Universität zu Köln

JÜLICH
FORSCHUNGSZENTRUM

Simulations of the Atomic Beam Transport in an Atomic Beam Source under the Influence of Spin-Selective Sextupole Magnets

Inaugural-Dissertation

zur

Erlangung des Doktorgrades

der Mathematisch-Naturwissenschaftlichen Fakultät

der Universität zu Köln

vorgelegt von

Martin Gaißer

aus Radolfzell

Jülich, 2013

Berichterstatter: Prof. Dr. H. Ströher

Prof. Dr. J. Jolie

Tag der mündlichen Prüfung: 22.1.2014

Abstract

Polarized internal gas targets are important tools in spin-physics experiments at particle accelerators. For many experiments it is imperative to get the highest possible target density. Research in recent decades, however, led only to marginal increase of the intensity of atomic beam sources (ABS), which constitute the main factor in increasing the target density. The main problem, hindering further improvement, is the complexity of the processes in the ABS, which prevents a complete description of all effects. A particularly big challenge is the description of the change in flow regime from laminar to molecular during the formation of the atomic beam by the nozzle. Now, with ever increasing computer power, it becomes possible to simulate all the processes in a complete ABS and to use algorithms to optimize the device.

The present thesis summarizes the important effects in polarized internal gas targets and describes the implementation of a new computer program, which was interfaced with an optimization algorithm. The program is based on OpenFOAM 1.7.1 and uses the Direct Simulation Monte Carlo (DSMC) method to simulate the gas flow. Besides a generic interface for magnetic fields which act on the magnetic moments of the atoms, many important effects like recombination and spin-exchange collisions are included in the program. Furthermore, a generic framework for optimization is implemented. It can be used to optimize very different problems with different optimization algorithms. Currently, Adaptive Simulated Annealing (ASA) is implemented together with the capability to handle the optimization of an ABS. All parts of the program were tested and results are provided. Although the execution time of a simulation run is still very long, the new program might become a valuable tool for simulating and optimizing polarized internal gas targets. This will lead to better statistics in experiments and will enable certain new experiments.

Zusammenfassung

Polarisierte interne Gastargets sind wichtige Instrumente in Spinphysik-Experimenten an Teilchenbeschleunigern. Viele Experimente benötigen zwingend eine höchstmögliche Targetdichte. Die Forschung während der letzten Jahrzehnte brachte allerdings nur geringe Fortschritte bei der Erhöhung der Intensität von Atomstrahlquellen (ABS), die den größten Einfluss auf die Targetdichte haben. Das Hauptproblem, das weitere Verbesserungen verhindert, besteht in der Komplexität der Prozesse in der ABS, welche eine komplette Beschreibung aller Effekte verhindert. Eine besondere Herausforderung stellt der Übergang vom laminaren zum molekularen Flussbereich während der Erzeugung des Atomstrahls in der Düse dar. Mit stetig steigender Computerleistung wird es heutzutage möglich, alle Abläufe in einer kompletten ABS zu simulieren und einen Algorithmus zur Optimierung der Quelle zu verwenden.

Die vorliegende Doktorarbeit fasst alle wichtigen Effekte in polarisierten internen Gastargets zusammen und beschreibt die Implementierung eines neuen Computerprogramms, welches mit einem Optimierungsalgorithmus verbunden wurde. Das Programm baut auf OpenFOAM 1.7.1 auf und nutzt die Direct Simulation Monte Carlo (DSMC) Methode um den Gasfluss zu simulieren. Neben einer generischen Schnittstelle für Magnetfelder, die auf die magnetischen Momente der Atome wirken, wurden viele weitere Effekte wie Rekombination und Spinaustausch-Stöße in das Programm miteingebaut. Desweiteren wurde ein generisches Rahmenwerk für die Optimierung entwickelt und implementiert, mit dessen Hilfe sehr verschiedene Probleme mit verschiedenen Algorithmen optimiert werden können. Zur Zeit ist Adaptive Simulated Annealing (ASA) zusammen mit der Fähigkeit zur Optimierung einer ABS implementiert. Alle Programmteile wurden getestet und Ergebnisse werden präsentiert. Obwohl die Ausführung des Programms immer noch sehr viel Zeit in Anspruch nimmt, könnte das Programm ein wertvolles Werkzeug zur Simulation und Optimierung von internen polarisierten Gastargets werden. Dies wird zu geringeren statistischen Fehlern in Experimenten führen und bestimmte neue Experimente erst ermöglichen.

