötigen Grundlagen zu schaffen. Die hier aufgeführte Beschreibung wurde ausgearbeitet nach Krämer [7, Kap. 3] und [6, Kap. 8] sowie nach Toro[16]. Für ein tiefgründiges Studium der Fluiddynamik sei auf die Ausführungen nach Batchelor [1] verwiesen.

Das vollständige Gleichungssystem der Eulergleichungen beinhaltet drei Erhaltungsgleichungen: Die Massenerhaltung (Kontinuitätsgleichung), die Impulserhaltung in alle drei Raumrichtungen sowie die Energieerhaltung. Es entsteht so ein Gleichungssystem mit fünf Gleichungen und den fünf Unbekannten ρ, u, v, w, p. In ihrer konservativen Form

lauten die fünf Gleichungen wie folgt:

$$\rho_t + (\rho u)_x + (\rho v)_y + (\rho w)_z = 0 \qquad \text{(Massenerhaltung)} \tag{1.3}$$

$$(\rho u)_t + (\rho u^2 + p)_x + (\rho uv)_y + (\rho uw)_z = 0 \qquad \text{(Impulserhaltung in x)} \tag{1.4}$$

$$(\rho v)_t + (\rho uv)_x + (\rho v^2 + p)_y + (\rho vw)_z = 0 \qquad \text{(Impulserhaltung in y)} \tag{1.5}$$

$$(\rho w)_t + (\rho uw)_x + (\rho vw)_y + (\rho w^2 + p)_z = 0 \qquad \text{(Impulserhaltung in z)} \tag{1.6}$$

$$E_t + [u(E+p)]_x + [v(E+p)]_y + [w(E+p)]_z = 0 \qquad \text{(Energieerhaltung)} \tag{1.7}$$

E stellt dabei die totale Energie pro Volumen da:

$$E = \rho \left[\frac{1}{2}(u^2 + v^2 + w^2) + e \right] \tag{1.8}$$

Die innere Energie e eines idealen Gases ergibt sich zusammen mit der idealen Gasgleichung zu:

$$e = c_v T = c_v \frac{p}{\rho R} = e(\rho, p) \tag{1.9}$$

Diese Bachelorarbeit beschäftigt sich ausschließlich mit eindimensionalen Strömungsvorgängen, so dass die Gleichungen (1.5) und (1.6) nicht benötigt werden.
Die Eulergleichungen sind ein streng hyperbolisches Differenzialgleichungssystem erster Ordnung. Sie lässt sich damit in konservativer Form wie folgt darstellen:

$$U_t + F(U)_x = 0 \tag{1.10}$$

Für U bzw. F gilt dann:

$$U = \begin{pmatrix} \rho \\ \rho u \\ E \end{pmatrix}, \quad F = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ u(E+p) \end{pmatrix} \tag{1.11}$$

Gl. (1.10) gilt, soweit keine Quellterme zu berücksichtigen sind (rechte Seite = 0). In Kapitel 2.3.2 wird der dort benötigte Quellterm erläutert.

1.2.3 Riemannlöser nach Godunov

Da die Eulergleichungen zum Grundtyp der streng hyperbolischen Differentialgleichungssysteme gehören, müssen sie im Allgemeinen numerisch approximiert werden. Hierfür existieren die verschiedensten Ansätze. Im Rahmen dieser Arbeit finden ausschließlich Riemannlöser nach Godunov zur Berechnung der Eulergleichungen Anwendung.
Die Idee von Godunov war es, die Approximation als stückweise konstant anzunehmen (siehe Abb. (1.2), vgl. Munz [9, Kap. 5, S. 15]). Damit ergeben sich m Volumen (hier

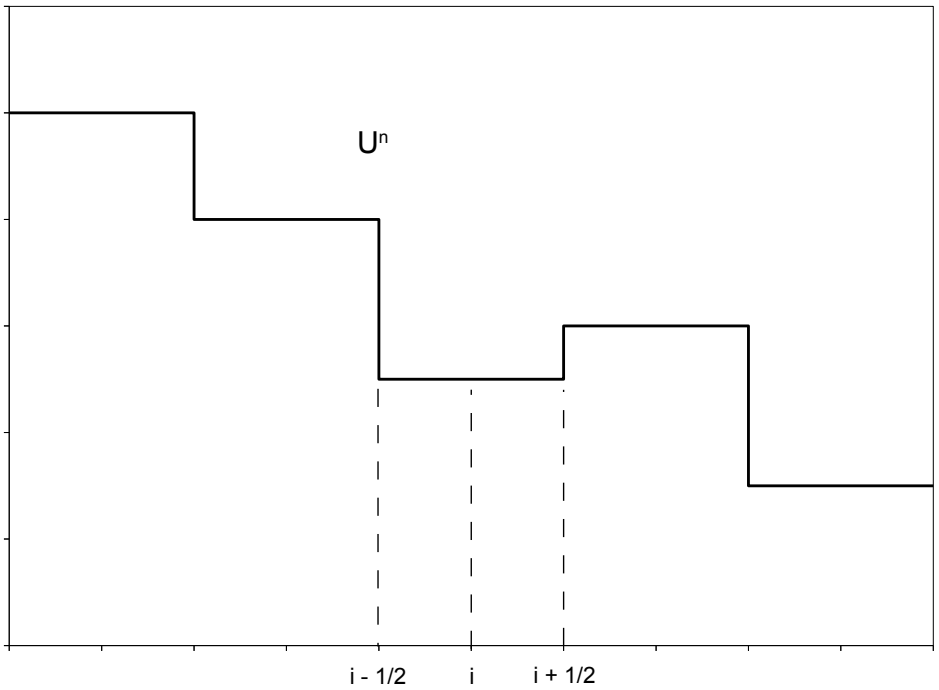

Abbildung 1.2: *Godunov's Idee zur Berechnung von hyperbolischen Differentialgleichungen ist es, die DGL in viele kleine lokale Riemannprobleme zu zerlegen. Damit kann die DGL einfacher approximiert werden.*

fünf konstante Abschnitte), die in ihrem inneren konstante Werte und an ihren Rändern Sprünge (Unstetigkeiten) vorweisen. Somit sind aus dem Problem des Lösens der DGL lokale Riemannprobleme entstanden, die einzeln wie folgt gelöst werden können:

$$U_i^{n+1} = U_i^n - \frac{\Delta t}{\Delta x}(g_{i+1/2}^n - g_{i-1/2}^n) \tag{1.12}$$

Es handelt sich somit um ein upwind-Verfahren erster Ordnung und ist damit stabil, wenn die Courant-Friedrichs-Lewy (CFL)-Bedingung

$$\frac{\Delta x}{\Delta t} \geq |a| \tag{1.13}$$

für a als Ausbreitungsgeschwindigkeit der Informationen erfüllt ist. Dies stellt sicher, dass keine Informationen in einem Zeitschritt an die Nachbarzelle weiter gegeben wird. Andernfalls wird die Lösung instabil. Im Anwendungsfall stellt U_i^n den Zustand im i-ten Volumen (Abschnitt) im n-ten Zeitschritt da und $g_{i+1/2}^n$ den numerischen Fluss über

die rechte Seite des Volumens, sowie $g^n_{i-1/2}$ den numerischen Fluss über die linke Seite im n-ten Zeitschritt da. Besondere Anwendung findet das Verfahren nach Godunov zur Berechnung der Vorgänge im Stoßrohr. Diese Vorgänge wurden intensiv 1978 in dem nach Gary A. Sod benannten Sod-Problem [15] untersucht. Es dient bis heute noch zur Verifikation von CFD-Verfahren, insbesondere auch den Riemannlösern.
Da Modelica eigene Module zur Berechnung der zeitliche Ableitungen enthält, ist es nicht notwendig, die zeitliche Ableitung des Zustandes der Volumina mit dem Differenzenquotient anzunähern:

$$U_t \approx \frac{U_i^{n+1} - U_i^n}{\Delta t} \tag{1.14}$$

Es reicht somit die semidiskrete Darstellung der hyperbolischen Differentialgleichung.

$$\partial U = -\frac{g^n_{i+1/2} - g^n_{i-1/2}}{\Delta x} \tag{1.15}$$

Damit ist für die Approximation der Eulergleichung in Modelica ausschließlich die Berechnung des numerischen Flusses notwendig.

1.2.4 Numerische Flussberechnung

Da es aus den oben genannten Gründen von besonderer Bedeutung ist, wie der numerische Fluss berechnet wird, soll in diesem Abschnitt ausführlich auf diese Berechnung eingegangen werden. Dafür wird die Idee der einzelnen Flüsse vorgestellt und die Vor- und Nachteile kurz betrachtet. Die Implementierung der Flüsse in Modelica ist ebenfalls Bestandteil der vorhandenen Gasdynamikbibliothek und wurde von Sielemann [13] erstellt. Die hier vorgestellten sowie weitere Verfahren werden von Toro [16] genau beschrieben.

Hyperbolische Differenzialgleichungssysteme lassen sich zusätzlich zur Gl. (1.10) auch wie folgt darstellen:

$$U_t + AU_x = 0 \tag{1.16}$$

A stellt hierbei die Jacobi-Matrix dar. Diese wird durch

$$A(U) = \partial F/\partial U = \begin{pmatrix} 0 & 1 & 0 \\ -\frac{1}{2}(\kappa-3)u^2 & (3-\kappa)u & \kappa - 1 \\ u\left[\frac{1}{2}(\kappa-1)u^2 - H\right] & H - (\kappa-1)u^2 & \kappa u \end{pmatrix} \tag{1.17}$$

berechnet. Hier steht κ für den Isentropenexponent und für die totale Enthalpie H gilt:

$$H = (E + p)/\rho \tag{1.18}$$

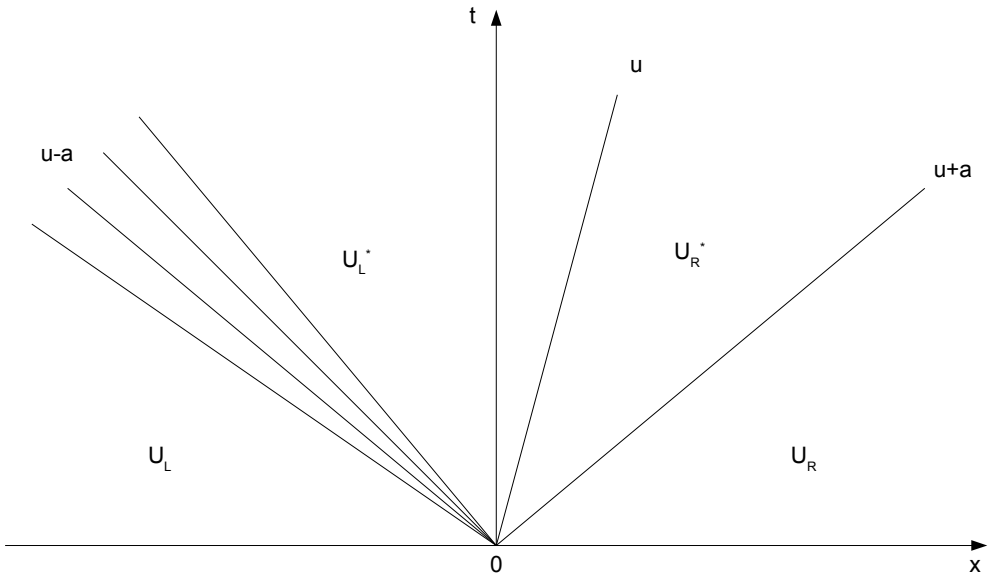

Abbildung 1.3: *Wellenausbreitung um einen Fixpunkt der Eulergleichungen. U_L und U_R stellen die Konstanten Werte links und rechts vom Fixpunkt da. Die Informationen breiten sich mit den Wellengeschwindigkeiten u-a und u+a aus. Diese Wellen können Stoßwellen (-fronten, hier rechts) oder Verdünnungswellen (-fächer, hier links) sein. Dadurch entstehen die Werte U_L^* und U_R^* die durch die dritte Welle, die sogenannte Kontaktunstetigkeit, getrennt werden. Diese breitet sich mit u aus. Die Wellengeschwindigkeiten entsprechen den Eigenwerten der Jacobi-Matrix*

Da die Jacobi-Matrix von U abhängig ist, ist das System der Eulergleichungen nicht linear. Von einen Fixpunkt x_0 breiten sich im Falle der Eulergleichungen drei Wellen aus (s. Abb. (1.3)). Die einzelnen Wellengeschwindigkeiten entsprechen den Eigenwerten der Jacobi-Matrix. Im Falle der Eulergleichungen ergeben sich diese zu:

$$\lambda_1 = u - a, \quad \lambda_2 = u, \quad \lambda_3 = u + a \tag{1.19}$$

und damit sind die Eigenvektoren:

$$K_1 = \begin{pmatrix} 1 \\ u - a \\ H - ua \end{pmatrix}, \quad K_2 = \begin{pmatrix} 1 \\ u \\ \frac{1}{2}u^2 \end{pmatrix}, \quad K_3 = \begin{pmatrix} 1 \\ u + a \\ H + ua \end{pmatrix} \tag{1.20}$$

Hier gilt jeweils für die Schallgeschwindigkeit $a = \sqrt{\frac{\kappa p}{\rho}}$. Zur Flussberechnung ist die Abbildung der einzelnen Wellen und die daraus resultierende Berechnung der Werte U_L^*

und U_R^* von entscheidender Bedeutung. Diese Approximation wird im folgenden für die verschiedenen Diskretisierungen, wie sie in der Gasdynamikbibliothek benutzt werden, dargelegt.

Riemannlöser nach Roe

Eine Berechnung von Riemannproblemen stellte Roe 1981 in [11] vor. Seitdem wurde dieser Riemannlöser zu einer der bekanntesten in der Numerik. Die zugrunde liegende Idee ist es, das exakte Riemannproblem durch ein neues Riemannproblem zu approximieren, welches eine Linearisierung des exakten Riemannproblems darstellt und wie folgt aussieht:

$$U_t + A_{lr} U_x = 0 \tag{1.21}$$

Die neue Jacobi-Matrix A_{lr} bildet sich aus den konstanten Werten U_L und U_R, wodurch das System linear wird. Durch diese Linearisierung kann das neu entstandene Riemannproblem einfach und dazu noch exakt berechnet werden. Damit die Approximation hinreichend genau wird, muss die Matrix $A_{lr}(U_L, U_R)$ folgende Kriterien erfüllen [11, 16]:

- Das neue System muss ebenfalls streng hyperbolisch sein und damit muss $A_{lr}(U_L, U_R)$ diagonalisierbar sein und alle Eigenvektoren müssen linear unabhängig sein.
- Konsistenz mit der ursprünglichen Jacobi-Matrix: $A_{lr}(U_l, U_r) = A(U)$
- $A_{lr}(U_L, U_R) \times (U_l - U_r) = F_R - F_L$

Es lässt sich nun zeigen, dass die Matrix $A_{lr} = A(\tilde{U})$ die drei oben genannten Bedingungen erfüllt. Die Werte für $\tilde{U}$ werden dabei wie folgt von Roe definiert:

$$\begin{aligned} \tilde{\rho} &= \sqrt{\rho_L \rho_R}, \quad \tilde{u} = \frac{\sqrt{\rho_L} u_L + \sqrt{\rho_R} u_R}{\sqrt{\rho_L} + \sqrt{\rho_R}} \\ \tilde{H} &= \frac{\sqrt{\rho_L} H_L + \sqrt{\rho_L} H_L}{\sqrt{\rho_L} + \sqrt{\rho_R}}, \quad \tilde{a} = (\kappa - 1)\sqrt{\tilde{H} - 1/2u^2} \end{aligned} \tag{1.22}$$

So ergeben sich die neuen Eigenwerte:

$$\tilde{\lambda_1} = \tilde{u} - \tilde{a}, \quad \tilde{\lambda_2} = \tilde{u}, \quad \tilde{\lambda_3} = \tilde{u} + \tilde{a} \tag{1.23}$$

mit dem entsprechenden Eigenvektoren wie in (1.20). Der Fluss, so wie Roe in beschreibt, sieht wie folgt aus:

$$g_{i+1/2} = \frac{1}{2}(F_L + F_R) - \frac{1}{2}\sum_{i=1}^{3} \tilde{\alpha_i} |\tilde{\lambda_i}| \tilde{K_i} \tag{1.24}$$

Es fehlt somit nur noch die lokale Wellenstärken $\tilde{\alpha_i}$, die sich berechnet durch:

$$\Delta U = U_L - U_R = \sum_{i=1}^{3} \tilde{\alpha_i} \tilde{K_i} \tag{1.25}$$

Es ergibt sich ausgeschrieben und nach $\tilde{\alpha_i}$ aufgelöst:

$$\begin{aligned} \tilde{\alpha_2} &= \frac{\kappa - 1}{\tilde{a}^2} \left[\Delta\rho(\tilde{H} - \tilde{u}^2) + \tilde{u}\Delta(\rho u) - \Delta E - \Delta\rho\tilde{u}^2 \right] \\ \tilde{\alpha_1} &= \frac{1}{2\tilde{a}} \left[\Delta\rho(\tilde{u} + \tilde{a}) - \Delta(\rho u) - \tilde{a}\tilde{\alpha_2} \right] \\ \tilde{\alpha_3} &= \Delta\rho - (\tilde{\alpha_1} + \tilde{\alpha_2}) \end{aligned} \tag{1.26}$$

Verfahren nach Harten, Lax und van Leer

Das Harten-Lax-van Leer-Verfahren [5] nimmt als Vereinfachung nicht nur für die Schockwelle, sondern auch für die Verdünnungswelle eine Front an. Dadurch ergeben sich an der so entstandenen Verdünnungsfront Unstimmigkeiten mit der Entropie, die aber im Weiteren vernachlässigt werden. Ebenso wird angenommen, dass es nur einen *-Wert U_{HLL} gibt. Die Kontaktunstetigkeit wird also vernachlässigt (vgl. Abb. (1.4)). U_{HLL} kann berechnet werden durch:

$$U_{HLL} = \frac{S_R * U_R - S_L * U_L + F_L - F_R}{S_R - S_L} \tag{1.27}$$

Hierbei stehen S_L und S_R für die Ausbreitungsgeschwindigkeit der Fronten nach links und nach rechts und werden wie folgt berechnet:

$$S_R = u_R + a_R, \quad S_L = u_L - a_L \tag{1.28}$$

a_L und a_R ist jeweils die lokale Schallgeschwindigkeit an der linken bzw. an der rechten Front. Davis und Einfeldt [2, 3] haben dazu noch vorgeschlagen, statt u und a die von Roe berechneten Mittelwerte $\tilde{u}$ und $\tilde{a}$ (s. Gl. (1.22)) zu verwenden. Hier hat sich herausgestellt, dass für eine verbesserte Robustheit der Maximalwert für die rechte und der Minimalwert für die linke Seite benutzt werden sollte.

$$S_R = \max(u_R + a_R, \tilde{u}_R + \tilde{a}_R), \quad S_L = \min(u_L - a_L, \tilde{u}_L + \tilde{a}_L) \tag{1.29}$$

Schlussendlich ergibt sich dann der Fluss nach Harten, Lax und van Leer zu:

$$g_{i+1/2} = \frac{S_R F_L - S_L F_R + S_L S_R (U_R - U_L)}{S_R - S_L} \tag{1.30}$$

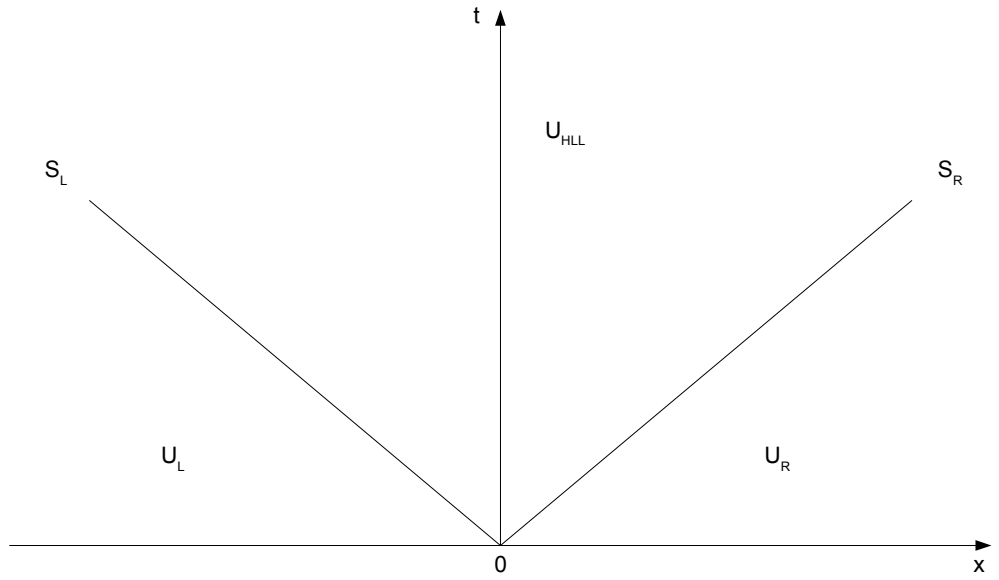

Abbildung 1.4: *Wellenausbreitung um einen Fixpunkt des HLL-Verfahren. Die Kontaktunstetigkeit wird in diesem Fall vernachlässigt. Damit ergibt sich nur noch ein *-Wert U_{HLL} Die Wellen breiten sich mit den Geschwindigkeiten S_L und S_R aus.*

Lokales Verfahren nach Lax und Friedrichs: Der monotone Fluss nach Rusanov

Die Berechnung des Flusses nach Lax und Friedrichs stellt das einfachste und damit das zeiteffizienteste Verfahren da. Dagegen ist es jedoch auch das ungenaueste. Für die speziellen Anwendungen muss damit Rechenzeit gegen Genauigkeit abgewogen werden. Da für diese Arbeit keine realen Daten vorliegen, somit also eine Bestimmung der Exaktheit nicht möglich ist, wurde sich für den weiteren Verlauf für dieses Verfahren aufgrund der hohen Berechnungsgeschwindigkeit entschieden.
Das Verfahren wurde bereits 1954 von Lax [8] vorgestellt und gehört damit zu den älteren Verfahren. Ursprünglich wurde es für die Berechnung von finiten Differenzen entwickelt und wurde erst später auf die Finite-Volumen-Verfahren angewendet. Gleich dem HLL-Verfahren wird nur eine Verdünnungsfront und eine Stoßfront angenommen und somit wird wieder die Kontaktunstetigkeit vernachlässigt. Des Weiteren wird hier im Gegensatz zum HLL-Verfahren für die Wellengeschwindigkeit angenommen, dass sie genau in einem Zeitschritt Δt die Zellgrenze erreicht (vgl. Abb. (1.5)). Damit gilt für ein gleichmäßiges Gitter, dass in jeder Zelle die gleiche Wellengeschwindigkeit vorliegt, was die gröbere Approximation gegenüber dem HLL-Verfahren darstellt.