Contents

1 Introduction

Experiments with spin-polarized protons and antiprotons allow to measure a multitude of parameters which are not accessible in collisions of unpolarized particles. For example, the Polarized Antiproton eXperiments ($\mathcal{PAX}$) Collaboration proposed to measure the transversity distribution of the valence quarks of the proton which is only directly accessible in collisions of transversely polarized protons and antiprotons [1]. Beams and targets of polarized protons and deuterons are available at various facilities and are usually produced in polarized atomic beam sources (ABS) [2], where the hyperfine splitting of hydrogen or deuterium states in magnetic fields is exploited. The only low-intensity low-quality polarized antiproton beam however was produced in the decay of $\overline{\Lambda}$-hyperons at Fermilab [3]. As a method to produce polarized beams of higher intensity, spin-filtering was proposed by Csonka in 1968 [4] and was first measured with protons by the FILTEX group [5] at the Test Storage Ring (TSR) in Heidelberg [6]. Recently, measurements with higher beam energy and other machine parameters were performed by the PAX group [7, 8] at COSY [9] in Jülich to test the theoretical predictions [10, 11]. Although the polarization-buildup rate and the final beam polarization for protons is small, this might be different for antiprotons, where some parameters to calculate the relevant cross sections are unknown. However, beam polarizations of 15-20% seem feasible after filtering for two beam lifetimes at a dedicated polarizer ring [12]. It was proposed by the PAX group to measure the cross section of double polarized $p\bar{p}$ scattering at the CERN Antiproton Decelerator (AD) ring.

One way to increase the polarization-buildup rate is to increase the target thickness, since it enters linearly in the rate. A high target thickness would also be preferable in many other experiments. To this end, a lot of effort was put into increasing the ABS output and increasing the areal target density in the gas-storage cells (e.g. [13]). Neither experimental nor theoretical work (e.g. [14, 15, 16]), however, led to major advances in recent decades. This may be due to principle limitations or due to the multitude of different effects, which could not be modeled all at once and such prevent a complete understanding of the atomic beam source. A hint for the latter one is that all recent sources have similar intensity, while only the beam from the RHIC source is by roughly 50% more intense than those from all others (compare Fig. 1.1). The decisive reason for this difference finally is not known.

The aim of the present work therefore was to develop a versatile computer program which can simulate a complete ABS (and more generally the whole polarized target including the ABS and the gas-storage cell) with all relevant effects. This should help