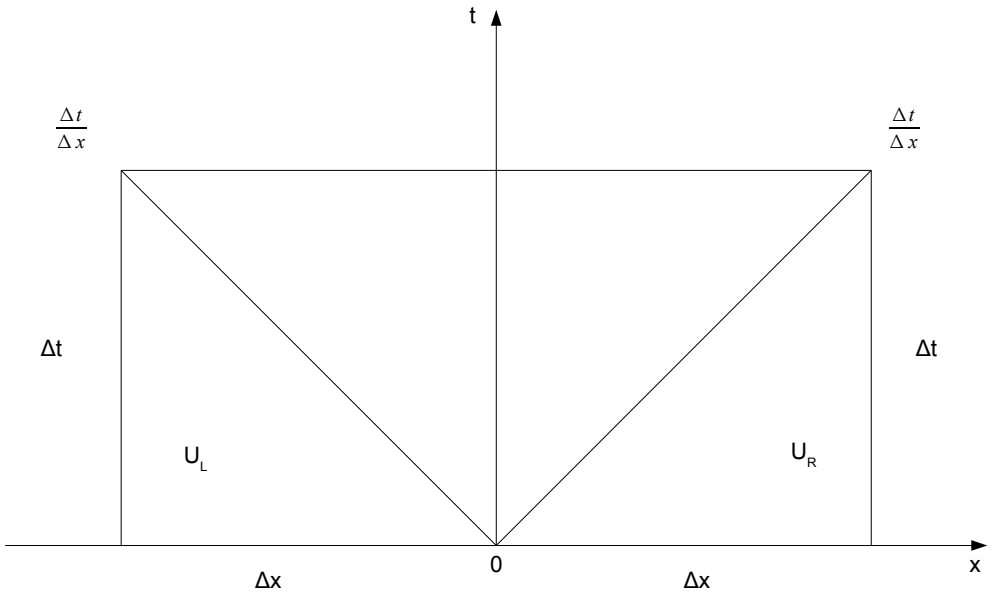

Abbildung 1.5: *Wellenausbreitung um einen Fixpunkt nach Lax und Friedrichs. Hier wird als starke Vereinfachung angenommen, dass die Wellen sich exakt um Δx nach dem Zeitschritt Δt ausgebreitet haben. Damit das Verfahren stabil ist, muss die maximal mögliche Wellengeschwindigkeit für das gesamte Gitter angenommen werden. Bei konstantem Zeitschritt ist das Gitter äquidistant.*

Der Fluss nach Lax und Friedrichs wird dann berechnet durch:

$$g_{i+1/2} = \frac{1}{2}(F(U_R) + F(U_L) - \frac{\Delta x}{2\Delta t}(U_R - U_L) \tag{1.31}$$

Da Modelica jedoch die zeitliche Ableitung übernimmt, liegt, wie oben beschrieben, die semidiskrete Darstellung und somit kein expliziter Zeitschritt Δt vor. Um dennoch das Verfahren anwenden zu können, wird die CFL-Bedingung

$$\frac{\Delta x}{\Delta t} > max(|u_R + c|, |u_L + c|) = \lambda_{max} \tag{1.32}$$

in Gleichung (1.31) eingesetzt. Dies stellt gerade eben noch sicher, dass sich keine Wellen innerhalb einer Gitterzelle schneiden und somit das Verfahren stabil ist. Es ergibt sich damit der von Rusanov [12] entwickelte Rusanovfluss:

$$g_{i+1/2} = \frac{1}{2}(F(U_R) + F(U_L) - \frac{\lambda_{max}}{2}(U_R - U_L) \tag{1.33}$$

Die MUSCL-Verfahren

Die MUSCL-Verfahren sind Verfahren, welche den Raum nicht in erster sonder in höherer Ordnung annähern. Hierbei werden die als konstant angenommenen Zellwerte zunächst rekonstruiert, bevor dann mit einem der genannten Verfahren und den neuen Zellwerten der numerische Fluss berechnet wird. In diesem Fall werden die Werte linear, also in zweiter Ordnung im Raum rekonstruiert. Für die lineare Rekonstruktion sind nicht nur die benachbarten Zellwert von Nöten, sondern es werden auf beiden Seiten je zwei Werte gebraucht. Besonders Stoßprobleme machen es nötig, Limiter-Funktionen zu verwenden, denn ansonsten führt die Rekonstruktion häufig zu Oszillationen. Aus diesem Grund soll die Limiter-Funktion dafür sorgen, dass weder neue Maxima noch neue Minima entstehen (vgl. Abb. (1.6)). Dadurch, dass keine neuen Extremstellen entstehen, die totale Variation

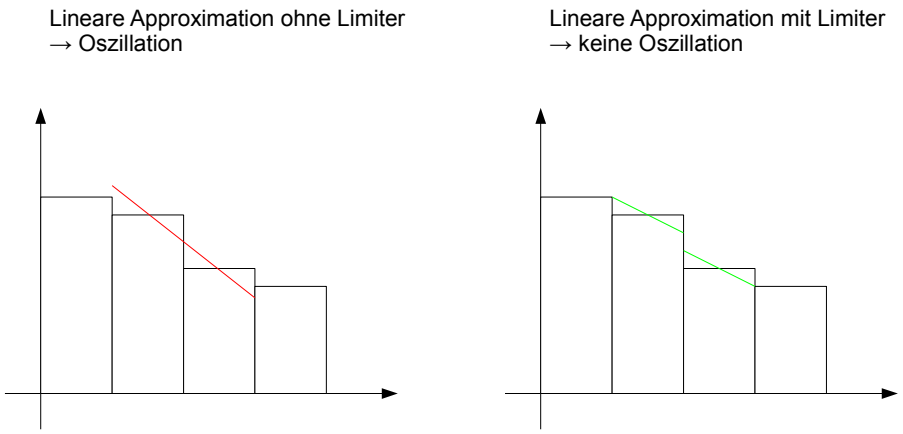

Abbildung 1.6: *Bei der Rekonstruktion der Zellwerte für die Berechnung des nächsten Zeitschritts muss darauf geachtet werden, dass keine neuen Extremstellen entstehen, da Oszillationen die Rechnung stark in die Länge ziehen. Die Rekonstruktion erfolgt mit der Steigung*

also nicht erhöht wird, ist diese Verfahren auch ein Total-Variation-Diminishing (TVD)-Verfahren. Die hier verwendete Rekonstruktion heißt MUSCL-Hancock-Verfahren nach van Leer [17]:

$$u_i^R = u_i + \frac{1}{2}\Delta_i, \quad u_{i+1}^L = u_{i+1} - \frac{1}{2}\Delta_{i+1} \tag{1.34}$$

Für die Berechnung der Δ werden zunächst die Differenzen zwischen den Zellen berechnet:

$$\begin{aligned} \Delta_{i-1/2} &= u_i - u_{i-1} \\ \Delta_{i+1/2} &= u_{i+1} - u_i \\ \Delta_{i+3/2} &= u_{i+2} - u_{i+1} \end{aligned} \tag{1.35}$$

Mit Hilfe dieser Werte werden nun die benötigten Δ über eine Limiter-Funktion ermittelt:

$$\Delta_i = \begin{cases} \max[0, \min(\beta\Delta_{i-1/2}, \Delta_{i+1/2}), \min(\Delta_{i-1/2}, \beta\Delta_{i+1/2})], & \Delta_{i+1/2} > 0 \\ \min[0, \max(\beta\Delta_{i-1/2}, \Delta_{i+1/2}), \max(\Delta_{i-1/2}, \beta\Delta_{i+1/2})], & \Delta_{i+1/2} < 0 \end{cases} \tag{1.36}$$

β darf alle Werte zischen 1 und 2 annehmen. Für $\beta = 1$ stellt dies den Minbee (oder Minmod) Flux Limiter dar, für $\beta = 2$ den Superbee Flux Limiter. Nun können mit Hilfe der Gleichung (1.34) die Zellwerte rekonstruiert werden und der Fluss mit einem beliebigen Verfahren ermittelt werden (hier werden nur der Rusanov-Fluss und der Fluss nach Roe verwendet).

1.2.5 Gittererstellung für CFD-Programme

Für die Berechnung von Strömungen durch CFD-Programme benötigt man zunächst ein Gitter, welches die zu berechnende Strömung begrenzt. Die Güte des Gitters ist von besonderer Wichtigkeit, um genaue aber dennoch zeiteffiziente Lösungen zu erhalten. Aus diesem Grund wurde bereits viel an Strategien zur Gitteroptimierung geforscht, unter anderem auch mit evolutionären Algorithmen in Hanff [4]. Die Gitteroptimierung soll jedoch nicht Gegenstand diese Abschnitts sein, sondern es soll lediglich auf fundamentale Dinge zur Erstellung eines akzeptablen Gitters eingegangen werden.

Grundsätzlich lassen sich zwei Arten von Gittern unterscheiden: Die strukturierten und die unstrukturierten Gitter. Die zu erst genannten Gitter zeichnen sich dadurch aus, dass die Gitterzellen eindeutig indizierbar sind (beispielsweise durch kartesische oder Kugelkoordinaten). Dazu hat ein Knotenpunkt immer gleich viele Nachbarknoten. Ein strukturiertes Gitter setzt nicht voraus, dass alle Gitterzellen identische Ausmaße haben. Gitter mit identischen Ausmaß der Zellen sind eine Unterart der strukturierten Gitter und nennen sich gleichmäßige Gitter. Auf der anderen Seite können die unstrukturierten Gitter beliebig aussehen. Dies hat den Vorteil, dass sie sich so besser an die Topologie anpassen können. Dies bringt jedoch den Nachteil mit sich, dass keine eindeutige Indizierung mehr möglich ist. Dadurch erhöht sich der Speicherbedarf deutlich, da für jede Gitterzelle die Nachbargitterzellen gespeichert werden müssen. Oft entstehen solche unstrukturierten Gitter durch Optimierungen, sodass der erhöhte Speicherbedarf gegen die Rechengenauigkeit abgewogen werden muss.
Für die Güte von Gittern haben sich aus Erfahrungen einige Merkmale herauskristallisiert. Zunächst gilt für alle Gitter, dass je höher die Divergenz ist, desto feinmaschiger das Gitter für eine gute Approximation sein muss. Im Fall der Fluidströmung bedeutet dies, dass Orte, an denen beispielsweise Stöße auftreten, wie Vorderkanten von Überschalltragflächen, höher aufgelöst werden müssen, als Orte der Freiströmung. Ebenso sollten Nachbarzellen einen möglichst kleinen Größenunterschied vorweisen, sodass ein Übergang zu einem feineren Gitter fließend sein sollte. Auch sollten die einzelnen Gitterzellen möglichst gleichförmig/ symmetrisch sein. So ist also ein gleichseitiges Dreieck besser geeignet, als

ein Dreieck mit zwei langen Seiten und einer kurzen.
Für diese Arbeit wird ein Gitter für ein zylindrisches Rohr benötigt. Die Diskretisierung der Länge des Rohres ist besonders einfach, da dass Rohr von der Seite betrachtet ein Rechteck ist. Somit liegt hier eine gleichmäßige Diskretisierung in Rohrscheiben nahe. Die Diskretisierung der Stirnfläche stellt ein größeres Problem dar, wenn die oben genannte Gütemerkmale berücksichtigt werden sollen. Zunächst lässt sich vorstellen, den Kreis durch horizontale und vertikale Linien einzuteilen (Abb. (1.7) links). Hier wird das Zentrum sehr gut durch ein gleichmäßiges Gitter diskretisiert, am Rand ist es jedoch ungeeignet, insbesondere an den Stellen, wo statt viereckige dreieckige Gitterzellen entstehen, die dadurch sprunghaft kleiner werden. Damit ist dies Gitter insgesamt ungeeignet. Zweitens lässt sich vorstellen, den Kreis durch Speichen und kleiner werdende Kreise zu diskretisieren (Abb. (1.7) rechts). Hier entsteht ein gutes Gitter am Rand, zum Zentrum hin werden jedoch die Zellen schnell kleiner und auch hier entsteht ein Sprung von vier Ecken zu drei. Es ist somit ebenfalls ungeeignet.
Es lässt sich nun jedoch ein Gitter erstellen welches beide Vorteile vereinigt (Abb. (1.8)),

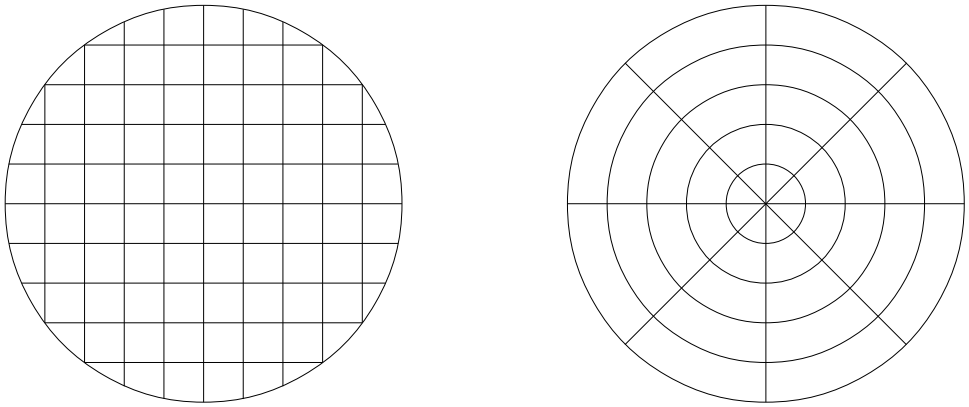

Abbildung 1.7: *Hier sind besonders einfache Diskretisierungen von Kreisen zu sehen. Beide sind jedoch unbrauchbar. Die linke am Rand und die rechte in der Mitte, da hier jeweils dreieckige Gitterzellen gebildet werden.*

ein sogenanntes O-Grid: Im Zentrum wird das gleichmäßige Gitter aus vertikalen und horizontalen Linien verwendet, am Rand das Gitter aus Kreisen und Speichen. Hier entsteht nur am Übergang eine Unregelmäßigkeit, die jedoch gegenüber den beiden Grundgittern gering ist. Dieses Gitter soll zusammen mit der bereits erwähnten Einteilung in Rohrscheiben an dieser Stelle ausreichen, um das Rohr zu diskretisieren.

1.2.6 OpenFOAM

Zur Validierung der Ventilmodelle wurde zusätzlich das CFD-Programm OpenFOAM benutzt. OpenFOAM ist ein auf dem Betriebssystem Linux laufendes Open-Source Pro-

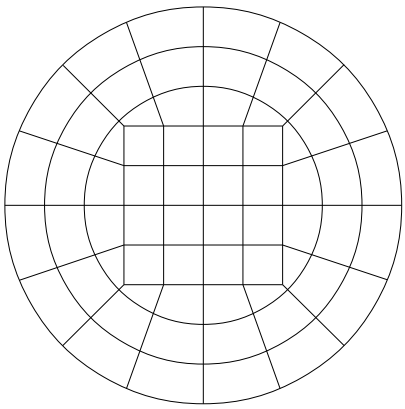

Abbildung 1.8: *Das O-Grid kombiniert die beiden oben genannten Gitter, wodurch keine dreieckigen Gitterzellen entstehen. Es ist in der Praxis ein gängiges Gitter zur Diskretisierung von Kreisen*

gramm, welches ohne graphische Benutzeroberfläche arbeitet. Dies macht den Umgang mit OpenFOAM zunächst nicht trivial, jedoch lassen sich nach einiger Einarbeitungszeit gute Simulationen erstellen.
Der Simulationsordner einer OpenFOAM Simulation gliedert sich in drei Unterordner: 0, constant und system. Im Ordner 0 werden alle benötigten Randbedingungen und Initialbedingungen erstellt. Wie dies geschieht wird in Abschnitt 2.4.2 beschrieben. Im Ordner system werden alle Einstellungen zum Algorithmus gespeichert. Dieser Ordner enthält drei Dateien: controlDict, fvSchemes und fvSolution. Der wichtigste Teil der Datei controlDict, sieht in diesem Fall so aus:

```
 FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      controlDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
application     sonicFoam;
startFrom       startTime;
startTime       0;
stopAt          endTime;
endTime         1;
```

```
deltaT          1e-05;
writeControl    runTime;
writeInterval   0.001;
```

Listing 1.2: *controlDict-Datei für die Simulation in OpenFOAM*

Im folgenden sollen die hier vorgenommenen Einstellungen erläutert werden. Zunächst muss der Anwender mit „application“ den gewünschten Lösungsalgorithmus auswählen. Hier wurde der „sonicFoam“ gewählt, da dieser für transsonische und supersonische Strömungen ausgelegt wurde. Insbesondere bei kritischem Druckverhältnis hat die Strömung Schallgeschwindigkeit im engsten Querschnitt, aber auch bei voll geöffneten Ventil kann es bei hohen Druckgradienten zu Überschallströmungen kommen. Als nächstes muss die Startzeit festgelegt werden. Damit wird dem Algorithmus vorgegeben, auf welchen Ordner er für die Initialbedinugngen zugreifen muss, hier also der Ordner 0. Sollte der Algorithmus aus irgendeinem Grund abbrechen, kann man hier auch andere Werte angeben, sodass die Rechnung beim letzten ausgeschriebenen Wert neu beginnen kann. Das Intervall, in dem die Werte ausgeschriebenen werden sollen, wird mit writeInterval festgelegt. Dieser Wert kann entweder in Zeitschritten erfolgen (writeControl runTime: hier 0.001s) oder nach einer gewissen Anzahl an Rechnungen erfolgen (writeControl timeStep). endTime ist selbsterklärend: Hier wurde eine Sekunde gewählt, da sich nach dieser Zeit nahezu quasistationäre Strömung ausgebildet hat. Der letzte wichtige Parameter ist deltaT. Hiermit wird der Rechenzeitschritt festgelegt. Wählt man den Zeitschritt zu groß, kann es passieren, dass der Algorithmus zu keiner Lösung kommt, folglich nicht konvergiert (vgl CFL-Bedingung), wählt man ihn dagegen zu klein, dauert die Rechnung unnötig lange.
In den anderen beiden Dateien wird der Löser genauer spezifiziert. Hier werden beispielsweise die Art des Löseres und die Toleranzen festgelegt. Es wurden die Einstellungen aus den Tutorials zu den einzelnen Algorithmen übernommen und sollen deshalb nicht weiter erläutert werden.
Im Ordner constant wird alles gespeichert, was über den gesamten Rechnungsverlauf konstant bleibt. Dazu gehört in erster Linie das Gitter, welches sich im Unterordner polyMesh befindet. Ebenso werden in diesem Ordner die thermodynamischen Eigenschaften des Stoffes gespeichert. Schließlich wird auch hier das Turbulenzmodell für die Berechnung vorgegeben. In allen folgenden Berechnungen wird das k-ϵ-Turbulenzmodell nach Launder und Sharma verwendet. Die k-ϵ-Turbulenzmodelle gehören zu den einfachsten Turbulenzmodellen und basiert auf zwei partiellen Differentialgleichungen zur Berechnung der turbulenten kinetischen Energie k und der isotropen Dissipationsrate ϵ. Das von Launder und Sharma wird auch als das Standard-k-ϵ-Turbulenzmodell genannt, worin sich auch die Verwendung dieses in dieser Arbeit begründet.
Der Inhalt aller Dateien ist im Anhang (A.1) - (A.8) nachzulesen.

2 Modellbildung der Ventile

Dieses Kapitel beschreibt die tatsächlichen Realisierung der Ventile sowohl in Modelica als auch in OpenFOAM. Dabei wird auf die zu Grunde liegende Idee und die mathematisch/ physikalische Umsetzung eingegangen. Dafür wird hier zunächst der Versuchsaufbau zum Vergleich der einzelnen Simulationen kurz betrachtet.

2.1 Versuchsaufbau in Modelica

Die Simulationen in Modelica und, wie später auch in Abschnitt (2.4.2) beschrieben wird, in OpenFOAM werden mit den gleichen Randbedingungen initialisiert, damit die Ergebnisse der Simulationen im nächsten Kapitel verglichen werden können. Der Prinzipielle Versuchsaufbau ist in Abb. (2.1) für die Gasdynamikbibliothek zu sehen, welche sich nur in der graphischen Darstellung von der Standardbibliothek unterscheidet. Ganz links und ganz rechts werden die Randbedingungen vorgegeben. Für den Großteil

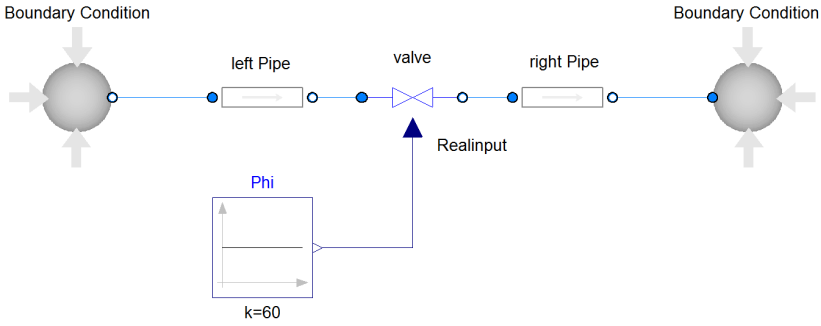

Abbildung 2.1: *Versuchsaufbau für die Betrachtung der Ventile in Modelica*

der Tests wurde ein Druck links von 1.1bar und rechts von 1bar festgelegt. Später wurden diese auch an den Arbeitsbereich angepasst. Die Temperaturen auf beiden Seiten sind mit 300K fest vorgegeben. Geschwindigkeiten werden hier nicht explizit vorgegeben, damit das Fluid ungehindert ein- bzw. ausströmen kann. Die Ausströmgeschwindigkeit aus den Randbedingungen wird gleich der Strömungsgeschwindigkeit am Anfang bzw. am Ende des Rohres gesetzt. Die Rohre haben jeweils ein Durchmesser von 0,1m und eine Länge

von 1m. Sie werden mit 1bar, 300K und mit verschwindender Strömungsgeschwindigkeit initialisiert. Beide Rohre werden für die Berechnung in zehn Volumen zerlegt. Da die Werte innerhalb der Rohre von geringer Bedeutung für diese Arbeit sind, reicht hier dieses grobe Netz. Dazu gewährleistet dies eine hohe Rechengeschwindigkeit (vgl. CFL-Bedingung). In der Mitte ist das Ventil zu sehen, welches in den folgenden Unterabschnitten erläutert wird. Dem Ventil wird von außen ein Wert (Realinput) für den Grad der Öffnung vorgegeben. Als Medium wird in allen Simulationen ein detailliertes Modell trockener Luft als ideales Gas verwendet. Dieses stellt alle thermodynamischen Eigenschaften des Mediums bereit, welche aus lediglich zwei Zustandsgrößen ermittelt werden, meist Druck und Temperatur.

2.2 Das Ventil aus der Standardbibliothek

Das Ventil aus der Standardbibliothek von Modelica wird über einen Durchflussfaktor ausgelegt. Dieser kann entweder der Av-(metrischer), Kv-(europäischer) oder der Cv(US-amerikanischer)-Wert sein, die alle ineinander umgerechnet werden können.

$$Av = 27.7 \cdot 10^{-6} * Kv \tag{2.1}$$

$$Av = 24 \cdot 10^{-6} \cdot Cv \tag{2.2}$$

$$Kv = 0.86 \cdot Cv \tag{2.3}$$

Die Werte für Av werden in m^2 die für Kv in m^3/h und für Cv in USG/min angegeben, obwohl dies nicht den physikalischen Einheiten entspricht (Einheitennachweis stimmt nicht!). Definiert wird der Kv-Wert als den Wasserdurchfluss durch ein Ventil bei einer Druckdifferenz von 1bar und einer Wassertemperatur von $5°C - 30°C$. Damit ergibt sich für den Volumenstrom:

$$Q = Kv \cdot \sqrt{\frac{\Delta p}{1bar} * \frac{1000\frac{kg}{s}}{\rho}} \tag{2.4}$$

Für den Cv-Wert gilt die gleiche Formel, allerdings wird hier die Druckdifferenz in Psi statt in bar angegeben und der berechnete Volumenstrom in USG/min statt m^3/h. Da der Av-Wert mit SI-Einheiten den Volumenstrom berechnet, muss dieser nicht umgerechnet werden. Damit sieht die Formel für den Av-Wert wie folgt aus:

$$Q = Av \cdot \sqrt{\frac{\Delta p}{\rho}} \tag{2.5}$$

Der Av-Wert lässt sich damit als effektiver Querschnitt betrachten. Er ist in etwa 1,4 mal so groß wie der tatsächliche Querschnitt des Ventilhalses. Damit ergibt sich zur Berechnung des Volumenstroms einen Av-Wert für ein Rohr mit 0,1m Durchmesser:

$$Av = 1,4A = 1,4 \cdot \frac{\pi}{4} d^2 = 0,011 \tag{2.6}$$

Aus dem Volumenstrom ergibt sich nun der Massenstrom durch Multiplikation mit der Dichte. Der Massenstrom wird dann durch die Konnektoren an die Rohre weitergegeben. Durch den Realinput wird der Wert der Öffnung des Ventils vorgegeben. Dieser Wert liegt zwischen 0 (vollständig geschlossenes Ventil) und 1 (vollständig geöffnetes Ventil). Es lässt sich Auswählen, wie dieser Wert den Massenstrom bestimmen soll. Hier wurde ein linearer Zusammenhang gewählt. Eine umfassende Beschreibung der Funktionsweise der Ventile ist in der Modelica Dokumentation [10] gegeben.

2.3 Zwei Ventile in der Gasdynamikbibliothek

Um die Umsetzung der Ventile besser zu verstehen, wird zunächst auf die von Sielemann eingeführten Konnektoren eingegangen. Die Konnektoren bestehen ähnlich der Beschreibungen in Abschnitt 1.2.1 aus Werten, die übergeben werden, jedoch aus keinen Flüssen. Dazu werden die Variablen nicht wie im allgemeinen Fall in beide Richtungen übergeben, sondern jeweils nur in eine. Es entstehen Input- und Outputwerte. Die Inputwerte eines Konnektors entsprechen dann den Outputwerten des mit ihm verbundenen Konnektors und umgekehrt. Übergeben werden jeweils Finite Volumen, somit also der thermodynamische Zustand, die Geschwindigkeit des Fluids und die Länge des entsprechenden Volumens. Die Inputs und Outputs werden hierbei in einem dynamischen Array gespeichert, welches je nach ausgewählter Diskretisierung unterschiedlich lang ist. Die Länge des Array entspricht dabei stets der Hälfte an Volumen, die zur Berechnung des Flusses notwendig sind, z.B. für den Rusanovfluss, welcher lediglich die Zustände aus dem Volumen links wie rechts benötigt, hat das Array die Länge eins. Dies ist mit einigem Programmieraufwand verbunden, ist aber von Nöten, da auf diese Weise die Flüsse direkt am Eingang und am Ausgang der einzelnen Komponenten berechnet werden können. Die Abbildung (2.2) veranschaulicht graphisch dieses Funktionsprinzip.
Im Realen ist, wie eingangs erwähnt, ein Schmetterlingsventil verbaut. Als Vereinfachung wird in den folgenden Modellen für den Querschnitt des Ventilhalses mit der projizierten Fläche der Verschlussklappe gerechnet. Die projizierte Fläche ist eine Ellipse. Damit ergibt sich der Öffnungsquerschnitt des Ventilhalses in beiden Modellen zu:

$$A_V = \frac{\pi}{4}D^2 - \frac{\pi}{4}D \cdot d = \frac{\pi}{4}D^2(1 - cos(\varphi)) \tag{2.7}$$

2.3.1 Ventilberechnung über Ausflussfunktion

Das erste erstellte Ventilmodell kommt noch, abgesehen von den Konnektoren, vollständig ohne Diskretisierungen aus. Die Idee für dieses Modell ist es, aus der Druckdifferenz den Massenstrom zu berechnen, welcher auf beiden Seiten des Ventils gleich sein muss (Kontinuitätsgleichung). Exakt dieser Ansatz wurde ebenfalls für das einfache bereits verwendete Ventilmodel verwendet und liefert damit auch exakt die gleichen Ergebnisse.

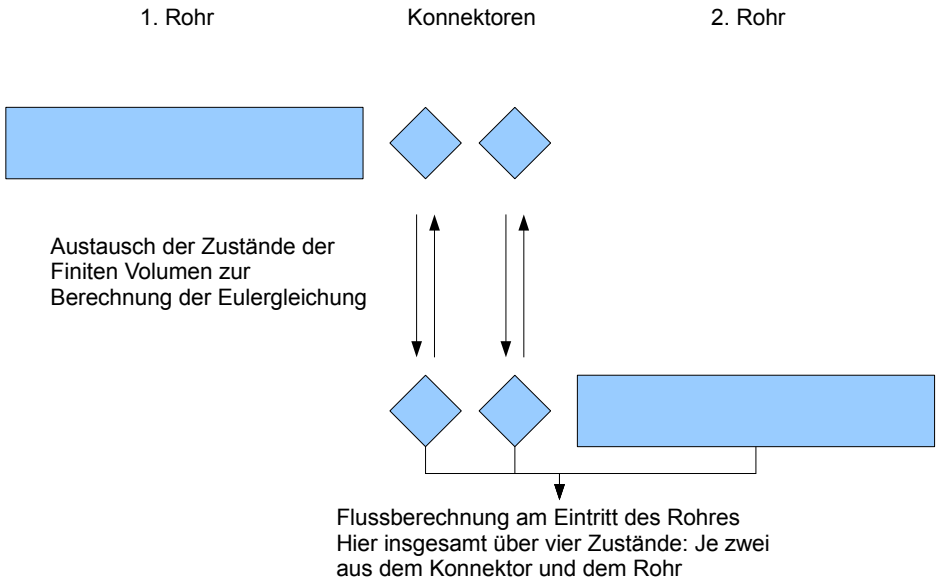

Abbildung 2.2: *Um die Flüsse direkt am Eingang bzw. am Ausgang eines Rohres berechnen zu können, sind die Zustände aus dem vorherigen bzw. dem nachfolgenden Rohr von Nöten. Diese werden durch die Konnektoren ausgetauscht. Übergeben werden jeweils die äußersten Zustände. Für das 1.Rohr ist der Output demnach die letzten zwei Zustände des selbigen Rohres und der Input die ersten zwei Zustände des 2.Rohres. Ähnliches gilt für das 2.Rohr.*

Für die Berechnung des Massenstroms wird zunächst die Ausflussfunktion Ψ eingeführt:

$$\Psi(x,\kappa) = \begin{cases} \sqrt{\frac{\kappa}{\kappa-1} x^{1/\kappa}(x^{1/\kappa}-x)}, & \left(\frac{2}{\kappa+1}\right)^{\frac{\kappa}{\kappa-1}} < x < 1 \\ \left(\frac{2}{\kappa+1}\right)^{\frac{1}{\kappa-1}} \sqrt{\frac{\kappa}{\kappa+1}}, & x \leq \left(\frac{2}{\kappa+1}\right)^{\frac{\kappa}{\kappa-1}} \end{cases} \tag{2.8}$$

x steht hier für den Druckquotienten p_2/p_1 und κ für den Isentropenexponent. Somit entspricht $\left(\frac{2}{\kappa+1}\right)^{\frac{\kappa}{\kappa-1}}$ genau dem kritischen Druckverhältnis. Damit ist die obere Gleichung für die Berechnung des Massenstroms von unterkritischen Strömungen zu gebrauchen und ist damit abhängig von der Druckdifferenz. Die untere Gleichung ist zur Berechnung von

kritischen bis überkritischen Strömungen da und damit unabhängig von der Druckdifferenz. Mit Hilfe dieser Ausflussfunktion lässt sich nun der Massenstrom bestimmen zu:

$$\dot{m} = A\Psi(p_{Out}/p_{In}, \kappa)\sqrt{2p_{in}\rho_{Out}} \tag{2.9}$$

Reibungs- und andere Verluste lassen sich mit einer Ausflusszahl $\mu \leq 1$ berücksichtigen, diese sollen aber hier vernachlässigt werden. Um zu gewährleisten, dass sich das Fluid in beide Richtungen bewegen kann, muss darauf geachtet werden, dass stets der Ort des größeren Drucks den Index 'In' hat und der des niedrigeren Drucks den Index 'Out' (bei gleichem Druck darf es natürlich zu keinem Massenstrom kommen). Ebenso muss sich das Vorzeichen umkehren, wenn es zu einer Richtungsänderung des Massenstroms kommt. Zusammen mit Gleichung (2.9) ergeben sich nun insgesamt drei Gleichungen für den Massenstrom, aus denen die Fließgeschwindigkeiten ermittelt werden können:

$$\dot{m} = A_V\Psi(p_{Out}/p_{In}, \kappa)\sqrt{2p_{In}\rho_{In}} \tag{2.10}$$

$$\dot{m} = A\rho_{In}v_{In} \tag{2.11}$$

$$\dot{m} = A\rho_{Out}v_{Out} \tag{2.12}$$

A_V entspricht hier der Querschnittsfläche des Ventils, während A die Querschnittsfläche der Rohre repräsentiert. Nun fehlt nur noch für die vollständige Beschreibung des Ventils die Zustandsvariablen des Fluid auf beiden Seiten des Ventils. Die Drücke p_1 und p_2 sollen hierfür direkt von den Rohren übernommen werden. Ebenso soll die Temperatur T_1 auf diese Art übernommen werden. In Modelica sieht das dann wie folgt aus:

```
mediumA.p = stencil_a.state_b[
   Discretization.halfStencilLength].p;
mediumA.T = stencil_a.state_b[
   Discretization.halfStencilLength].T;
mediumB.p = stencil_b.state_a[
   Discretization.halfStencilLength].p;
```

Listing 2.1: *Ventilmodell mit Ausflussfunktion: Berechnung des Massenstroms*

stencil_a bzw. stencil_b sind hier die jeweiligen Konnektoren. Die Verknüpfung a-b bzw. b-a stellen die Inputwerte da, wohingegen die Verknüpfung a-a bzw. b-b die Outputwerte darstellen würden. Da für die Beschreibung eines idealen einphasigen Gases immer zwei Werte von Nöten sind, ist mediumA bereits vollständig beschrieben. Für den zweiten Wert für mediumB wird die bei der adiabaten Drosselung konstant bleibende Enthalpie angesetzt:

```
mediumA.h = mediumB.h;
```

Listing 2.2: *Ventilmodell mit Ausflussfunktion: Die Enthalpie bleibt bei der adibaten Drosselung konstant*

Schlussendlich müssen nur noch die jeweiligen thermodynamischen Zustände und die Fließgeschwindigkeiten an die Outputvariablen der Konnektoren übergeben werden und das Ventilmodell ist damit fertig. Der vollständige Quellcode kann im Anhang (A.10) nachgelesen werden.

2.3.2 Das diskretisierte Ventil

Die zentrale Gleichung des Ventils, welches über die Diskretisierung realisiert wurde, ist die semidiskrete Darstellung der Eulergleichung, wie sie in ähnlicher Form in Gleichung (1.15) bereits vorgestellt wurde und an dieser Stelle nochmal in Modelica Semantik aufgegriffen werden soll:

```
der(U[i,:]) = (-flux[i,:] + sourceTerm[i,:])/((A[i] + A[i
   +1])/2*dx);
```

Listing 2.3: *Semidiskrete Darstellung der Eulergleichung*

Die linke Seite beschreibt die zeitliche Ableitung der konservativen Variablen der Eulergleichungen. Da es sich bei U um die inneren Zustände des Finiten Volumens handelt, werden diese dem Mediendaten entsprechend Gleichung (1.11) wie folgt zugewiesen:

```
U[i,1] = medium[i].d;
U[i,2] = medium[i].d*v[i];
U[i,3] = medium[i].d*(0.5*v[i]*v[i] + medium[i].u);
```

Listing 2.4: *Zuweisung der konservativen Variablen für die einzelnen Finite Volumen*

Die rechte Seite besteht aus dem intrazellulärem Fluss und einem Quellterm, der bereits in Abschnitt 1.2.2 angemerkt wurde. Da in diesem Fall nicht, wie für die Eulergleichungen vorgesehen, der flächenbezogene Fluss (Massenstrom pro Fläche, Impulsstrom pro Fläche und Energiestrom pro Fläche), sondern der absolute Fluss berechnet wurde, muss nicht nur wie in Gleichung (1.15) durch dx geteilt werden, sondern durch das gesamte Finite Volumen (gleiches gilt für den Quellterm). Die Größe des Volumens ist demnach, linear angenähert durch:

$$V = \frac{A_i + A_{i+1}}{2} dx \tag{2.13}$$

Im einfachsten Fall, zum Beispiel für den Rusanovfluss, wird das Ventil in zwei Volumina unterteilt. Die intrazellulären Flüsse werden aus der Differenz der Flüsse zwischen den Zellen gebildet, welche sich wiederum, wie im Abschnitt 1.2.3, durch die Riemannlöser aus den benachbarten Volumen berechnen. Die mittleren beiden Volumen repräsentieren stets die eigentliche Querschnittsveränderung. Die Querschnitte an den Außenseiten entsprechen demnach denen des Rohres, die Querschnitte innen dem des Ventilhalses. Durch diees Querschnittsveränderung ergibt sich im Gegensatz zur Gl. (1.10) für den

Impulsstrom ein Quellterm, der in gleicher Form auch bei Querschnittsveränderungen im bereits bestehendem Rohrmodel selber Verwendung findet:

$$S_i = p_i \cdot (A_{i+1} - A_i) \tag{2.14}$$

Dieser Quellterm entspricht der Kraft, welche das Rohr auf das Finite-Volumen in Längsrichtung ausübt (vgl. (2.3)). Damit sich das Rohr nicht verformt, muss es mit dem

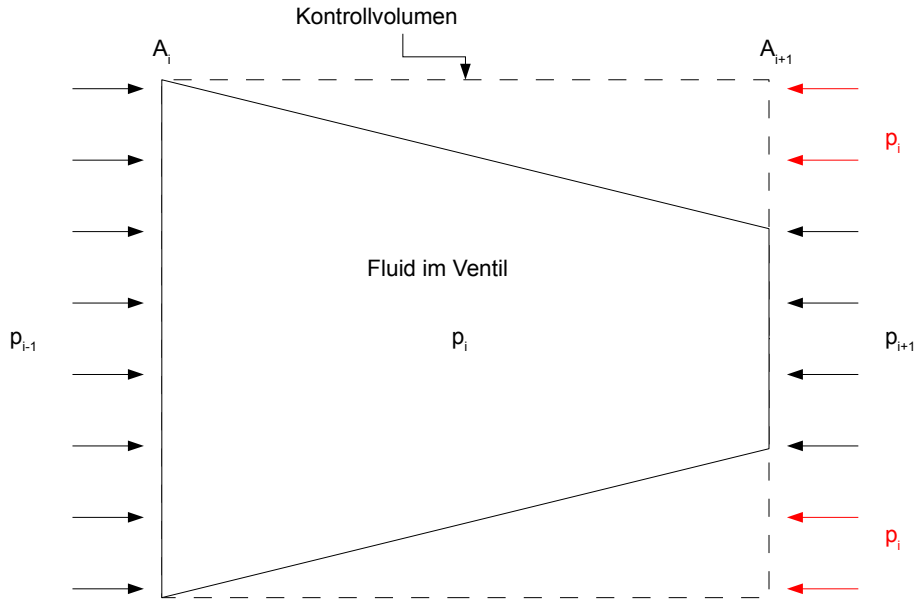

Abbildung 2.3: *Aufgrund der Querschnittsveränderung entsteht ein Quellterm für den Impulsstrom. Das Fluid drückt mit p_i auf die Rohrwand. Damit es zu keiner Verformung kommt, muss das Rohr den gleichen Druck auf das Fluid ausüben. Die resultierende Kraft ist der Druck mal die Differenz der Querschnitte.*

selben Druck auf das Fluid wirken, wie das Fluid auf das Rohr (hier p_i). Die projizierte Fläche in Längsrichtung ergibt sich aus der Differenz der Querschnittsflächen.
Um nun die unterschiedlichen Diskretisierungen dynamisch einzubinden, müssen die zu übergebenden Variablen für die Flussberechnung auch als dynamisches Array instanziiert werden. Diese werden dann nacheinander für die Flussberechnung beschrieben, so dass sich das in Abbildung (2.4) dargestellte Schaubild ergibt. Der Quellcode in Modelica sieht dabei wie folgt aus:

```
for i in 1:Discretization.halfStencilLength loop
  states[i,:] = cat(1,stencil_a.state_b[i:end], medium[1:i
      + Discretization.halfStencilLength - 1].state);
```