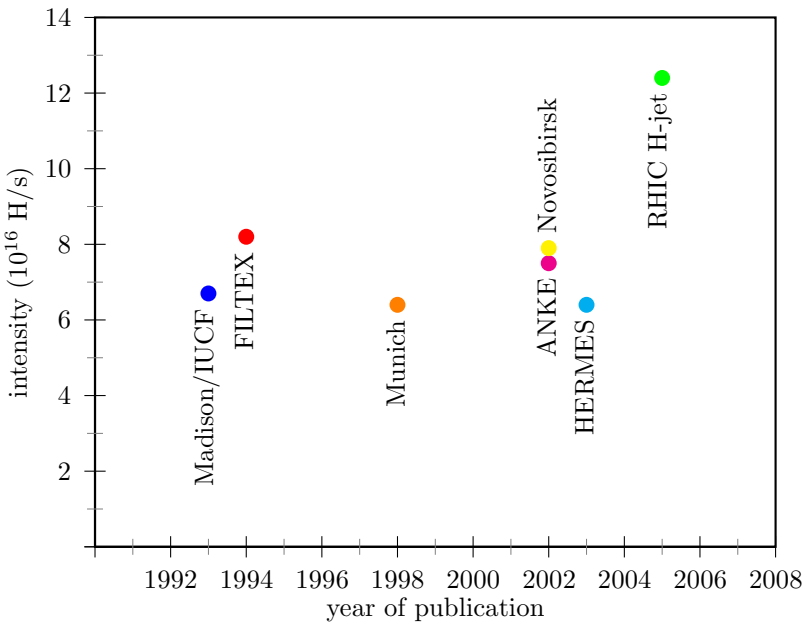

Figure 1.1 – *Intensity (in two hyperfine states) of different polarized hydrogen atomic beam sources against year of publication, References: Madison/IUCF [17], FILTEX [18], Munich [19], Novosibirsk [20], ANKE [21, 22], HERMES [23], RHIC H-jet [24, 25]*

to better understand the interplay of the different ABS components. Further, an optimization algorithm was implemented that automatically optimizes the atomic beam source to maximize the output of spin-polarized hydrogen or deuterium atoms. The theoretical foundations and the principle of Stern-Gerlach type atomic beam sources are explained in Chap. 2. The Direct Simulation Monte Carlo (DSMC) method was applied to simulate the gas flow. It is described in Chap. 3. Chapter 4 gives a short introduction to the program, on which the current work is based and describes in detail the solver *dsmcFoam* that was extended during the work. The changes made to the solver are detailed in Chap. 5. Tests of the new parts of the program were performed and some results are shown in Chap. 6. The optimization algorithm is described in Chap. 7 and a new solver for the optimization procedure is described in Sec. 7.2. Finally, information about the setup of simulations together with examples of configuration files can be found in the appendices.

2 Polarized Internal Gas Targets

2.1 Hyperfine Structure of Hydrogen and Deuterium

Angular momenta of electrons and nuclei in atoms produce magnetic fields which in turn retroact on the magnetic momenta. This leads to different energy levels of the ground state in hydrogen atoms, where electron (total angular momentum $J = 1/2$) and proton spin ($I = 1/2$) can couple in various relative orientations. If an additional external magnetic field is present, it also affects the spin coupling. The effect of a strong B field on the spins overwhelms the coupling effect and each magnetic moment couples individually to the external field. The eigenstates of the corresponding Hamiltonian can be described with the projection of the spins $|m_{\mathrm{S}}, m_{\mathrm{I}}\rangle$ onto the direction of the external field in the high B field case, whereas in the low-field case they are described by the total angular momentum $F = J + I$ and its projection m_{F}. A description of the intermediate case in an external magnetic field $\vec{B} = B_{\mathrm{z}}\vec{e}_{\mathrm{z}}$ can be obtained by calculating the $(2J + 1)(2I + 1)$ different eigenvalues and eigenvectors of the Hamiltonian $\mathcal{H}$. For the ground state of hydrogen and deuterium the angular momentum of the electron J is equal to the spin $S = 1/2$ and $m_{\mathrm{J}} = m_{\mathrm{S}}$. With the operators $\mathcal{I}$, $\mathcal{I}_{\mathrm{z}}$ and $\mathcal{S}_{\mathrm{z}}$ one has

$$\mathcal{H} = a\,\mathcal{I}\cdot\mathcal{S} + \frac{g_{\mathrm{S}}\,\mu_{\mathrm{B}}}{\hbar}\mathcal{S}_{\mathrm{z}}B_{\mathrm{z}} - \frac{g_{\mathrm{I}}\,\mu_{\mathrm{N}}}{\hbar}\mathcal{I}_{\mathrm{z}}B_{\mathrm{z}}, \tag{2.1}$$

where g_{S} and g_{I} are the gyromagnetic factors of the electron and nucleus, respectively. Furthermore, $\mu_{\mathrm{B}} = \frac{e\hbar}{2m_{\mathrm{e}}}$ is the Bohr magneton and $\mu_{\mathrm{N}} = \mu_{\mathrm{B}}\frac{m_{\mathrm{e}}}{m_{\mathrm{N}}}$ is the nuclear magneton. The coupling constant a can be calculated from the wavefunctions or from the measured hyperfine splitting (HFS) energy by

$$E_{\mathrm{HFS}} = \frac{a\hbar^2}{2}(2I + 1) = h\,\nu_{\mathrm{HFS}}, \tag{2.2}$$

which defines the hyperfine frequency ν_{HFS}. Using the definitions of the creation and annihilation operator $J_{\pm} = J_1 \pm iJ_2$ for the first term in the Hamiltonian the following identity holds:

$$\mathcal{I}\cdot\mathcal{S} = \mathcal{I}_{\mathrm{z}}\mathcal{S}_{\mathrm{z}} + \frac{1}{2}(\mathcal{I}_{+}\mathcal{S}_{-} + \mathcal{I}_{-}\mathcal{S}_{+}). \tag{2.3}$$

With this, the matrix elements of the Hamiltonian $\langle m_{S'}, m_{I'}|\mathcal{H}|m_S, m_I\rangle$ can be written in the basis $|m_S, m_I\rangle = \{|\frac{1}{2}, I\rangle \ldots |\frac{1}{2}, -I\rangle, |-\frac{1}{2}, -I\rangle \ldots |-\frac{1}{2}, I\rangle\}$ and the eigenvalue problem can be solved analytically for $J = S = \frac{1}{2}$. The solutions for arbitrary I are

$$
\begin{aligned}
E_1 &= E_{\mathrm{HFS}}\left(\frac{I}{2I+1}+\frac{1}{2}x\right), \\
E_j &= \frac{E_{\mathrm{HFS}}}{2(2I+1)}\left(-1+\sqrt{(x+1)^2(2I+1)^2+4x(1-j)(2I+1)}\right), \\
E_{2I+2} &= E_{\mathrm{HFS}}\left(\frac{I}{2I+1}-\frac{1}{2}x\right), \text{ and} \\
E_{4(\mathrm{I}+1)-j} &= \frac{E_{\mathrm{HFS}}}{2(2I+1)}\left(-1-\sqrt{(x+1)^2(2I+1)^2+4x(1-j)(2I+1)}\right)
\end{aligned}
\tag{2.4}
$$

with $j = 2 \ldots 2I+1$ and $x = \frac{B_\mathrm{z}}{B_\mathrm{c}}$. The critical B field B_c is defined by

$$
\hbar\omega_\mathrm{S}(B_\mathrm{z} = B_\mathrm{c}) = E_{\mathrm{HFS}} \tag{2.5}
$$

with the Larmor frequency

$$
\omega_\mathrm{S} = \frac{g_\mathrm{S}\,\mu_\mathrm{B}}{\hbar}B_\mathrm{z}. \tag{2.6}
$$

The relations of Eq. (2.5) give the energies of the 4 hyperfine states of hydrogen ($j = 2$) and the 6 states of deuterium ($j = 2, 3$) where the energy of the nuclear spin in the external B field is neglected due to the approximation $\frac{g_\mathrm{I}}{g_\mathrm{S}}\frac{m_\mathrm{e}}{m_\mathrm{N}} \approx 0$. The eigenstates of the Hamiltonian in arbitrary B fields are

$$
\begin{aligned}
|1\rangle &= \left|\frac{1}{2}, I\right\rangle, \\
|j\rangle &= \left|\frac{1}{2}, I-j+1\right\rangle\cos\theta_j + \left|-\frac{1}{2}, I-j+2\right\rangle\sin\theta_j, \\
|2(I+1)\rangle &= \left|-\frac{1}{2}, -I\right\rangle, \text{ and} \\
|4(I+1)-j\rangle &= -\left|\frac{1}{2}, I-j+1\right\rangle\sin\theta_j + \left|-\frac{1}{2}, I-j+2\right\rangle\cos\theta_j.
\end{aligned}
\tag{2.7}
$$