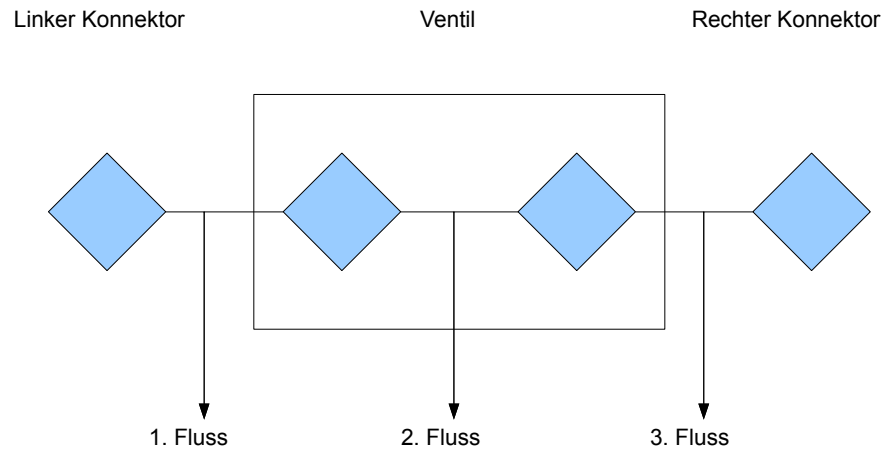

Abbildung 2.4: *Schaubild zur Flussberechnung im Ventil*

```
    states[n+2-i,:] = cat(1,medium[
       Discretization.halfStencilLength+2-i:end].state,
       stencil_b.state_a[1:end+1-i]);

    vs[i,:] = cat(1,stencil_a.v_b[i:end], v[1:i +
       Discretization.halfStencilLength - 1]);
    vs[n+2-i,:] = cat(1,v[Discretization.halfStencilLength
       +2-i:end], stencil_b.v_a[1:end+1-i]);

    A[i] = Modelica.Constants.pi/4*D^2;
    A[n+2-i] = Modelica.Constants.pi/4*D^2;
  end for;
```

Listing 2.5: *Zusammenstellung der Werte zur Übergabe an die Ausflussfunktion*

Es fehlen jetzt nur noch die Werte für den Fluss in der Mitte, welcher sich aus allen Zuständen innerhalb des Ventils zusammensetzt:

```
states[Discretization.halfStencilLength + 1,:] =
   medium.state;
vs[Discretization.halfStencilLength + 1,:] = v;
A[Discretization.halfStencilLength + 1] =
   Modelica.Constants.pi/4*D^2*(1 - abs(Modelcia.Math.cos(
```

```
u/180*Modelica.Constant.pi)));
```

Listing 2.6: *Zustände für die Berechnung des Flusses in der Mitte*

Schlussendlich werden die Outputvariablen der Konnektoren mit den neu errechneten Zustände an den Enden der einzelnen Komponenten beschrieben, sodass die benachbarten Komponenten im nächsten Zeitschritt mit diesen Werten rechnen können. Der gesamte Quellcode ist ebenfalls im Anhang (A.9) zu finden.

2.4 Ventilerstellung mit OpenFOAM

Bei der Arbeit mit OpenFOAM stellte sich schnell heraus, dass das Gitter für das Rohr, wie es in Abschnitt 1.2.5 beschrieben ist (O-Grid), bei der Simulation sehr viel Zeit benötigt. Da Modelica dazu die Strömung eindimensional berechnet, liegt der Schluss nahe, in OpenFOAM ebenfalls auf eine Dimension zu verzichten. Anschaulich lässt sich dies durch einen Schnitt durch die Längsachse des Rohrs beschreiben. Damit vereinfacht sich das komplexe 3D-Gitter zu einem einfachen Quader mit einer einzigen Zelle in die Tiefe (vgl. Abb. (2.5)). Hier ist links das komplexe Gitter und rechts die deutlich sichtbare

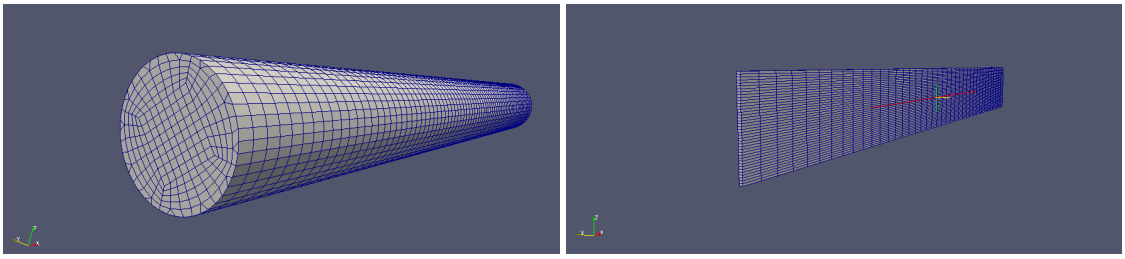

Abbildung 2.5: *Gegenüberstellung des komplexen Rohrgitters und des quasi 2D-Gitters*

Vereinfachung zu sehen. Die Veränderung der Ergebnisse, welches diese Vereinfachung mit sich bringt, ist in der Abbildungen (2.6) dargestellt. Links sind jeweils die Ergebnisse aus der 3D und rechts die aus der 2D-Berechnung eines ein Meter langen Rohres. Es lässt sich erkennen, dass das Einstellen des schlussendlichen Druckverteilung sowie die Endgeschwindigkeit nahezu identisch sind. Da die Rechendauer sich durch diese Vereinfachung von ca. 10h auf 1.25h verkürzt hat, ist im Weiteren mit dem einfachen Gitter gerechnet worden.

2.4.1 Erstellung des 2D-Gitters

Das 2D-Gitter für die Rohre samt Ventil ist für diese Arbeit mit der BlockMesh-Funktion von OpenFOAM erstellt worden. Die Erstellung des Gitters ist hierbei auf Text basierend. Dazu werden zunächst alle benötigten Punkte vorgegeben und diese dann zu Hexaedern zusammengefügt. Sollen Körper erzeugt werden, die weniger als acht Ecken haben, zum

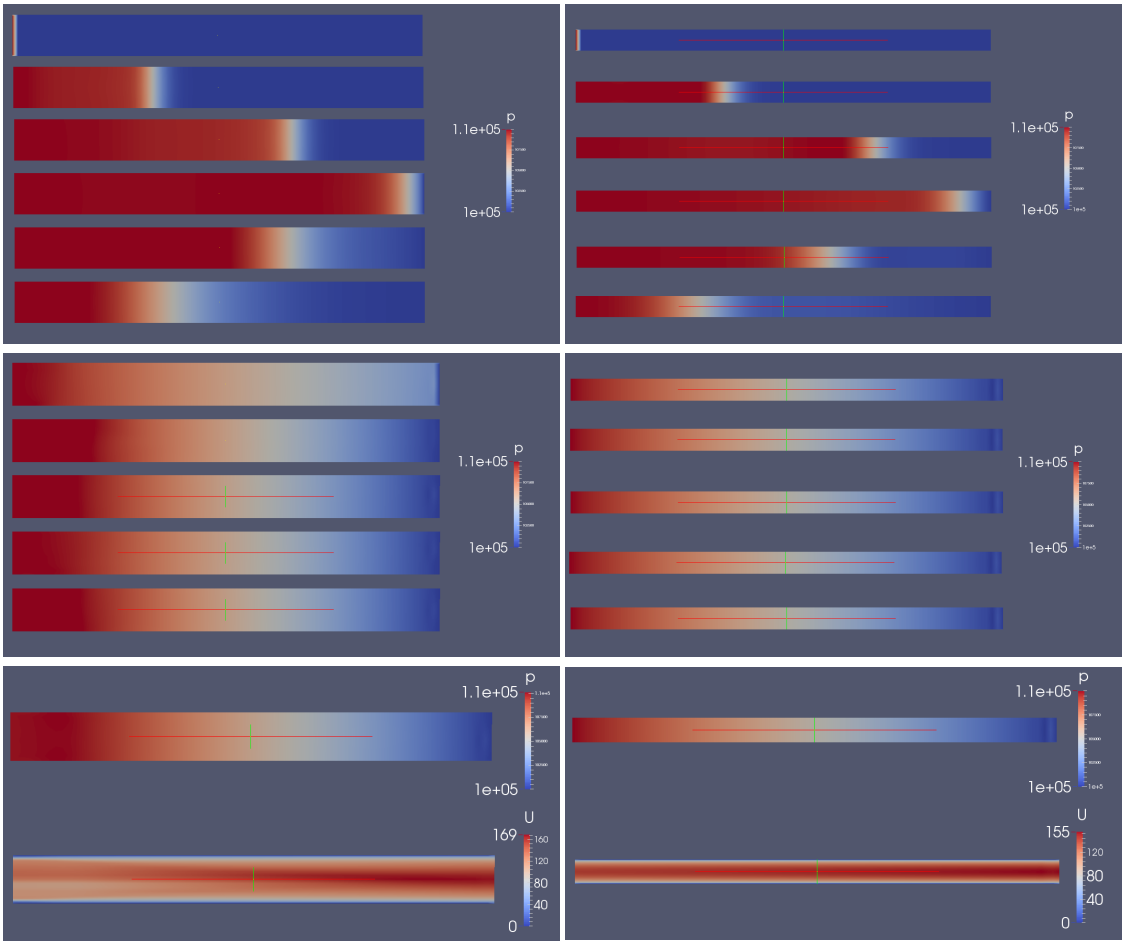

Abbildung 2.6: *Gegenüberstellung des Drucks in OpenFOAM. Links sind die Ergebnisse aus dem 3D zu sehen, rechts aus dem 2D. Der Zeitschritt ist von oben nach unten: 0ms, 1ms, 2ms, 3ms, 4ms, 5ms, 100ms, 200ms, 300ms, 400ms, 500ms und 1000ms. Ganz unten ist die Geschwindigkeitsverteilung nach 1000ms zu sehen. Trotz vernachlässigen einer Dimension sind die Resultate nahezu identisch.*

Beispiel Prismen, kann dies bewerkstelligt werden, indem man zwei Eckpunkte auf einen Punkt zusammen fallen lässt. Dieser Punkt wird dann doppelt angegeben, sodass auch bei Körpern mit weniger als acht Ecken stets acht Punkte angegeben werden müssen.
In diesem Fall baut sich das Gitter aus vier einzelnen Quadern auf. Der erste Quader bildet das erste Rohr. Es ist ein Meter lang, 5cm hoch und da OpenFOAM auch ein dreidimensionales Gitter für 2D-Simulationen benötigt 5mm tief. Die halbe Höhe gegenüber dem Rohr aus Modelica begründet sich darin, dass das Oberflächen-Volumen-Verhältnis für die Reibung gleich sein muss. In Längsachse teilt sich das Rohr in 40 Zellen, welche linear kleiner werden, sodass die erste Zelle fünfmal größer ist als die letzte. Dies begründet sich darin, dass am Ventil die Gradienten deutlich größer sind und somit das Gitter verfeinert werden muss (vgl. Abschnitt 1.2.5). Die Hochachse wird ebenfalls in 40 jedoch äquidistante Zellen geteilt. Die Tiefe ist mit einer Zelle diskretisiert.
An das erste Rohr gliedern sich oben und unten zwei weitere Quader an, sodass in der Mitte eine Aussparung entsteht: Die Ventilklappe. Da es sich in der Realität um ein Schmetterlingsventil handeln soll, wird hier ebenso mit der projizierten Fläche gerechnet. Damit lässt sich mit Hilfe des Kosinus der Öffnungswinkel direkt in die jeweilige Höhe der Quader umrechnen. In der Höhe wird das Ventil jeweils in so viel Zellen zerlegt, dass vom Rohr zum Ventil keine Veränderung des Gitters vorliegt. Das Ventil selber ist 1cm lang und wird längs in zwei Zellen aufgespalten. Ebenso wie das Rohr ist es 5mm tief. An das Ventil schließt sich ein weiteres 1m langes Rohr an, welches von seinem Gitteraufbau gegenüber dem ersten Rohr spiegelverkehrt aufgebaut ist. In Abbildung (2.7) ist das Gitter der linken Hälfte mit einem zu 40% geöffnetem Ventil zu sehen, die rechte schließt sich spiegelsymmetrisch an. Für den Vergleich mit dem Ventilen aus Modelica

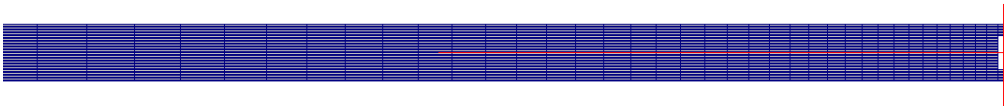

Abbildung 2.7: *Linke Seite einer Rohrleitung mit 40% geöffnetem Ventil. Die rechte Seite schließt sich symmetrisch an.*

entstanden Gitter mit 10%, 20%, 40%, 80% und voll geöffnetem Ventil. Dies entspricht den Ventilstellungen von $26°, 37°, 53°, 78°$ und $90°$. Das voll geöffnete Ventil besteht im Gegensatz zu den anderen nur aus einem 2m langem Quader mit einem gleichmäßigem Gitter.
Die hier gemachte Vereinfachung in eine Verengung gegenüber der schräg stehenden Ventilklappe begründet sich in der Einfachheit des Gitters. Insbesondere an den Übergängen würden Sprünge im Gitter für eine schräge Klappe entstehen. Bei geringen Ventilöffnungen ist diese Vereinfachung aufgrund der Kleinwinkelnäherung vertretbar, genauso bei sehr großen (insbesondere die Gitter für das vollständig geöffnete Ventil sind identisch). Hierzwischen wurde die Ungenauigkeit, besonders durch das Lee hinter der Ventilklappe, in Kauf genommen (s. Kap. 3.3).

2.4.2 Vorgeben der Randbedingungen

Im Ordner 0 des Simulationsverzeichnis werden die Randbedingungen und die Initialisierungswerte vorgegeben. Der Ordner gliedert sich in die Dateien p (Druck), U (Geschwindigkeit) und T (Temperatur), sowie in die für die Turbulenz notwendigen Dateien k (kinematische Turbulenzenergie) und epsilon (isotropen Dissipationsrate). Um mit den in Modelica benutzten Randbedingungen zu simulieren wurden die Randbedingungen in OpenFOAM wie folgt gewählt und können im Anhang in den einzelnen Dateien nachgelesen werden.
Für den Druck wurde gleich wie in Modelica für das Inlet ein Druck von 1.1bar vorgegeben und für das Outlet ein Druck von 1bar. Genauso nachvollziehbar ist die Randbedingung „zeroGradient“ für die Rohrwände bzw. die Ventilklappe, ergo soll an diesen Stellen kein Druckgradient vorhanden sein: Es herrscht auf der Innenseite der Wand der gleiche Druck wie auf der Außenseite. Initialisiert wurde das Fluid, ebenfalls wie in Modelica, mit 1bar.
Noch einfacher sind die Bedingungen für die Geschwindigkeit: An den Wänden soll jeweils keine Geschwindigkeit vorliegen ($U = U_x = U_y = U_z = 0$), damit Reibung berücksichtigt wird. Am Inlet und am Outlet soll das Fluid ungestört ein- bzw. ausfließen können und somit ist hier die Randbedingung zeroGradient. Damit muss auch kein Einlaufverhalten berücksichtigt werden. Ebenso wie in Modelica wird das Fluid in Ruhe initialisiert.
Die Inlet- und Outlet-Temperatur wurde wie in Modelica zu 300K gewählt und ebenso wird das Fluid mit dieser Temperatur initialisiert. Damit sich das Rohr adiabat verhält, wurden für die Wände die Randbedingung zeroGradient für die Temperatur gewählt. Da nun keine Temperaturdifferenz zwischen Fluid und Rohr vorliegt, kommt es zu keinem Wärmestrom, womit das Rohr adiabat ist. Die Werte für k und ϵ wurden jeweils aus einem Tutorial von OpenFOAM übernommen. Dieses Tutorial simuliert eine 2D-Überschallströmung um ein Prisma und entspricht damit einem ähnlichen Aufbau wie dem Rohr mit Ventil. Hierbei wird mit einer Wandfunktion die Werte an den Wenden berechnet. Abweichend vom Tutorial wurden die Bedingungen am Inlet und am Outlet zu „zeroGradient“ gewählt, da hier ebenso das Fluid nicht beeinflusst werden soll (im Tutorial ist eine feste Eintrittsgeschwindigkeit vorgegeben).
Damit die Simulation in 2D abläuft, müssen alle Randbedingungen der Vorder- und Rückseite durch „empty“ beschrieben werden.

2.4.3 Simulation starten und Ergebnisse ausgeben

Um die Simulation zu starten, muss man mit Hilfe der Konsole in den Simulationsordner gehen und den Befehl „sonicFoam“ eingeben. Die Ergebnisse der Simulation werden dann in Ordner geschrieben, die jeweils als Namen den Zeitschritt der Berechnung haben. Nach der Berechnung lassen sich die Ergebnisse mit Hilfe des Befehls „paraFoam“ auf der graphischen Benutzeroberfläche ParaView ausgeben.

3 Ergebnisse und Auswertung

In dieser Arbeit wurden vier verschiedene Ventilmodelle vorgestellt: Das erste Ventilmodel gehört zur Standardbibliothek in Modelica. Das zweite Ventilmodel wurde ebenso in Modelica erstellt und berechnet den Massenstrom über die Ausflussfunktion. Das dritte, ebenfalls in Modelica erstellt, wird durch die Eulergleichungen berechnet. Das letzte ist eine CFD-Simulation in OpenFOAM. In diesem Kapitel werden die Unterschiede unter besonderer Berücksichtigung der Ergebnisse aus Geschwindigkeit (resp. Massenstrom) und Druck sowie der Rechenzeit herausgearbeitet. Dabei wird auf die Anwendbarkeit dieser vier als Model für ein Schmetterlingventil eingegangen.