With $j = 2$ and $j = 2$ and 3 the mixing angles $\cos\theta_j$ and $\sin\theta_j$ are given by

$$
\cos\theta_j = \sqrt{\frac{1}{2}(1+\cos(2\theta_j))}, \tag{2.8}
$$

$$
\sin\theta_j = \sqrt{\frac{1}{2}(1-\cos(2\theta_j))}, \text{ and} \tag{2.9}
$$

$$
\cos(2\theta_j) = \frac{(x+1)(2I+1)+2(1-j)}{\sqrt{(x+1)^2(2I+1)^2+4x(1-j)(2I+1)}}. \tag{2.10}
$$

The force to a particle in state $|n\rangle$ in an inhomogeneous magnetic field is

$$\vec{F}_n = -\nabla E_n = -\frac{dE_n}{dx}\nabla\left(\frac{|\vec{B}|}{B_\mathrm{c}}\right) = -\frac{dE_n}{dx}\,\frac{1}{B_c}\nabla|\vec{B}| = -\mu_n\nabla|\vec{B}|. \tag{2.11}$$

From that one obtains the effective magnetic moment $\mu_n = \frac{dE_n}{dx}\,\frac{1}{B_\mathrm{c}}$ as

$$\begin{aligned}
\mu_1 &= \frac{1}{2}g_\mathrm{S}\,\mu_\mathrm{B}, && (2.12)\\
\mu_j &= \frac{1}{2}g_\mathrm{S}\,\mu_\mathrm{B}\,\cos(2\theta_j) \text{ for } j = 2\ldots 2I+1, && (2.13)\\
\mu_{2(\mathrm{I}+1)} &= -\frac{1}{2}g_\mathrm{S}\,\mu_\mathrm{B}, \text{ and} && (2.14)\\
\mu_{4(\mathrm{I}+1)-j} &= -\frac{1}{2}g_\mathrm{S}\,\mu_\mathrm{B}\,\cos(2\theta_j) \text{ for } j = 2\ldots 2I+1. && (2.15)
\end{aligned}$$

In case of several active magnetic fields at a certain point, one has to keep in mind that the forces from the fields do not add up linearly, i.e. $\vec{F}(\sum_i \vec{B}_i) \neq \sum_i \vec{F}_i(\vec{B}_i)$ since μ_n is nonlinear in $\vec{B}$. Therefore, one has to add up all the magnetic fields first and then use formula (2.11) to calculate the force of the combined fields. For the calculation of $x = \frac{|\vec{B}|}{B_\mathrm{c}}$ in $\mu(\vec{B})$ one needs only the absolute value of the combined field strength. However, for the gradient one needs the Jacobi matrices of the individual fields as well since

$$\begin{aligned}
\nabla\left|\sum_i \vec{B}_i\right| &= \nabla\sqrt{\left(\sum_i B_{i,\mathrm{x}}\right)^2 + \left(\sum_i B_{i,\mathrm{y}}\right)^2 + \left(\sum_i B_{i,\mathrm{z}}\right)^2}\\
&= \frac{1}{|\vec{B}|}\cdot J^\mathrm{T}(\vec{B})\cdot\vec{B}\\
&= \frac{1}{|\sum_i \vec{B}_i|}\cdot\left(\sum_i J_i^\mathrm{T}(\vec{B}_i)\right)\cdot\vec{B} && (2.16)
\end{aligned}$$

where $J^\mathrm{T}(\vec{B})$ is the transpose of the Jacobi matrix of $\vec{B} = \sum_i \vec{B}_i$.

The polarization of an ensemble of particles is defined as the normalized expectation value of the operators S_z for electron polarization P_e and I_z for the nuclear polarization P_z. The expectation value of an operator A for an ensemble is given by the trace of the density matrix:

$$\langle A\rangle = \mathrm{Tr}(\rho A) = \sum_k p_k\langle k|A|k\rangle \tag{2.17}$$

Table 2.1 – *Values for the hyperfine splitting of the ground state of hydrogen and deuterium. The frequency values are from [26]*