3.1 Vergleich des Standardventils mit dem Ventil mit Ausflussfunktion

Sowohl dem Standardventilmodel als auch dem Ventilmodel mit der Ausflussfunktion liegt das selbe Prinzip zu Grunde. Bei beiden Varianten wird der Druck vor und hinter dem Ventil für die Berechnung des Massenstroms verwendet. Ebenso verbindet beide Ventilmodelle, dass die Enthalpie und damit für ideale Gase auch die Temperatur vor und hinter dem Ventil gleich groß sind.
In den Simulationen mit den oben genannten Randbedingungen von 1.1bar und 1bar zeigen sich nahezu identische Werte für den Druckverlauf vor und hinter beiden Ventilen, sowie für den späteren Einbaufall wichtigen Massenstrom. In Abb. (3.1) ist der Zeitverlauf dieser Werte beider Ventile zu sehen. Hierbei ist zu beachten, dass die Ventile linear von $t = 0$ bis $t = 3$ geöffnet werden und dann ganz geöffnet bleiben. Dabei gilt für beide Ventile, dass der Öffnungsquerschnitt linear anwächst und nicht der Öffnungswinkel. Hier ist auch zu sehen, dass der berechnete Druck durch die Standardbibliothek bei geringer Ventilöffnung zu starker Oszillation neigt, wohingegen die Oszillation beim zweiten Ventilmodel schnell abklingt, wie es auch im realen Fall zu erwarten ist. Die anfängliche Stoßfront baut sich aufgrund der Dissipation innerhalb weniger Zehntelsekunden fast vollständig ab. Durch die starken Oszillationen ist die Rechenzeit der Standardbibliothek etwa um den Faktor 2 größer. Ebenso fällt auf, dass bei voll geöffnetem Ventil der Druck vor dem Ventil nur wenig unter 1.1bar liegt und hinter dem Ventil nur knapp über 1bar. Es liegt hier folglich, obwohl das Ventil vollständig geöffnet ist, kein linearer Druckabfall über das gesamte System vor. Dieser Punkt wird in Abschnitt 3.2 noch einmal aufgenommen. Damit gewährleistet ist, dass die gute Übereinstimmung kein Zufall ist, wurde die gleiche

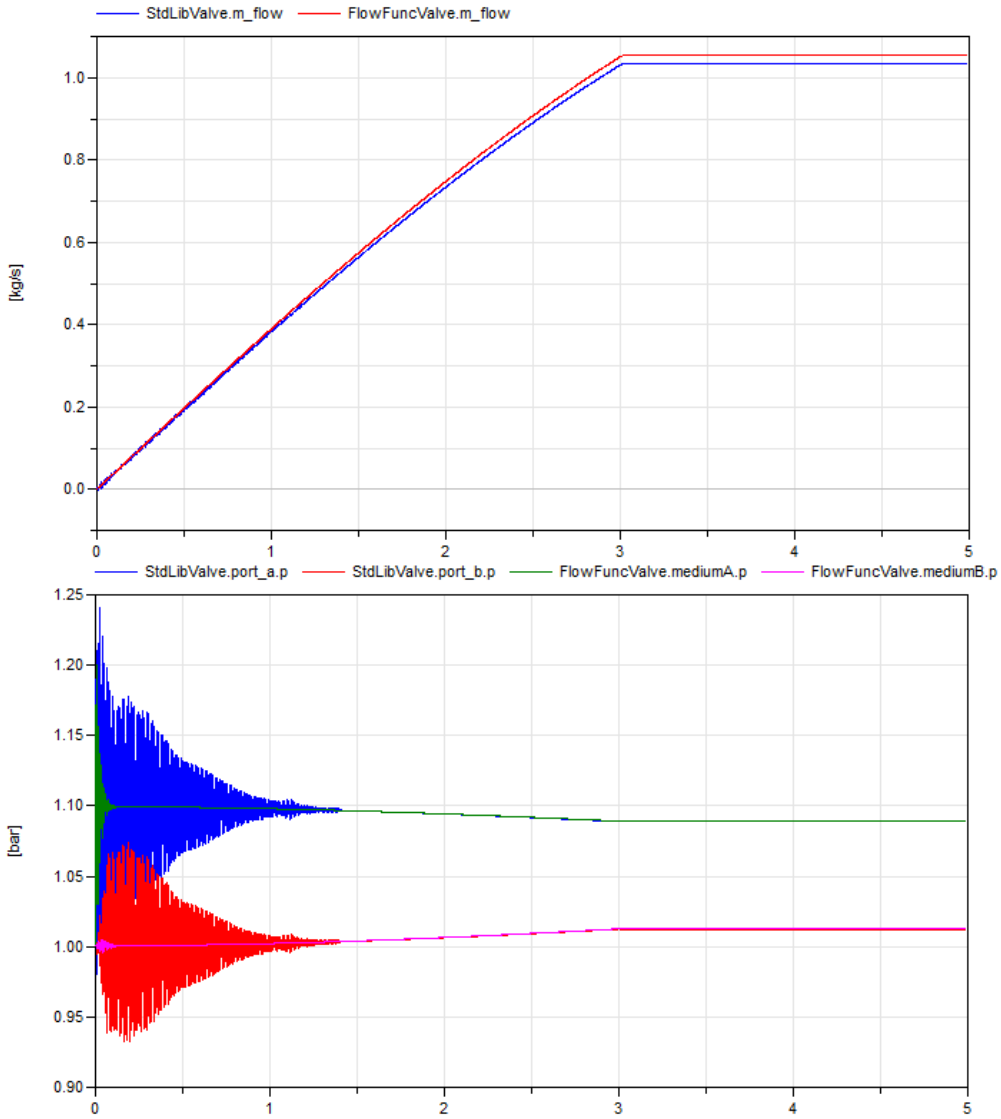

Abbildung 3.1: *Bei geringer Druckdifferenz (1.1bar auf 1bar) ist der Unterschied der berechneten Massenströme im Öffnungsvorgang vernachlässigbar. Auch die Druckkurven sind nahezu identisch, abgesehen von den zunächst stark oszillierenden Werten für das Standardventil. Hierin zeigt sich die Verbesserung der Gasdynamikbibliothek gegenüber der Standardbibliothek.*

Simulation mit den Randbedingungen 1.5 und 0.5bar wiederholt. Hier ist bereits eine deutlichere Abweichung zu sehen (Abb. (3.2)). Nach etwa 1.3 Sekunden steigt der Druck

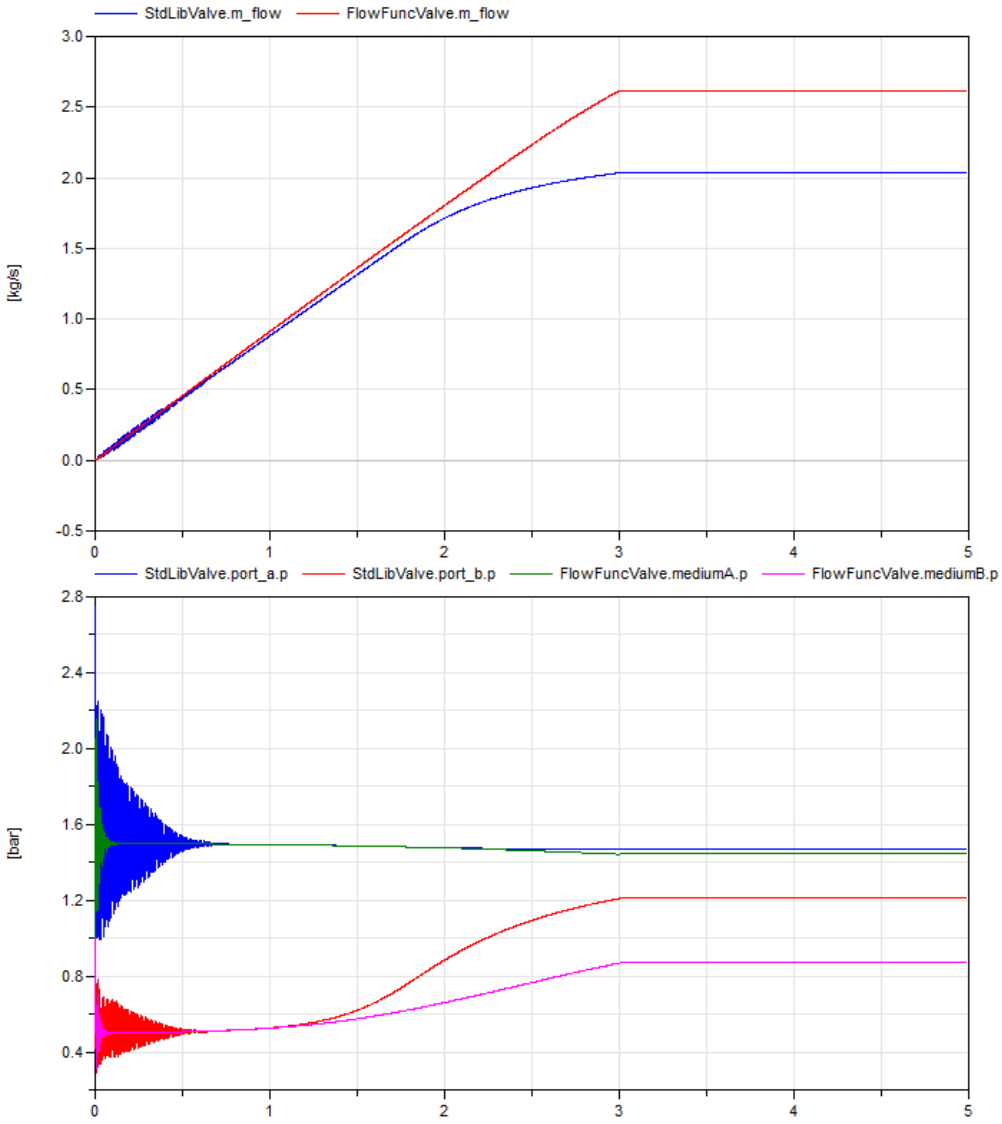

Abbildung 3.2: *Auch bei höheren Druckdifferenzen verhalten sich beide Ventile sehr ähnlich. Abweichungen entstehen jedoch nach etwa halben Öffnungsvorgang (1.5s). Hier steigt der Gegendruck des Standardventils deutlich schneller an, was in einem geringeren Massenstrom resultiert. Grund hierfür sind Gleichungen zum kompressiblen Verhalten im Standardventil.*

hinter dem Ventil aus der Standardbibliothek deutlich schneller an, wohingegen der

Druckabfall vor dem Ventil leicht hinter dem Ventil mit der Ausflussfunktion zurückbleibt. Der nun deutlich geringere Druckgradient hat zur Folge, dass der Massenstrom nicht länger linear steigt, sondern in einer Kurve tangential in sein Maximum übergeht. Eine explizite Erklärung für dieses Verhalten wurde in der Dokumentation nicht gegeben. Es wurde jedoch festgestellt, dass sich ein Kompressibilitätsfaktor zum selben Zeitpunkt der Abweichung im Massenstrom verändert. Dadurch ist die Berechnung durch die Standardbibliothek an dieser Stelle womöglich präziser.
Da sich dennoch das Ventil aus der Standardbibliothek und das Ventil mit Ausflussfunktion stark ähneln, wird im Weiteren auf Vergleiche der anderen Ventilmodellen mit dem zuerst genannten verzichtet.

3.2 Vergleich des diskretisierten Ventil mit dem mit Ausflussfunktion

Da das Model des diskretisierte Ventil das Ventilmodel mit der Ausflussfunktion ablösen soll, ist der Vergleich dieser beiden Ventilvarianten von besonderem Interesse für diese Arbeit. Dabei ist der wichtigste Punkt, wie sich die Massenströme der Ventile verhalten, da dieser maßgeblich durch das jeweilige Ventil geregelt werden soll.
Bei der Betrachtung beider Massenströme fällt schnell auf, dass die Ausflussfunktion eine nahezu proportionale Abhängigkeit zwischen Öffnungsquerschnitt und Massenstrom errechnet, während die Ventilcharakteristik des diskretisierten Ventil eine S-Form aufweist (vgl. Abb. (3.3)). Die unterschiedlichen Ventilcharakteristiken begründen sich in der stark unterschiedlichen Berechnung der Massenströme. Der Massenstrom des Ventils mit Ausflussfunktion berechnet sich nach Gleichung (2.9) und sei hier nochmal aufgeführt:

$$\dot{m} = A\Psi(p_2/p_1, \kappa)\sqrt{2p_1\rho_1} \tag{3.1}$$

Da der Isentropenexponent κ eine Konstante ist und sich sowohl der Quotient der Drücke als auch der Druck und die Dichte vor dem Ventil gegenüber dem Ventilquerschnitt nur geringfügig ändern, lässt sich hieraus die Proportionalität zwischen Massenstrom und Querschnittsfläche begründen. Der Massenstrom des diskretisierten Ventils hingegen berechnet sich durch die erste Gleichung des numerischen Flusses über eine Zellwand. Für den Rusanovfluss ist dieser gegeben durch (vgl. Gl. (1.33)):

$$\dot{m} = A\frac{1}{2}(\rho_l v_l + \rho_r v_r) - \frac{1}{2}\lambda_{max}(\rho_r - \rho_l) \tag{3.2}$$

Unter der Annahme, dass sich die Zustände links und rechts nur geringfügig unterscheiden, ergibt sich sofort die allgemeine Gleichung für den Massenstrom:

$$\dot{m} = A \cdot \rho \cdot v \tag{3.3}$$

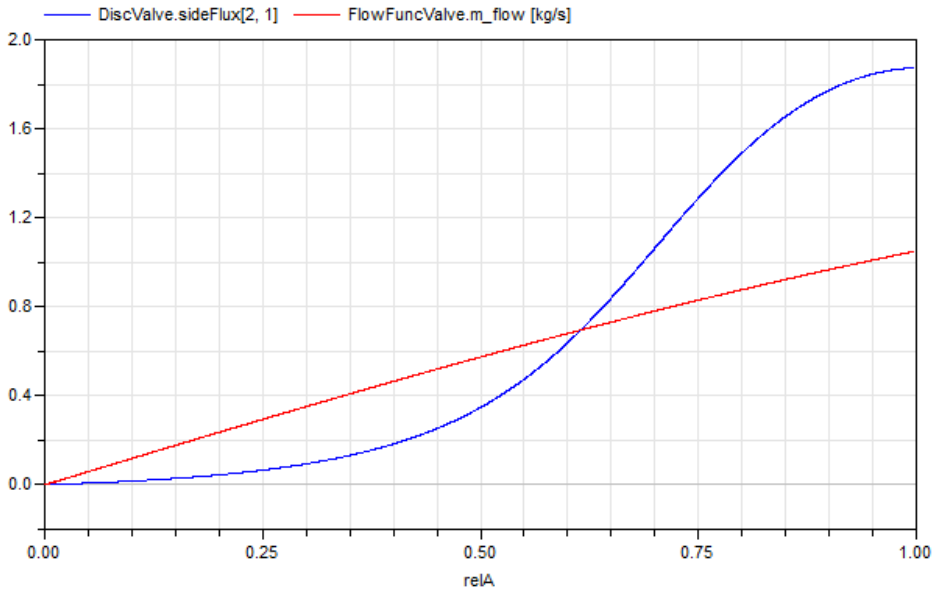

Abbildung 3.3: *Gegenüberstellung der Massenströme des diskretisierten Ventil und des Ventils mit Ausflussfunktion in Abhängigkeit des relativen Querschnitts. Hier ist zu sehen, dass sich die Ventilcharakteristiken deutlich unterscheiden. Während das Ventilmodel mit Ausflussfunktion eine lineare Steigung berechnet, ergibt sich eine S-Kurve aus den Eulergleichungen.*

Wie oben soll nun auch hier angenommen werden, dass sich die Dichte nur geringfügig gegenüber den anderen beiden Variablen verändert. Es bleibt hier dennoch nicht nur die Abhängigkeit vom Ventilquerschnitt sondern auch die Abhängigkeit von der Geschwindigkeit des Fluids. Die Geschwindigkeit des Fluids im erstgenannten Ventil berechnet sich dann zwar nach der gleichen Gleichung, dann jedoch in Abhängigkeit von dem bereits errechneten Massenstrom. Obwohl also keine explizite Geometrie vorgegeben wird, nur eine Queschschnittsänderung, werden zwei deutlich unterschiedliche Ventilcharakteristiken berechnet. Hält man nun die tatsächliche Ventilcharakteristik eines Schmetterlingsventil daneben, fällt auf, dass diese ebenfalls eine S-Form aufweist (vgl. Abb. (3.4)). Diese

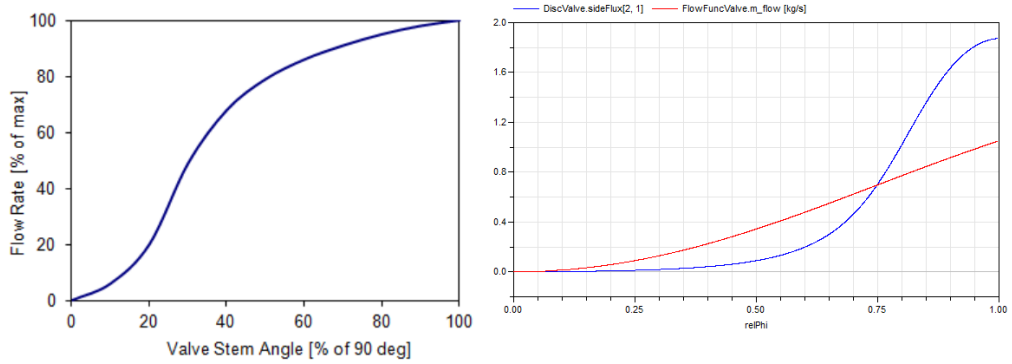

Abbildung 3.4: *Anhand der Ventilcharakteristik des realen Schmetterlingsventils (links, [14]) lässt sich erkennen, dass die Eulergleichungen (rechts, blau) diese besser wiedergeben als die Ausflussfunktion (rechts, rot), da es sich hier ebenfalls um eine S-Kurve handelt. Die S-Kurve ist hier jedoch verschoben, dies kann u.a. an der Druckdifferenz liegen.*

S-Kurve ist gegenüber der S-Kurve des diskretisierten Ventils nach links verschoben. Wird Beispielsweise der Druckgradient erhöht, verschiebt sich letztere S-Kurve ebenfalls weiter nach links.
Vergleicht man die Druckkurven beider Ventile fällt auf, dass die Strömung des diskretisierten Ventils bei voll geöffnetem Ventil einen linearen Abfall des Druckes über das gesamte Rohrsystem aufweist. Dies gleicht der Strömung einer Rohrleitung ohne Ventil. Wie schon im Abschnitt 3.1 erwähnt, passiert dies bei dem Ventil mit Ausflussfunktion nicht, damit ist das diskretisierte Ventil an dieser Stelle plausibler. Bei der Ausflussfunktion ist der Quotient aus Gegen- und Vordruck von entscheidender Bedeutung: Je weiter dieser Quotient gegen eins strebt, desto geringer ist das Ergebnis der Ausflussfunktion und damit der Massenstrom. Dies macht jedoch bei sich öffnenden Ventil keinen Sinn, da hier der Massenstrom ansteigt. Somit steigt der Druck hinter dem Ventil, bzw. sinkt der Druck vor dem Ventil nur geringfügig im Öffnungsvorgang. Der Massenstrom im diskretisiertem Ventil ist unabhängig vom Druckquotienten. Hier kann sich folglich die lineare Druckverteilung einstellen, ohne dass der Massenstrom beeinflusst wird. In diesem

Punkt sind die unterschiedlichen Ideen der beiden Ventile am offensichtlichsten. Die Änderung des Drucks während des Öffnungsvorganges ist in Abb. (3.5) zu sehen.
Als letzter Vergleich soll an dieser Stelle die Veränderung der Geschwindigkeit mit

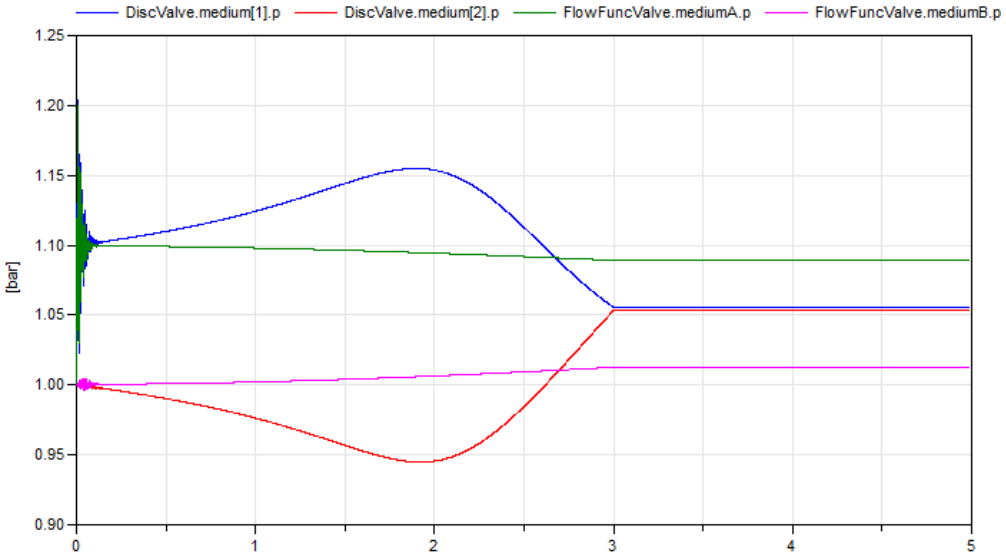

Abbildung 3.5: *Ebenso wie die Kurve des Massenstroms weichen auch die Druckkurven vor und hinter den Ventilen deutlich voneinander ab. Während sich die berechneten Drücke im Öffnungsvorgang durch die Ausflussfunktion nur minimal annähern, resultieren die Eulergleichungen in fast gleichgroßen Drücken vor und hinter dem Ventil. In der Simulation mit der Ausflussfunktion fällt nahezu der gesamte Druck über dem Ventil ab, im diskretisierten Fall fällt er bei vollständig geöffnetem Ventil linear über das gesamte Rohrsystem ab. Letzteres ist plausibler, da im Idealfall das Ventil keine Auswirkung auf den Druckverlauf haben sollte, wenn dieses geöffnet ist.*

zunehmenden Druckgradienten dienen. Dazu wurden die Drücke aus dem vorliegenden Anwendungsfall benutzt. Diese sind:

$$7bar \rightarrow 5.5bar$$
$$5.5bar \rightarrow 3.7bar$$
$$3.7bar \rightarrow 1.9bar$$

Um dies zu simulieren wurde mit Hilfe eines Python-Scripts der Druck am Inlet und am Outlet stetig von 1.5bar auf 7bar in 30 Schritten erhöht. Die resultierenden Geschwindigkeiten im stationären Fall bei den Ventilstellungen $26°, 37°, 53°$ und $90°$ wurde als Fläche über die Drücke in den Abb. (3.6) bis (3.9) aufgetragen (grün: Ventil mit Ausflussfunktion; rot: diskretisiertes Ventil).

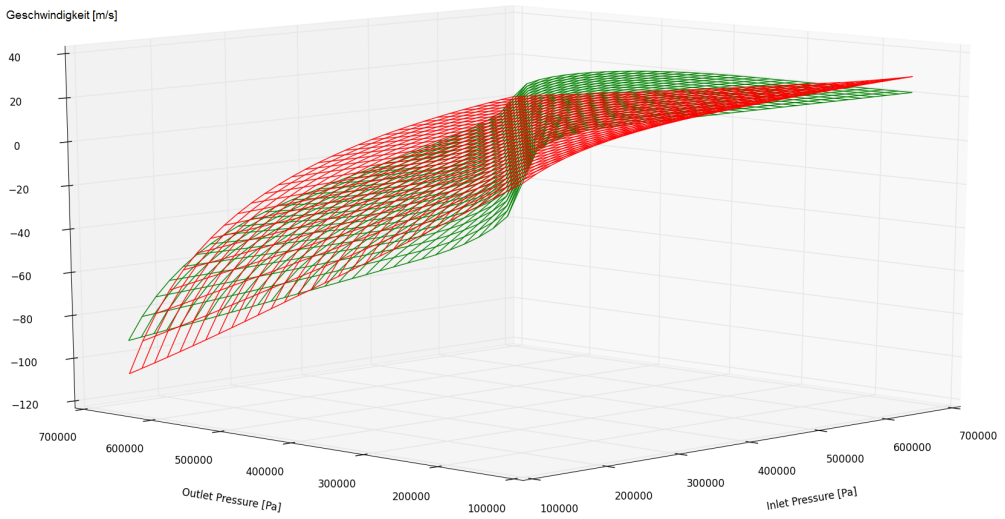

Abbildung 3.6: *Geschwindigkeit als Fläche über Inlet- und Outlet-Drücken für die Ventilstellung* 26°

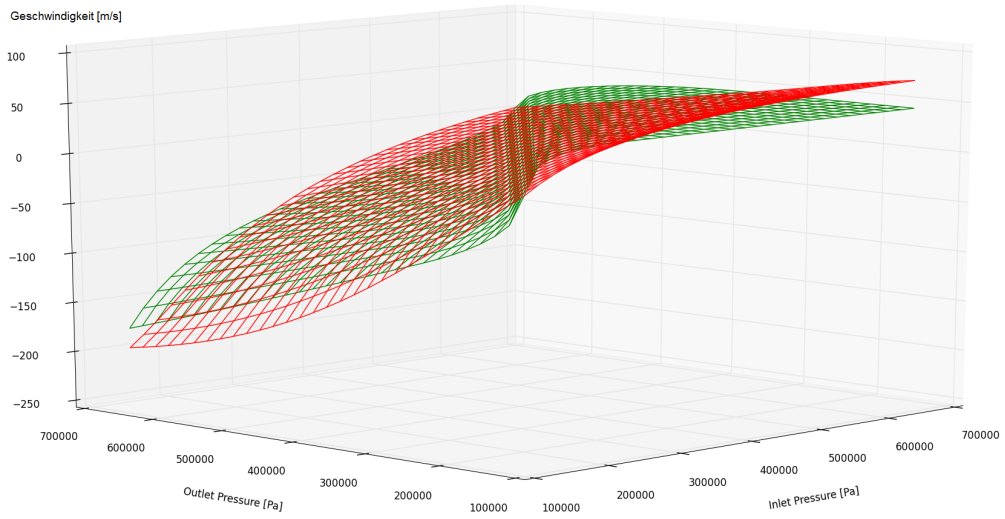

Abbildung 3.7: *Geschwindigkeit als Fläche über Inlet- und Outlet-Drücken für die Ventilstellung* 37°

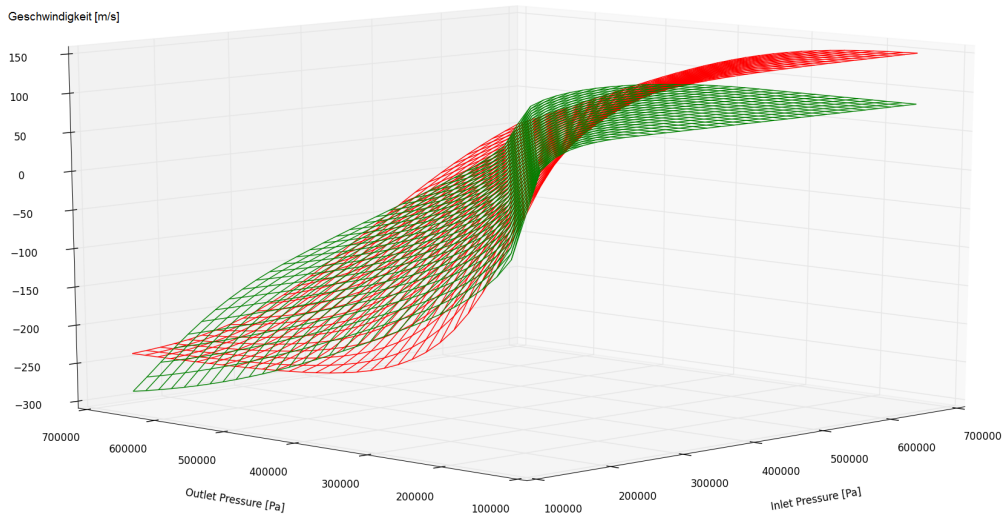

Abbildung 3.8: *Geschwindigkeit als Fläche über Inlet- und Outlet-Drücken für die Ventilstellung* 53°

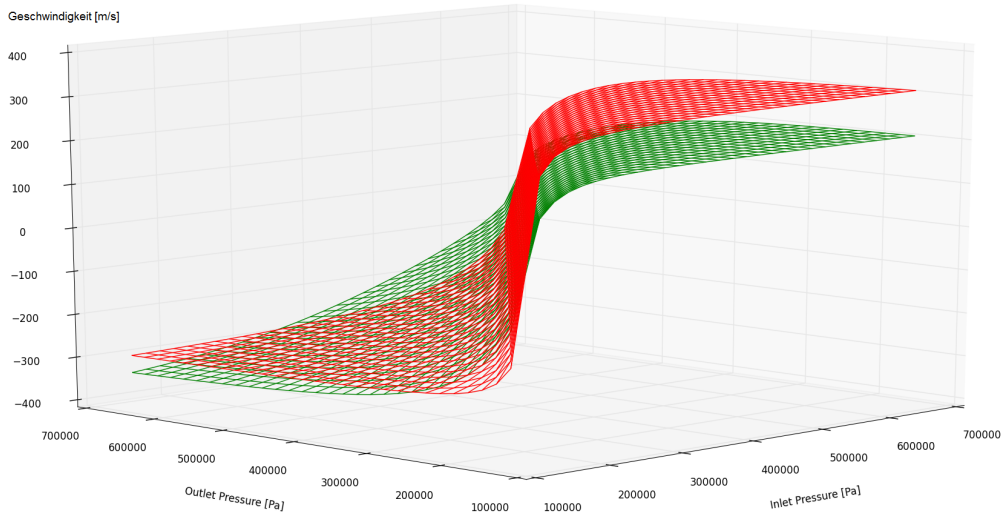

Abbildung 3.9: *Geschwindigkeit als Fläche über Inlet- und Outlet-Drücken für die Ventilstellung* 90°

Zunächst lässt sich erkennen, dass alle Flächen nicht symmetrisch zur Isobaren sind. Dies liegt daran, dass die Geschwindigkeit jedes mal links vom Ventil gemessen wurde. Damit ist der Messpunkt in Strömungsrichtung sowohl vor (Inlet-Druck größer Outlet-Druck) als auch hinter dem Ventil (Inlet-Druck kleiner Outlet-Druck). Da über dem Ventil der Druck abfällt, sinkt auch die Dichte des Fluids. Dies resultiert dann sofort in einer höheren Geschwindigkeit, damit die Kontinuitätsgleichung weiterhin erfüllt bleibt. Damit hängt die Differenz der Geschwindigkeiten vom Druckgradienten am Ventil ab. Da bei vollständig geöffnetem diskretisiertem Ventil dieser Druckgradient sehr klein ist (vgl. Abb. (3.5)), ist hier die Fläche nahezu symmetrisch. Dass auch die Geschwindigkeitsfläche hinter dem Ventil mit Flussfunktion bei vollständig geöffnetem Ventil abflacht liegt daran, dass bei hohen Geschwindigkeiten aufgrund der Reibung zunehmend der Druck in den Rohren abfällt. Damit wirkt sich ein Absenken des Druckes am Inlet (bei negativem Druckgradienten) nur noch wenig auf den Druck direkt am Ventil aus. Bei geringen Geschwindigkeiten ist der Druckabfall im Rohr zu vernachlässigen. Deshalb ist die Geschwindigkeit hinter dem Ventil bei geringer Öffnung und hohem Druckgradienten um einiges höher als vor dem Ventil.
Auch fällt auf, dass das Ventil mit Ausflussfunktion, sowie die Ausflussfunktion auch definiert ist, bei überkritischem Druckverhältnis sperrt, d.h. vor dem Ventil bildet die Geschwindigkeit in Abhängigkeit von unterschiedliche überkritischen Drücken eine horizontale Ebene:

$$\dot{m} = A_V \cdot \psi \cdot \sqrt{p_1 \rho_1} \tag{3.4}$$

$$\dot{m} = A \cdot \rho_1 \cdot v_1 \tag{3.5}$$

$$\Rightarrow v_1 = \frac{A_V}{A} \cdot \psi \cdot \sqrt{\frac{p_1}{\rho_1}} \tag{3.6}$$

Dass das Verhältnis von Druck zu Dichte auch konstant ist, liegt daran, dass sich die Temperatur vor dem Ventil mit steigender Temperatur nicht relevant verändert und diese Größen über die Ideale Gasgleichung zusammenhängen. Bei dem diskretisierten Ventil flacht die Kurve mit steigenden Druckgradienten zwar ebenfalls ab, es bleibt jedoch eine gewisse Steigung vorhanden. Die Steigung hängt unter anderem auch vom Öffnungswinkel ab: Bei kleinen Winkeln ist die Steigung deutlich sichtbar, während bei vollständig geöffnetem Ventil die Steigung zunehmend gegen 0 geht. Mit verantwortlich hierfür wird die bereits beschriebene Druckkurve sein, die sich bei kleinen und großen Ventilöffnungen deutlich unterscheiden. Auf die gleiche Art wird sich die Temperatur einwirken, die nicht wie im Zusammenhang der Ausflussfunktion konstant gehalten wird. Über dem diskretisierten Ventil fällt sie deutlich ab.
Die Rechenzeit ist bei beiden Ventilvarianten in etwa gleich: Das diskretisierte Ventil wird nur minimal schneller berechnet, was einen kleinen Vorteil darstellt.

3.3 Vergleich des diskretisierten Ventil mit den CFD-Simulationen

Da die CFD-Simulation die erste Simulation mit tatsächlicher Vorgabe einer Geometrie ist und des Weiteren nur die Randbedingungen vorgegeben wurden, liegt die Annahme nahe, dass es sich hierbei um die genaueste Simulation handelt. Da jedoch auch hier mit der projizierten Fläche für das Ventil gerechnet wird, weicht die Simulation an diesem Punkt von der Realität ab. Dennoch soll der Vergleich der CFD-Simulation mit dem diskretisierten Ventil stark in die Bewertung des Ventils einfließen.
Um die Geschwindigkeiten des Fluids beider Simulationen zu vergleichen muss zunächst die ausgebildete Rohrströmung der CFD-Simulation in eine Durchschnittsgeschwindigkeit über den Querschnitt umgerechnet werden. Dazu wurde das ursprüngliche Strömungsbild durch mehrere Stützstellen linear angenähert (vgl. Abb. (3.10)) und so mit Hilfe des Sehnentrapezverfahren die Durchschnittsgeschwindigkeit berechnet. In Tabelle (3.1) sind die Ergebnisse für das vollständig geöffnete Ventil zusammengefasst. Aus Symmetriegründen wird hier nur mit den Werte bis zur Mitte des Rohres gerechnet.

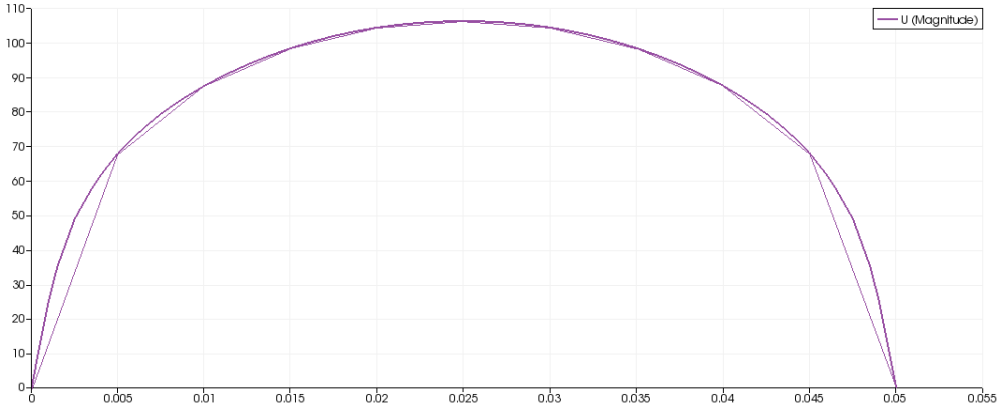

Abbildung 3.10: *Das Geschwindigkeitsprofil aus der OpenFOAM-Simulation mit vollständig geöffnetem Ventil an der Stelle x=0.5 zusammen mit der Annäherung des Sehnentrapezverfahrens.*

Nach dem selben Prinzip wurde nun auch die Durchschnittsgeschwindigkeiten der anderen Ventilstellungen berechnet, ebenso das Verhältnis von Durchschnittsgeschwindigkeit zu maximaler Geschwindigkeit. Diese Ergebnisse sind in Tab. (3.2) zusammengefasst.

Trägt man nun die Durchschnittsgeschwindigkeiten über den Grad der Öffnung des Ventils auf, ergibt sich die Ventilcharakteristik die in Abb. (3.11) zusammen mit den Ventilcharakteristik des diskretisierten Ventils und des mit Ausflussfunktion zu sehen ist. Der Öffnungsgrad ist hier wieder als Winkelstellung angegeben.
Es lässt sich erkennen, dass auch die CFD-Simulationen eine S-Förmige Kurve errechnen.

x	v(x)	$\frac{v_1+v_2}{2}$
0	0	-
0.005	68.02	34.01
0.01	87.57	77.8
0.015	98.47	93.02
0.02	104.46	101.47
0.025	106.33	105.4
-	v_{avg}	82.337
-	$\frac{v_{avg}}{v_{max}}$	0.774

Tabelle 3.1: *Berechnete Durchschnittsgeschwindigkeit bei vollständig geöffnetem Ventil aus der OpenFOAM-Simulation*

Grad der Ventilöffnung	v_{avg}	$\frac{v_{avg}}{v_{max}}$
10%	5.17	0.778
20%	14.44	0.774
40%	34.17	0.774
80%	70.79	0.774
100%	82.34	0.774

Tabelle 3.2: *Durchschnittsgeschwindigkeiten und das Verhältnis zur maximalen Geschwindigkeit für alle Ventilöffnungen bei 1.1bar auf 1bar*

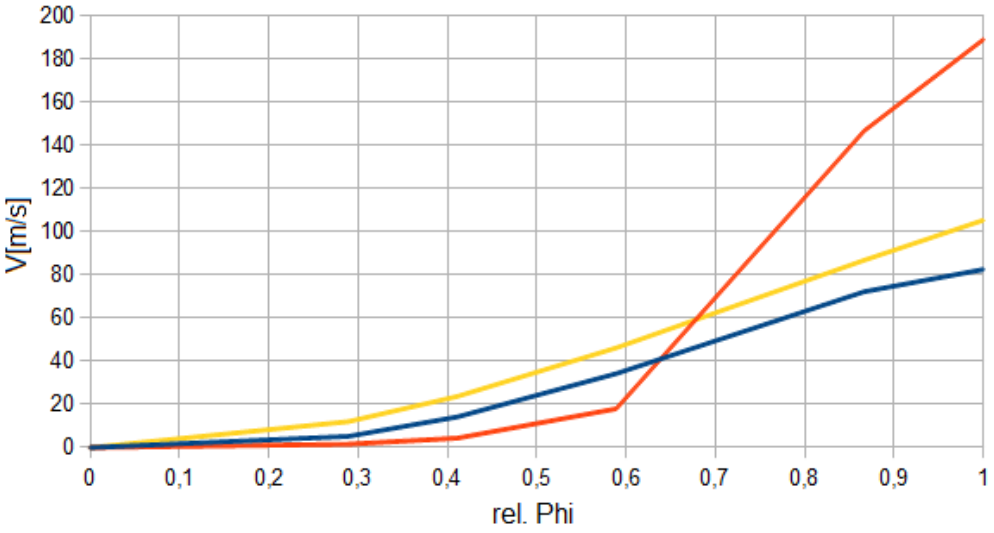

Abbildung 3.11: *Vergleich der Ventilcharakteristiken berechnet durch OpenFOAM (blau), des diskretisierten Ventils (rot) und des Ventils mit Ausflussfunktion (gelb) bei einem Druck von 1.1bar auf 1bar. Qualitative Aussage sind hier nicht möglich, da keine Realdaten vorliegen.*

Die Geschwindigkeiten sind hier jedoch bei geringen Ventilöffnungen um den Faktor 3.3 größer als die es diskretisierten Ventils, bei großen Ventilöffnungen jedoch um den Faktor 0.4 - 0.5 kleiner. Bei höheren Geschwindigkeiten scheinen die Verluste bei den CFD-Simulationen deutlich zuzunehmen. Während in Modelica nur mit Reibung innerhalb der Rohre und nicht im Ventil gerechnet wird, kommt bei OpenFOAM der Verlust durch die stark turbulente Strömung direkt hinter dem Ventil dazu (Dissipationsenergie). Hier entstehen auch Rückströmungen. Die Ausflussfunktion liefert stets ein höheren Wert (kleine Öffnungen Faktor 2.3, große 1.2).
Alle bisherigen Simulationen in diesen Abschnitt sind bei einem Druckgradient von 1.1bar auf 1bar entstanden. Um jedoch auch die Ergebnisse am Arbeitspunkt zu betrachten, wurden die gleichen Simulationen mit einem Druckgradient von 5.5bar auf 3.5bar durchgeführt. Es ergaben sich nun die mittleren Geschwindigkeiten, die in Tab. (3.3) aufgeführt sind, und damit die Ventilcharakteristik, die in Abb. (3.12) erneut mit den Ergebnissen aus Modelica zu sehen ist.

Bei diesen Druckdifferenzen ist die Abweichung zwischen dem diskretisierten Ventil und OpenFOAM nicht mehr ganz so stark. Bei geringen Ventilöffnungen ist die berechnete Geschwindigkeit in OpenFOAM nur noch um den Faktor 1.4 - 1.5 größer, bei großen beträgt sie etwa die Hälfte der Geschwindigkeit. Auch hier ist die Strömung durch die Ausflussfunktion wieder grundsätzlich größer als die entsprechenden Ergebnisse aus OpenFOAM. Bei kleinen Öffnungen um den Faktor 2, bei großen um den Faktor 1.2 - 1.3.