	ν_{HFS}/MHz	E_{HFS}/eV	B_{c}/mT
Hydrogen	1420.4057	$5.877 \cdot 10^{-6}$	50.7
Deuterium	327.3843	$1.355 \cdot 10^{-6}$	11.7

where p_k is the probability to find state $|k\rangle$ in the ensemble. With the relative population numbers n_k of state $|k\rangle$ one obtains

$$P_{\mathrm{e}} = n_1 - n_{2\mathrm{I}+2} + \sum_{j=2}^{2\mathrm{I}+1} (n_j - n_{4(\mathrm{I}+1)-j}) \cos(2\theta_j) \tag{2.18}$$

and

$$\begin{aligned} P_{\mathrm{z}} &= n_1 - n_{2\mathrm{I}+2} \\ &+ \sum_{j=2}^{2\mathrm{I}+1} \left[n_j \left(I - j + \frac{3}{2} - \frac{1}{2}\cos(2\theta_j) \right) + n_{4(\mathrm{I}+1)-j} \left(I - j + \frac{3}{2} + \frac{1}{2}\cos(2\theta_j) \right) \right]. \end{aligned} \tag{2.19}$$

For the deuteron this information is not sufficient to characterize the full polarization state, since there are three possible values of m_I. One additional number, named tensor polarization P_{zz}, is needed for the description

$$\begin{aligned} P_{\mathrm{zz}} &= \langle 3I_z^2 - 2 \rangle \\ &= n_1 + n_{2\mathrm{I}+2} \\ &+ \sum_{j=2}^{2\mathrm{I}+1} n_j \left[\frac{3}{2} \left((I-j+1)^2 (1 + cos(2\theta_j)) + (I-j+2)^2 (1 - \cos(2\theta_j)) \right) - 2 \right] \\ &+ \sum_{j=2}^{2\mathrm{I}+1} n_{4(\mathrm{I}+1)-j} \left[\frac{3}{2} \left((I-j+1)^2 (1 - \cos(2\theta_j)) + (I-j+2)^2 (1 + \cos(2\theta_j)) \right) - 2 \right]. \end{aligned} \tag{2.20}$$

The formulas so far have been for general nuclear spin I. In this thesis however, only the two cases with $I = \frac{1}{2}$ for hydrogen and $I = 1$ for deuterium are needed and will be given explicitly here. The hyperfine splitting energies for these atoms are known very precisely. The values for the frequency, the splitting energy and the critical magnetic field are given in Tab. 2.1.

For hydrogen one obtains the energies E_1 to E_4 with the dependence on the external field shown in Fig. 2.1.

$$
\begin{aligned}
E_1 &= \frac{E^{\mathrm{H}}_{\mathrm{HFS}}}{4}\left(1+2x\right), \\
E_2 &= \frac{E^{\mathrm{H}}_{\mathrm{HFS}}}{4}\left(-1+2\sqrt{1+x^2}\right), \\
E_3 &= \frac{E^{\mathrm{H}}_{\mathrm{HFS}}}{4}\left(1-2x\right), \text{ and} \\
E_4 &= \frac{E^{\mathrm{H}}_{\mathrm{HFS}}}{4}\left(-1-2\sqrt{1+x^2}\right), \\
\cos(2\theta_2) &= \frac{x}{\sqrt{x^2+1}}.
\end{aligned}
$$

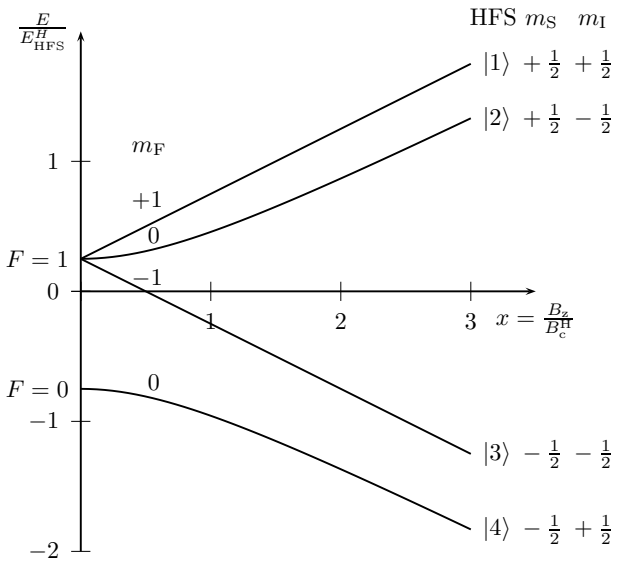

Figure 2.1 – *Hyperfine splitting energy in an external magnetic field for hydrogen*