Grad der Ventilöffnung	v_{avg}	$\frac{v_{avg}}{v_{max}}$
10%	10.07	0.774
20%	24.22	0.778
40%	57,90	0.776
80%	120,82	0.775
100%	135.92	0.776

Tabelle 3.3: *Durchschnittsgeschwindigkeiten und das Verhältnis zur maximalen Geschwindigkeit für alle Ventilöffnungen bei 5.5bar auf 3.5bar*

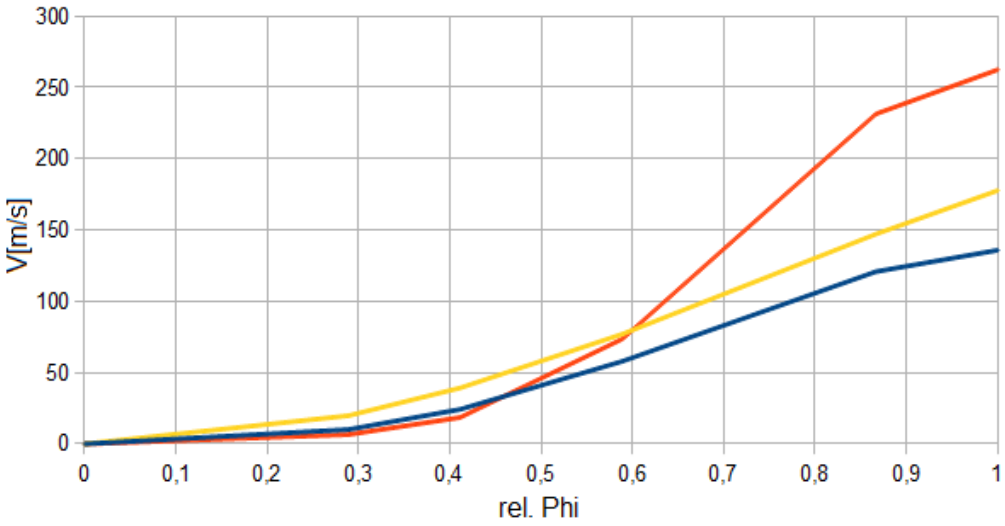

Abbildung 3.12: *Vergleich der Ventilcharakteristiken berechnet durch OpenFOAM (blau), des diskretisierten Ventils (rot) und des Ventils mit Ausflussfunktion (gelb) bei einem Druck von 5.5bar auf 3.5bar. Qualitative Aussage sind hier nicht möglich, da keine Realdaten vorliegen.*

Nach Betrachtung der Geschwindigkeiten soll im Folgenden der Druckverlauf entlang des Rohrsystems als letzte wichtige Größe betrachtet werden. Dazu sind exemplarisch in Abb. (3.13) die Druckkurven von Modelica und OpenFOAM bei dem Druckgradient 1.1bar auf 1bar bei 10% (26°), 80% (78°) und vollständige geöffnetem Ventil dargestellt.
Es ist zu erkennen, dass sich die Druckkurven in den CFD-Simulationen deutlich

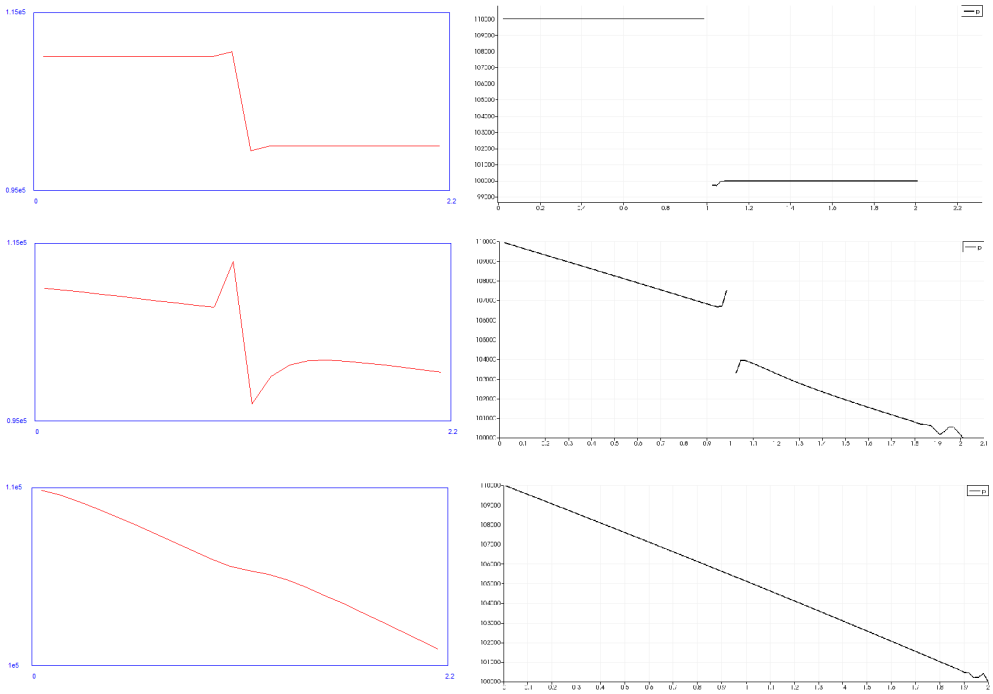

Abbildung 3.13: *Vergleich der Druckkurven simuliert in Modelica (links) und OpenFOAM (rechts). Hier ist eine gute Übereinstimmung zu erkennen. Die Kurven sind, wie bereits erwähnt, zumindest für die vollständige Ventilöffnung plausibel (immer geringerer Druckabfall über dem Ventil mit zunehmendem Öffnungsgrad). Die Sprungstellen in den CFD-Simulationen stellen das Ventil da. An dieser Stelle wird kein Druck berechnet. In der Graphik aus Modelica sind die Drücke vor und hinter dem Ventil linear verbunden.*

den entsprechenden des diskretisierten Ventils ähneln, und sich damit beide von den Druckkurven des Ventils mit Ausflussfunktion unterscheiden. Auch die CFD-Simulationen ergeben einen vollständigen Druckabfall über dem geschlossenen Ventil und einen immer höheren Druckabfall in den Rohren bei sich öffnendem Ventil, bis schließlich ein linearer Druckabfall bei vollständig geöffnetem Ventil vorliegt. Die Simulationen mit höherem Druckgradienten (5.5bar auf 3.5bar) ergeben qualitativ den selben Verlauf und sind deswegen hier nicht noch einmal aufgeführt.

Schließlich sei an dieser Stelle noch erwähnt, dass sich die Temperaturkurven ebenfalls gleichen: Sowohl das diskretisierte Ventil als auch die OpenFOAM-Simulationen zeigen bei kleinen Ventilöffnungen einen Temperaturanstieg zum Ventil hin und einen deutlichen Temperaturabfall über dem Ventil. Ebenso steigert sich auch die Geschwindigkeit, wie für das diskretisierte Ventil im vorherigen Abschnitt bereits erwähnt, im überkritischen Bereich bei Erhöhung der Druckdifferenz in den CFD-Simulationen.

3.4 Vergleich der unterschiedlichen Berechnung des numerischen Fluss

Um die einzelnen in Abschnitt 1.2.4 erläuterten Flüsse miteinander zu vergleichen wird hier der Faktor Rechenzeit und die Resultate aus den berechneten Massenströmen, welche dem ersten numerischen Fluss entsprechen, benutzt. Da jedoch keine exakten Lösungen und auch keine realen Daten vorliegen, ist letzteres rein quantitativ. Die Vergleiche der vorangegangenen Abschnitte sind jeweils mit dem Rusanovfluss entstanden. Dem entsprechend sind die Ergebnisse aus diesen Vergleichen in Zusammenhang mit den folgenden Unterscheidungen zu bewerten.
Zunächst soll rein objektiv die benötigte Rechenzeit der einzelnen Verfahren verglichen werden. Dazu wird erneut der bereits genannte Versuchsaufbau verwendet, jedoch wurde der Druck am Inlet auf 2bar erhöht, der am Outlet bleibt bei 1bar. In Tabelle (3.4) sind die benötigten Rechenzeiten für die fünf sekündige Simulation zu sehen. Die Flussberechnung

Numerische Methode	Benötigte Rechenzeit
Rusanovfluss	7.24s
Fluss nach Roe	(12.14s) Abbruch der Simulation nach 3.4s Rechenzeit und 1.4s simulierter Zeit
HLL-Verfahren	9.13s
MUSCL-Verfahren: Superbeelimiter	355s
MUSCL-Verfahren: Minbeelimiter	38.2s

Tabelle 3.4: *Zeitvergleich der unterschiedlichen numerischen Flussberechnungen. Der Fluss nach Roe bricht ab, da die Grenze von 200K des Mediummodels unterschritten wird.*

nach Roe bricht ab, da die berechnete Temperatur im Volumen hinter dem Ventil auf 200K fällt und damit außerhalb des Rahmen des Mediummodells liegt. Die tiefste Temperatur der anderen Diskretisierungen liegt maximal bei etwa 250K. Wie zu erwarten war, ist die Simulation mit dem Rusanovfluss am schnellsten und die MUSCL-Verfahren brauchen am längsten. Besonders mit dem Superbeelimiter braucht dieses Verfahren fast 50mal so lange,

wie die Flussberechnung nach Rusanov. Dies liegt daran, wie in Abb. (3.14) zu sehen ist, dass trotz der Limiterfunnktion die MUSCL-Verfahren am stärksten oszillieren. Ebenso

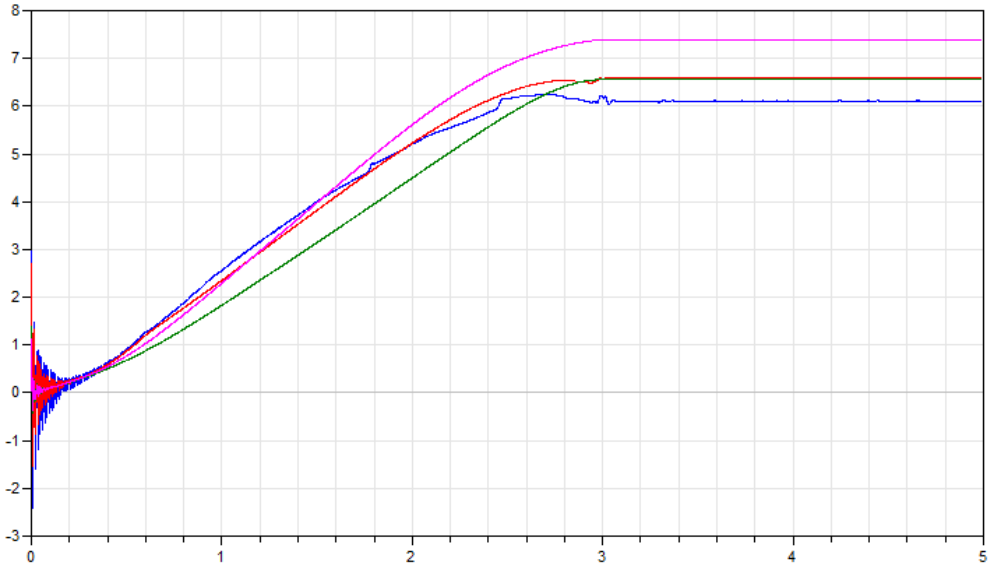

Abbildung 3.14: *Massenstrom der einzelnen Diskretisierungen: Blau: MUSCL-Verfahren mit Superbeelimiter; Rot: MUSCL-Verfahren mit Minbeelimiter; Grün: HLL-Verfahren; Pink: Rusanovfluss. Eine Qualitative Aussage, welche Diskretisierung näher an der Realität liegt, ist hier nicht möglich, da keine realen Daten vorliegen*

ist zu erkennen, dass der Massenstrom, der durch den Rusanovfluss berechnet wurde, relevant größer ist als die nahezu identischen Massenströme aus dem HLL-Verfahren und dem MUSCL-Verfahren mit Minbeelimiter. Dies resultiert auch in eine insgesamt höhere Geschwindigkeit, was für die Interpretation der obigen Ergebnisse von Bedeutung ist. Vor dem Ventil sind die Drücke aller Verfahren nahezu identisch, hinter dem Ventil weichen sie wie die Geschwindigkeiten teils deutlich ab. Es gilt: Niedrigerer Druck spiegelt sich in niedrigere Geschwindigkeit wieder.
Damit lässt sich festhalten, dass, wie aus der Theorie zu erwarten war, für viele oder lange Simulationen sich der Rusanov durch seine geringe Rechenzeit bewährt, wohingegen für ein wenig mehr Rechenaufwand sich das HLL-Verfahren durch eine wahrscheinlich genauere Lösung anbietet. Sowohl die Flussberechnung nach Roe als auch die MUSCL-Verfahren sind in diesem Anwendungsfall ungeeignet.

3.5 Fazit

Es wurde festgestellt, dass beide Ventilvarianten in Modelica nach den ihnen zu Grunde liegenden Formeln plausible Ergebnisse liefern. Dies führt zu einem Teil zu ähnlichen, zum anderen Teil jedoch zu erheblich unterschiedlichen Ergebnissen. Besonders die Ventilcharakteristiken beider Ventile machen hier einen Unterschied aus. Dies gilt sowohl für die Geschwindigkeits- als auch für die Druckkurve in Abhängigkeit vom Öffnungswinkel. Damit kann das eine Model gegenüber dem anderen Model in einem Fall besser sein, im nächsten jedoch auch wieder schlechter. Aufgrund der hier gefundenen Ergebnisse für den Druck ist das diskretisierte Ventil das besser geeignete für das Schmetterlingsventil. Die Resultate für die Geschwindigkeiten lassen keine so deutliche Tendenz für eines der beiden Ventile erkennen. Bei kleinen Ventilöffnungen sind beide Resultate aus Modelica ähnlich derer aus OpenFOAM. Hier liegen die Ergebnisse des diskretisierten Ventil bei Drücken im Anwendungsbereich besonders nahe an den CFD-Simulationen. Bei großen Ventilöffnungen sind die Ergebnisse aus der Flussfunktion näher an den CFD-Simulationen, da hier die Simulation des diskretisierte Ventil deutlich höhere Geschwindigkeiten ergeben. Es gilt jedoch auch zu beachten, dass der verwendete Rusanovfluss stets höhere Geschwindigkeiten berechnet als die anderen Diskretisierungen. Damit ist eine endgültige Entscheidung für eines der Ventile nur mit Realdaten zweifelsfrei möglich, tendenziell scheint jedoch das diskretisierte Ventil in diesem Fall die bessere Wahl zu sein.

4 Zusammenfassung und Ausblick

4.1 Zusammenfassung

Im Zuge dieser Arbeit ist ein diskretisiertes Ventilmodel für die gleichzeitig objektorientierte und gleichungsbasierte Modelierungssprache Dymola Modelica entstanden. Der Hauptpunkt ist hierbei nicht die Erstellung selbst, sondern vielmehr die Validierung des Ventils und die Abgrenzung zu bereits bestehenden Ventilmodellen in Modelica. Das neu erstellte Model soll zur Simulation eines Rohrleitungssystems im Klimasystem eines Flugzeuges Anwendung finden.
Eines dieser bereits vorhandenen Ventilmodelle ist das Model aus der Standardbibliothek von Modelica. Bereits vor dieser Arbeit wurde festgestellt, dass die Standardbibliothek nur unzureichend kompressible Strömungen berechnen kann. Es treten Oszillationen auf, die in der Realität nicht vorkommen. Besonders bei Randwertproblemen verstreicht einiges an simulierter Zeit, bis sich der stationäre Fall eingeschwungen hat. Durch diese Oszillationen ist die Rechenzeit auch verhältnismäßig lang, doch sobald der stationäre Fall eingetreten ist, sind die Ergebnisse für Vergleiche durchaus verwendbar.
Um diesen Oszillationen der Standardbibliothek entgegenzuwirken, ist von Sielemann eine Gasdynamikbibliothek entstanden, die bereits Randbedingungen und Rohrmodelle enthält, welche diskretisiert werden und mit Hilfe der Eulergleichungen und verschiedenen Riemannlösern berechnet werden können. In diese Bibliothek wurden nun Ventilmodelle eingefügt.
Als erstes wurde ein Ventilmodell geschrieben, welches auf der bereits verwendeten Idee der Ausflussfunktion $\psi(p_2/p_1, \kappa)$ basiert. In diesem Model wird der Massenstrom durch das Ventil mit Hilfe der im unterkritischen Druckbereich sowohl vom Druckquotienten als auch vom Isentropenexponent und im überkritischen Druckbereich nur vom Isentropenexponent abhängigen Ausflussfunktion berechnet. Der Massenstrom wird dabei vom Querschnitt des Ventilhalses, welcher mit $(1 - \cos(\varphi)) \cdot A$ berechnet wird, reguliert. Die benötigten Drücke vor und hinter dem Ventil werden durch die Rohre vorgegeben. Auch wird das Ventil als adiabat angenommen, sodass die Enthalpie über dem Ventil konstant bleibt und damit für ein ideales Gas auch die Temperatur. Das Ventil selber gibt dann die aus dem Massenstrom und aus dem vollen Ventilquerschnitt ermittelte Geschwindigkeit an die angeschlossenen Rohre zurück.
Ebenso wurde das diskretisierte Ventilmodell geschrieben, welches gleich den Rohren auf den Eulergleichungen aufbaut. Dazu wurde das Ventil gleich den Rohren in Finite Volumen unterteilt, so dass mit den bereits vorhandenen Riemannlöseren die Berechnung des

Fluids übernommen werden konnte. Der Querschnitt des Ventilhalses, der hier wiederum die regelbare Größe des Ventils darstellt, wurde gleich wie bei dem oben genannten Ventil mit Ausflussfunktion berechnet.
Zusätzlich zu den beiden Ventilen in Modelica sind mit Hilfe des CFD-Programms OpenFOAM Simulationen entstanden, welche als Referenzwerte für die Modelica-Simulationen dienen sollen, da nun hier das erste Mal eine feste Geometrie vorgegeben wurde. Diese Simulation rechnen im 2D mit einem Gitter, welches das im Flugzeug verbaute Schmetterlingsventil durch seine projizierte Fläche im Querschnitt des Rohres darstellt.
Als Vergleichswerte für die Ventile wurde der Massenstrom, resp. die Geschwindigkeit, da diese von Relevanz für das Klimasystem sind, sowie der Druck herangezogen. Bei dem Vergleich zwischen der Standardbibliothek und dem Ventil mit der Ausflussfunktion stellte sich heraus, dass sich die Ventile in diesen beiden Punkten nahezu identisch verhalten. Es stellte sich erst ein Unterschied bei hohen Druckdifferenzen ein, bei denen das Ventil aus der Standardbibliothek sich nicht mehr wie in der Dokumentation beschreiben verhielt.
Auch die CFD-Simulationen und das diskretisierte Ventil verhielten sich sehr ähnlich, besonders die Druckkurven. Die Geschwindigkeitscharakteristiken beider Ventile wiesen eine S-Form gleich der Charakteristik eines Schmetterlingsventils auf, die Beträge wichen jedoch voneinander ab: Das Fluid des diskretisierten Ventils war bei einem Druckgefälle von 1.1bar auf 1bar bei kleiner Ventilöffnung um den Faktor 0.3 geringer, bei großer Ventilöffnung um den Faktor 2 bis 2.3 größer. Die Ausflussfunktion berechnete generell höhere Geschwindigkeiten, die insgesamt näher an den Ergebnissen aus OpenFOAM lagen.
Bei einem höheren Druckgefälle (5.5bar auf 3.5bar) liegen alle Werte bei kleinen Ventilöffnungen sehr nahe beieinander, erst bei größeren (ca. 60°) liegt die Geschwindigkeit der diskretisierten Berechnung um etwa den Faktor 2 über den Ergebnissen aus OpenFOAM, die Ausflussfunktion liegt hier um den Faktor 1.3 drüber.
Die Resultate beider selbstgeschriebener Ventile in Modelica verglichen untereinander waren sehr unterschiedlich. Während sich der Druck des Ventils mit Ausflussfunktion während des Öffnungsvorganges nur stark begrenzt annähert, so dass bei vollständig geöffnetem Ventil weiterhin fast der gesamte Druck über dem Ventil abfällt, steigt die Druckdifferenz des diskretisierten Ventil zunächst an, um dann bei vollständig geöffnetem Ventil auf ein Minimum zu sinken, so dass der gesamte Druck über das Rohrsystem linear abfällt. Die Geschwindigkeit, bzw. der Massenstrom, verhält sich bei erst genanntem Ventil im Öffnungsvorgang proportional zur Querschnittsfläche des Ventilhalses mit zunehmender Steigung bei zunehmender Druckdifferenz, wohingegen das diskretisierte Ventil die besagte S-Kurve beschreibt, welche sich mit zunehmender Druckdifferenz nach links, also zu kleineren Winkeln, verschiebt.
Da keine realen Messdaten vorliegen, ist es schwer zu entscheiden, welches Model hier besser das Rohrleitungssystem beschreibt. Aufgrund der gefundenen Ergebnisse, besonders der Ventilcharakteristiken und der Ähnlichkeit zu den CFD-Simulationen, ist das diskretisierte Ventil in diesem Fall wahrscheinlich die bessere Wahl.

4.2 Ausblick

Jedwedes erstellte Ventil ist hier ohne den Faktor Reibung entstanden. Während die Rohre selbst bereits mit Reibung rechnen (Aufruf der Funktion zur Reibungsberechnung aus der Standardbibliothek), muss dieser Punkt u.a. bei dem diskretisierten Ventil noch ergänzt werden. Auch die Reibung wird im Quellterm der Impulsgleichung der Eulergleichungen eingefügt werden müssen. Allerdings wird die Reibung an der Geometrie den geringeren Anteil ausmachen. Hier wird besonderes die turbulente Strömung hinter dem Ventil ausschlaggebend sein, da hier, wie die Simulationen in OpenFOAM zeigen, Rückströmungen entstehen können. Hier wird der Faktor der Dissipationsenergie nicht zu vernachlässigen sein.

Um die CFD-Simulationen zu verbessern ist es sinnvoll, statt mit der projizierten Fläche des Ventils mit der tatsächlichen Geometrie des Schmetterlingsventil zu rechnen, da dies die starke Rückströmung hinter dem Ventil verhindern würde. Ebenso muss das Turbulenzmodel genauer betrachtet werden, da hier die größte Unsicherheit vermutet wird.

Ferner kann es von Vorteil sein, weitere Ventilcharakteristiken zu implementieren, ähnlich wie es in der Standardbibliothek bereits geschehen ist. Aktuell wird im diskretisierten Ventil nur die Funktion $(1 - \cos(\varphi))$ benutzt. Dies kann so erweitert werden, dass aus einer größeren Anzahl verschiedener Ventile ausgewählt werden kann. Diese Implementierung muss am Querschnitt des Ventilhalses geschehen. In diesem Zusammenhang muss dann auch der Öffnungsgard des Ventils adaptiv gestaltet werden, da bei einigen Ventilen, z.B. Absperrventil, kein Winkel zwischen 0 ° und 90 ° die Öffnung bestimmt. Auch ist es möglich, bei höherrangigen Diskretisierungen einen Übergang zwischen Rohr und Ventilhals zu implementieren. Damit würde auch eine Geometrie des Ventils beschreiben werden.

Schließlich ist es jedoch unvermeidlich, die Ventilmodelle mit realen Daten aus dem Flugzeug oder ggf. anderen realen Ventilen zu vergleichen. Erst aus diesen Werten kann man die tatsächliche Güte der Simulationen in Modelica bestimmen.