The matrix $U^{-1} = U^{\mathrm{T}}$, containing all the eigenvectors as columns and diagonalizing $\mathcal{H}$ via $U^{-1}\mathcal{H}U$, is

$$
U = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta_2 & 0 & -\sin\theta_2 \\ 0 & 0 & 1 & 0 \\ 0 & \sin\theta_2 & 0 & \cos\theta_2 \end{pmatrix}. \tag{2.21}
$$

The electronic and nuclear polarization of a sample of hydrogen atoms is given by

$$
\begin{aligned}
P_{\mathrm{e}} &= n_1 - n_3 + (n_2 - n_4)\cos(2\theta) \\
P_{\mathrm{z}} &= n_1 - n_3 - (n_2 - n_4)\cos(2\theta)
\end{aligned} \tag{2.22}
$$

and for single states one obtains

$$
\begin{aligned}
\langle 1|P_{\mathrm{e,z}}|1\rangle &= 1, \\
\langle 2|P_{\mathrm{e,z}}|2\rangle &= \pm\cos(2\theta), \\
\langle 3|P_{\mathrm{e,z}}|3\rangle &= -1, \text{ and} \\
\langle 4|P_{\mathrm{e,z}}|4\rangle &= \mp\cos(2\theta)
\end{aligned}
$$

(upper sign for P_e lower sign for P_z).

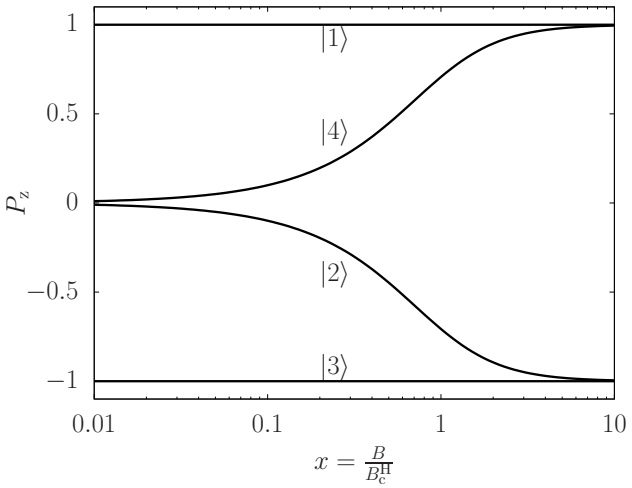

Figure 2.2 – *Nuclear polarization of the different hyperfine states $\langle k|P_\mathrm{z}|k\rangle$ for hydrogen*

The dependence of P_z on the external field is shown in Fig. 2.2. For deuterium one obtains the dependence on x for the energies E_1 to E_6, which are shown in Fig. 2.3

$$
\begin{aligned}
E_1 &= \frac{E_\mathrm{HFS}^\mathrm{D}}{6}\left(2+3x\right), \\
E_2 &= \frac{E_\mathrm{HFS}^\mathrm{D}}{6}\left(-1+\sqrt{9x^2+6x+9}\right), \\
E_3 &= \frac{E_\mathrm{HFS}^\mathrm{D}}{6}\left(-1+\sqrt{9x^2-6x+9}\right), \\
E_4 &= \frac{E_\mathrm{HFS}^\mathrm{D}}{6}\left(2-3x\right), \\
E_5 &= \frac{E_\mathrm{HFS}^\mathrm{D}}{6}\left(-1-\sqrt{9x^2-6x+9}\right), \\
E_6 &= \frac{E_\mathrm{HFS}^\mathrm{D}}{6}\left(-1-\sqrt{9x^2+6x+9}\right).
\end{aligned}
$$