Literaturverzeichnis

[1] Batchelor, George K.: *An introduction to fluid dynamics*. Cambridge university press, 2000

[2] Davis, SF: Simplified second-order Godunov-type methods. In: *SIAM Journal on Scientific and Statistical Computing* 9 (1988), Nr. 3, S. 445–473

[3] Einfeldt, Bernd: On Godunov-type methods for gas dynamics. In: *SIAM Journal on Numerical Analysis* 25 (1988), Nr. 2, S. 294–318

[4] Hanff, Jochen: *Optimierung von Knotenkoordinaten eingeschränkter Triangulation mit evolutionären Algorithmen*, Technische Universität Berlin, Diplomarbeit, 1998

[5] Harten, Amiram ; Lax, Peter D. ; Leer, Bram v.: On upstream differencing and Godunov-type schemes for hyperbolic conservation laws. In: *SIAM review* 25 (1983), Nr. 1, S. 35–61

[6] Krämer, Ewald: *Strömungslehre 2 Skript*. IAG Universität Stuttgart, 2013

[7] Krämer, Ewald: *Strömungslerhe 1 Skript*. IAG Universität Stuttgart, 2013

[8] Lax, Peter D.: Weak solutions of nonlinear hyperbolic equations and their numerical computation. In: *Communications on pure and applied mathematics* 7 (1954), Nr. 1, S. 159–193

[9] Munz, Claus-Dieter: *Numerische Simulation in der Luft- und Raumfahrt Teil 2*. IAG Universität Stuttgart, 2013

[10] OpenModelica: *Modelica Documetation*. `https://build.openmodelica.org/Documentation/`, 28.08.2014

[11] Roe, Philip L.: Approximate Riemann solvers, parameter vectors, and difference schemes. In: *Journal of computational physics* 43 (1981), Nr. 2, S. 357–372

[12] Rusanov, Vladimir V.: *Calculation of interaction of non-steady shock waves with obstacles*. NRC, Division of Mechanical Engineering, 1962

[13] Sielemann, Michael: *Device-Oriented Modeling and Simulation in Aircraft Energy Systems Design*. Dr. Hut, 2012

[14] Smuts, Jacques: *Butterfly Valves and Control Performance.* `http://blog.opticontrols.com/wp-content/uploads/2012/02/Butterfly-Valve-Characteristic.png`, 27.08.2014

[15] Sod, Gary A.: A survey of several finite difference methods for systems of nonlinear hyperbolic conservation laws. In: *Journal of computational physics* 27 (1978), Nr. 1, S. 1–31

[16] Toro, Eleuterio F.: *Riemann Solvers and Numerical Methods for Fluid Dynamics - A Practical Introduction.* 2nd Edition. Springer Verlag, 1999

[17] Van Leer, Bram: On the relation between the upwind-differencing schemes of Godunov, Engquist-Osher and Roe. In: *SIAM Journal on Scientific and Statistical Computing* 5 (1984), Nr. 1, S. 1–20

Anhang

OpenFOAM-Dateien

Listing A.1: *BlockMeshDict-Datei für das Ventil mit 10% Öffnung*

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      blockMeshDict;
}
convertToMeters 1;

vertices
(
  //Block 1: left pipe
    (0  0       0) //Vertex 0
    (1  0       0) //Vertex 1
    (1  0.005   0) //Vertex 2
    (0  0.005   0) //Vertex 3
    (0  0       0.05) //Vertex 4
    (1  0       0.05) //Vertex 5
    (1  0.005   0.05) //Vertex 6
    (0  0.005   0.05) //Vertex 7

  //Block 2: lower part of valve
    //Vertex 1
    (1.01       0       0) //Vertex 8
    (1.01       0.005   0) //Vertex 9
    //Vertex 2
    (1          0       0.0025) //Vertex 10
    (1.01       0       0.0025) //Vertex 11
    (1.01       0.005   0.0025) //Vertex 12
    (1          0.005   0.0025) //Vertex 13
```

```
  //Block 3: upper part of valve
    (1            0        0.0475) //Vertex 14
    (1.01         0        0.0475) //Vertex 15
    (1.01         0.005    0.0475) //Vertex 16
    (1            0.005    0.0475) //Vertex 17
    //Vertex 5
    (1.01         0        0.05) //Vertex 18
    (1.01         0.005    0.05) //Vertex 19
    //Vertex 6

  //Block 4: right pipe
    //Vertex 8
    (2.01         0        0) //Vertex 20
    (2.01         0.005    0) //Vertex 21
    //Vertex 9
    //Vertex 18
    (2.01         0        0.05) //Vertex 22
    (2.01         0.005    0.05) //Vertex 23
    //Vertex 19
);

blocks
(
    hex (0 1 2 3 4 5 6 7) (40 1 40) simpleGrading (0.2 1 1)
        //left pipe
    hex (1 8 9 2 10 11 12 13) (2 1 2) simpleGrading (1 1 1)
        //lower part of valve
    hex (14 15 16 17 5 18 19 6) (2 1 2) simpleGrading (1 1
       1) //upper part of valve
    hex (8 20 21 9 18 22 23 19) (40 1 40) simpleGrading (5
       1 1) //right pipe
);
edges
(
);
boundary
(
    Inlet
    {
        type patch;
```

```
    faces
    (
        (0 4 7 3)
    );
}
Outlet
{
    type patch;
    faces
    (
        (20 21 23 22)
    );
}
Wall
{
    type wall;
    faces
    (
        //Top
        (4 5 6 7)
        (5 8 19 6)
        (18 22 23 19)
        //Bottom
        (0 3 2 1)
        (1 2 9 8)
        (8 9 21 20)
        //Top Block2
        (10 11 12 13)
        //Bottom Block3
        (14 17 16 15)
    );
}
FrontAndBack
{
    type empty;
    faces
    (
        //Front
        (0 1 5 4)
        (1 8 11 10)
        (14 15 18 5)
```

```
            (8 20 22 18)
            //Back
            (3 2 6 7)
            (2 9 12 13)
            (17 16 19 6)
            (9 21 23 19)
        );
    }
    InternalFaceR1 //outlet left pipe
    {
        type wall;
        faces
        (
            (1 2 6 5)
        );
    }
    InternalFaceL2 //inlet right pipe
    {
        type wall;
        faces
        (
            (8 18 19 9)
        );
    }
    InternalFaceL1 //inlet valve
    {
        type patch;
        faces
        (
            (1 10 13 2)
            (14 5 6 17)
        );
    }
    InternalFaceR2 //outlet valve
    {
        type patch;
        faces
        (
            (8 9 12 11)
            (15 16 19 18)
        );
```

```
    }
);
mergePatchPairs
(
        //merge inlet und outlet of pipe and valve
    (InternalFaceR1 InternalFaceL1)
    (InternalFaceL2 InternalFaceR2)
);
```

Listing A.2: *turbulenceProperties-Datei aus dem Ordner constant*

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      turbulenceProperties;
}

simulationType  RASModel;
```

Listing A.3: *RASProperties-Datei aus dem Ordner constant*

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      RASProperties;
}

RASModel        LaunderSharmaKE;

turbulence      on;

printCoeffs     on;
```

Listing A.4: *thermophysicalProperties-Datei aus dem Ordner constant*

```
FoamFile
{
```

```
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      thermophysicalProperties;
}

thermoType
{
    type            hePsiThermo;
    mixture         pureMixture;
    transport       const;
    thermo          hConst;
    equationOfState perfectGas;
    specie          specie;
    energy          sensibleInternalEnergy;
}

mixture
{
    specie
    {
        nMoles          1;
        molWeight       28.9;
    }
    thermodynamics
    {
        Cp              1005;
        Hf              0;
    }
    transport
    {
        mu              1.8e-05;
        Pr              0.7;
    }
}
```

Listing A.5: *transportProperties-Datei aus dem Ordner constant*

```
FoamFile
{
    version     2.0;
```

```
    format      ascii;
    class       dictionary;
    location    "constant";
    object      transportProperties;
}

nu              nu [ 0 2 -1 0 0 0 0 ] 0.01;
```

Listing A.6: *controlDict-Datei aus dem Ordner sytem*

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      controlDict;
}

application     sonicFoam;

startFrom       startTime;

startTime       0;

stopAt          endTime;

endTime         1;

deltaT          10e-06;

writeControl    runTime;

writeInterval   0.001;

purgeWrite      0;

writeFormat     ascii;

writePrecision  6;

writeCompression off;
```

```
timeFormat      general;

timePrecision   6;

runTimeModifiable true;
```

Listing A.7: *fvSchemes-Datei aus dem Ordner sytem*

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSchemes;
}

ddtSchemes
{
    default         Euler;
}

gradSchemes
{
    default         Gauss linear;
}

divSchemes
{
    default         none;
    div(phi,U)      Gauss limitedLinearV 1;
    div(phi,e)      Gauss limitedLinear 1;
    div(phid,p)     Gauss limitedLinear 1;
    div(phi,K)      Gauss limitedLinear 1;
    div(phiv,p)     Gauss limitedLinear 1;
    div(phi,k)      Gauss upwind;
    div(phi,epsilon) Gauss upwind;
    div((muEff*dev2(T(grad(U))))) Gauss linear;
}

laplacianSchemes
```

```
{
    default          Gauss linear corrected;
}

interpolationSchemes
{
    default          linear;
}

snGradSchemes
{
    default          corrected;
}

fluxRequired
{
    default          no;
    p                ;
}
```

Listing A.8: *fvSolution-Datei aus dem Ordner sytem*

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSolution;
}

solvers
{
    "rho.*"
    {
        solver          diagonal;
    }

    "p.*"
    {
        solver          smoothSolver;
        smoother        symGaussSeidel;
```

```
        tolerance       1e-08;
        relTol          0;
    }

    "(U|e|R).*"
    {
        $p;
        tolerance       1e-05;
    }

    "(k|epsilon).*"
    {
        $p;
        tolerance       1e-08;
    }
}

PIMPLE
{
    nOuterCorrectors     2;
    nCorrectors          1;
    nNonOrthogonalCorrectors 0;
}
```

Moedlica-Dateien

Listing A.9: *Quelltext des diskretisierten Ventil*

```
model valveDiscretization
  replaceable package Medium =
     GasDynamics3.Media.SimpleAirGasDynamics
       constrainedby
          Modelica.Media.Interfaces.PartialMedium "Medium
          model"
       annotation(choicesAllMatching=true);

  replaceable package Discretization =
      GasDynamics3.Discretizations.LocalLaxFriedrichs (
        redeclare package Medium
       = Medium) constrainedby
    GasDynamics3.Discretizations.Partial.PartialDiscretization
       (redeclare
      package Medium = Medium) "Discretization"
    annotation(choicesAllMatching=true);

    parameter Modelica.SIunits.Length D = 0.1 "Diameter of
       full open valve";
    Modelica.SIunits.Area[n+1] A "Surfacearea of each
       volumina";

    final parameter Integer n =
       Discretization.stencilLength
    "Number of volumina";

    Medium.BaseProperties medium[n] "medium properties of
       each volumina";
    Modelica.SIunits.Velocity v[n] "velocity of each
       volumina";
    Real[n,3] U "conserved variables of euler-equation";
    Real[n+1,3] sideFlux "sideFlux of euler-equation";
    Real[n,3] flux "Flux of euler-equation";
    Real[n,3] sourceTerm "sourceterm of the euler-equation"
       ;

    Medium.ThermodynamicState[n+1, n] states
```

```
    "array of thermodynamic states for the flux-calculation
       ";

    Modelica.SIunits.Velocity[n+1, n] vs
    "array of velocities for the flux-calculation";

    parameter Modelica.SIunits.Velocity v_init = 0 "initial
        velocity";
    parameter Modelica.SIunits.Temperature T_init = 300 "
       initial temperature";
    parameter Modelica.SIunits.AbsolutePressure p_init = 1
       e5 "initial pressure";

  Interfaces.Stencil_a stencil_a(redeclare package Medium =
      Medium, redeclare
      package Discretization = Discretization) annotation (
         Placement(transformation(extent
         ={{-112,-10},{-90,10}})));
  Interfaces.Stencil_b stencil_b(redeclare package Medium =
      Medium, redeclare
      package Discretization = Discretization) annotation (
         Placement(transformation(extent
         ={{90,-10},{110,10}})));
  Modelica.Blocks.Interfaces.RealInput phi
    "Valveposition in deg: 0 = closed, 90 = full open"
                                          annotation (
      Placement(transformation(
       extent={{-20,-20},{20,20}},
       rotation=90,
       origin={0,-60})));

equation
  for i in 1:n loop
    //conserved variables of euler-equation
    U[i,1] = medium[i].d;
    U[i,2] = medium[i].d*v[i];
    U[i,3] = medium[i].d*(0.5*v[i]*v[i] + medium[i].u);

    //In cause of the smaller Area of the valve, a force
       has an effect on the fluid
    sourceTerm[i,:] = {0, medium[i].p*(A[i+1]-A[i]), 0};
```

```
  //intracellular flux equals the difference of the
     intercellular flux
  flux[i,:] = sideFlux[i+1,:] - sideFlux[i,:];

  //Euler-equtaion itself
  der(U[i,:]) = (-flux[i,:] + sourceTerm[i,:])/((A[i]+A[i
     +1])/2*0.1);  //Volume:(Ai + Ai+1)/2*dx; dx = 0.1
end for;

//Calculation of states, Areas and velocities for the
   sideFlux-calculation
for i in 1:Discretization.halfStencilLength loop
  //1,2,3...
  states[i,:] = cat(1,stencil_a.state_b[i:end], medium[1:
     i + Discretization.halfStencilLength - 1].state);
  //n+1, n, n-1...
  states[n+2-i,:] = cat(1,medium[
     Discretization.halfStencilLength + 2 - i:end].state,
      stencil_b.state_a[1:end + 1 - i]);

  A[i] = Modelica.Constants.pi/4*D^2;
  A[n+2-i] = Modelica.Constants.pi/4*D^2;

  vs[i,:] = cat(1,stencil_a.v_b[i:end], v[1:i +
     Discretization.halfStencilLength - 1]);
  vs[n+2-i,:] = cat(1,v[Discretization.halfStencilLength
     + 2 - i:end], stencil_b.v_a[1:end + 1 - i]);

end for;

//calculation of states, Area and velocities for the
   sideflux-calculation between the middle volumes
states[Discretization.halfStencilLength + 1,:] =
   medium.state;
A[Discretization.halfStencilLength + 1] =
   Modelica.Constants.pi/4*D^2*(1 - abs(Modelica.Math.cos
   (phi/180*Modelica.Constants.pi))); //(1-cos)
   entspricht Elipse; (1-cos^2) entspricht Kreis als
   Ventilquerschnitt
vs[Discretization.halfStencilLength + 1,:] = v;
```

```
  //calculation of sideflux
  for i in 1:n+1 loop
    sideFlux[i,:] = A[i]*Discretization.flux(states[i,:],
       vs[i,:],{0.1*j for j in 1:
       Discretization.stencilLength + 1});
  end for;

  // Prescribe connector variables
  stencil_a.state_a = {medium[i].state for i in 1:
     Discretization.halfStencilLength};
  stencil_a.v_a = {v[i] for i in 1:
     Discretization.halfStencilLength};
  stencil_a.x_side_a = {0.1*i for i in 1:
     Discretization.halfStencilLength};

  stencil_b.state_b = {medium[n+1-i].state for i in 1:
     Discretization.halfStencilLength};
  stencil_b.v_b = {v[n+1-i] for i in 1:
     Discretization.halfStencilLength};
  stencil_b.x_side_b = {0.1*i for i in 1:
     Discretization.halfStencilLength};

initial equation
  //of the Riemann-Problem
  for i in 1:n loop
    v[i] = v_init;
    medium[i].p = p_init;
    medium[i].T = T_init;
  end for;

  annotation (Diagram(coordinateSystem(preserveAspectRatio=
     false, extent={{-100,-100},{100,100}}), graphics),
              Icon(coordinateSystem(preserveAspectRatio=
                 false, extent={{-100,-100},
            {100,100}}), graphics={
        Polygon(
          points={{-40,20},{-40,-20},{0,0},{-40,20}},
          lineColor={0,0,255},
          smooth=Smooth.None),
        Polygon(
```

```
          points={{0,0},{40,20},{40,-20},{0,0}},
          lineColor={0,0,255},
          smooth=Smooth.None),
        Line(
          points={{-90,0},{-40,0}},
          color={0,0,255},
          smooth=Smooth.None),
        Line(
          points={{40,0},{90,0}},
          color={0,0,255},
          smooth=Smooth.None),
        Text(
          extent={{-100,100},{100,60}},
          lineColor={0,0,0},
          textString="%name")}));
end valveDiscretization;
```

Listing A.10: *Quelltext des Ventils mit Ausflussfunktion*

```
model valveFlowFunciton
  replaceable package Medium =
     GasDynamics3.Media.SimpleAirGasDynamics
    constrainedby Modelica.Media.Interfaces.PartialMedium "
       Medium model"
    annotation(choicesAllMatching=true);

  replaceable package Discretization =
      GasDynamics3.Discretizations.LocalLaxFriedrichs(
         redeclare package Medium
       = Medium) constrainedby
    GasDynamics3.Discretizations.Partial.PartialDiscretization
       (redeclare
      package Medium = Medium) "Discretization"
    annotation(choicesAllMatching=true);

    parameter Modelica.SIunits.Length D = 0.01 "Diameter of
        full open Valve" annotation (Dialog( enable = not
       useKv));

    Real k = Medium.isentropicExponent(mediumA.state) "
       Isentropic Exponent";
```

```
    Modelica.SIunits.Area A
    "crosssection of valve full open resp. crosssection of
       pipes";
    Modelica.SIunits.Area Av "crosssection of valve";

    Medium.BaseProperties mediumA(preferredMediumStates =
       true)
    "medium state before the valve";
    Medium.BaseProperties mediumB(preferredMediumStates =
       true)
    "medium state after the valve";
    Modelica.SIunits.MassFlowRate m_flow "Massflow of the
       Valve";

    Modelica.SIunits.Velocity v_in "Inflowvelocity";
    Modelica.SIunits.Velocity v_out "Outflowvelocity";

Interfaces.Stencil_a stencil_a(redeclare package Medium =
   Medium, redeclare
      package Discretization = Discretization)
                                annotation (Placement(
                                   transformation(extent
                                   ={{-110,
          -10},{-90,10}}), iconTransformation(extent
             ={{-110,-10},{-90,10}})));
  Interfaces.Stencil_b stencil_b(redeclare package Medium =
      Medium, redeclare
      package Discretization = Discretization)
                                annotation (Placement(
                                   transformation(extent
                                   ={{90,
          -10},{110,10}}), iconTransformation(extent
             ={{90,-10},{110,10}})));

  Modelica.Blocks.Interfaces.RealInput phi
    "Valveposition in deg: 0 = closed, 90 = full open"
                                          annotation (
      Placement(transformation(
       extent={{-20,-20},{20,20}},
       rotation=90,
       origin={0,-40})));
```

```
equation
   A =  Modelica.Constants.pi/4*D^2;
   Av = A*(1-abs(Modelica.Math.cos(phi*
      Modelica.Constants.pi/180)));

   //incoming values
   mediumB.p = stencil_b.state_a[
      Discretization.halfStencilLength].p; //->1
   mediumA.p = stencil_a.state_b[
      Discretization.halfStencilLength].p; //->end
   mediumA.T = stencil_a.state_b[
      Discretization.halfStencilLength].T; //->end

   //Adiabatic valve -> no loss of enthalpy
   mediumA.h = mediumB.h;

   //Konti-Equation
   v_in = m_flow/(A*mediumA.d);
   v_out = m_flow/(A*mediumB.d);

   //m_flow through valve
   m_flow = Av*outflowFunction(min(mediumB.p,mediumA.p)/max
      (mediumA.p,mediumB.p), k)*sqrt(2*max(
      mediumA.p,mediumB.p)*max(mediumA.d,mediumB.d))*sign(
      mediumA.p-mediumB.p);

   // Prescribe connector variables
   stencil_a.state_a =  {mediumA.state for i in 1:
      Discretization.halfStencilLength};
   stencil_b.state_b =  {mediumB.state for i in 1:
      Discretization.halfStencilLength};

   stencil_a.v_a = {v_in for i in 1:
      Discretization.halfStencilLength};
   stencil_b.v_b = {v_out for i in 1:
      Discretization.halfStencilLength};

   stencil_a.x_side_a = {0.1*i for i in 1:
      Discretization.halfStencilLength};
   stencil_b.x_side_b = {0.1*i for i in 1:
```

```
      Discretization.halfStencilLength};

    annotation (Icon(coordinateSystem(preserveAspectRatio=
       false, extent={{-100,-100},
            {100,100}}), graphics={
        Polygon(
          points={{-40,20},{-40,-20},{0,0},{-40,20}},
          lineColor={0,0,255},
          smooth=Smooth.None),
        Polygon(
          points={{0,0},{40,20},{40,-20},{0,0}},
          lineColor={0,0,255},
          smooth=Smooth.None),
        Line(
          points={{-90,0},{-40,0}},
          color={0,0,255},
          smooth=Smooth.None),
        Line(
          points={{40,0},{90,0}},
          color={0,0,255},
          smooth=Smooth.None),
        Text(
          extent={{-100,100},{100,60}},
          lineColor={0,0,0},
          textString="%name")}),
    experiment(
      StopTime=5,
      __Dymola_NumberOfIntervals=5000,
      __Dymola_Algorithm="Dassl"),
    __Dymola_experimentSetupOutput);
end valveFlowFunciton;
```