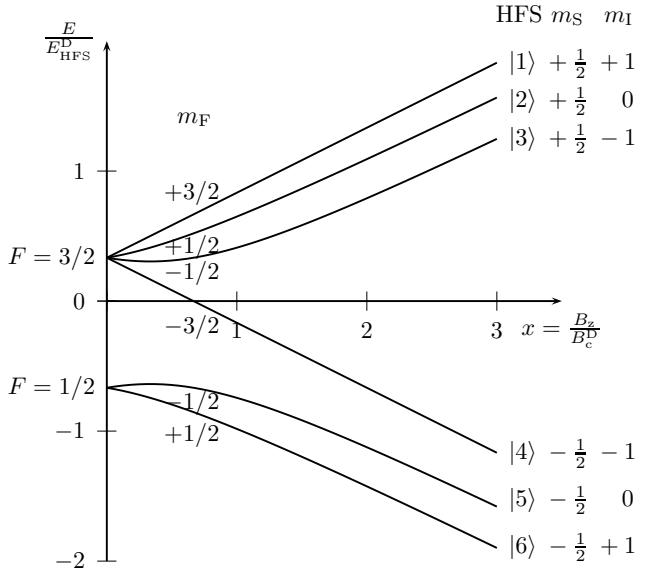

Figure 2.3 – *Hyperfine splitting energy in an external magnetic field for deuterium*

Here the mixing angles are

$$
\begin{aligned}
\cos(2\theta_2) &= \frac{3x+1}{\sqrt{9x^2+6x+9}} \quad \text{and} \\
\cos(2\theta_3) &= \frac{3x-1}{\sqrt{9x^2-6x+9}}.
\end{aligned}
\tag{2.23}
$$

The Matrix U is

$$U = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & \cos\theta_2 & 0 & 0 & 0 & -\sin\theta_2 \\ 0 & 0 & \cos\theta_3 & 0 & -\sin\theta_3 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & \sin\theta_3 & 0 & \cos\theta_3 & 0 \\ 0 & \sin\theta_2 & 0 & 0 & 0 & \cos\theta_2 \end{pmatrix}. \tag{2.24}$$

The polarizations of a sample of deuterium atoms are

$$\begin{aligned} P_{\mathrm{e}} &= n_1 - n_4 + (n_2 - n_6)\cos(2\theta_2) + (n_3 - n_5)\cos(2\theta_3), \\ P_{\mathrm{z}} &= n_1 - n_4 + \frac{1}{2}(n_2 + n_6 - n_3 - n_5) - \frac{1}{2}(n_2 - n_6)\cos(2\theta_2) - \frac{1}{2}(n_3 - n_5)\cos(2\theta_3), \text{ and} \\ P_{\mathrm{zz}} &= n_1 + n_4 - \frac{1}{2}(n_2 + n_3 + n_5 + n_6) - \frac{3}{2}(n_2 - n_6)\cos(2\theta_2) + \frac{3}{2}(n_3 - n_5)\cos(2\theta_3). \end{aligned} \tag{2.25}$$

For the 6 states one obtains for the nuclear vector polarization

$$\begin{aligned} \langle 1|P_{\mathrm{z}}|1\rangle &= 1, \\ \langle 2|P_{\mathrm{z}}|2\rangle &= \frac{1}{2}(1 - \cos(2\theta_2)), \\ \langle 3|P_{\mathrm{z}}|3\rangle &= -\frac{1}{2}(1 + \cos(2\theta_3)), \\ \langle 4|P_{\mathrm{z}}|4\rangle &= -1, \\ \langle 5|P_{\mathrm{z}}|5\rangle &= -\frac{1}{2}(1 - \cos(2\theta_3)), \text{ and} \\ \langle 6|P_{\mathrm{z}}|6\rangle &= \frac{1}{2}(1 + \cos(2\theta_2)) \end{aligned}$$

as shown in Fig. 2.4.

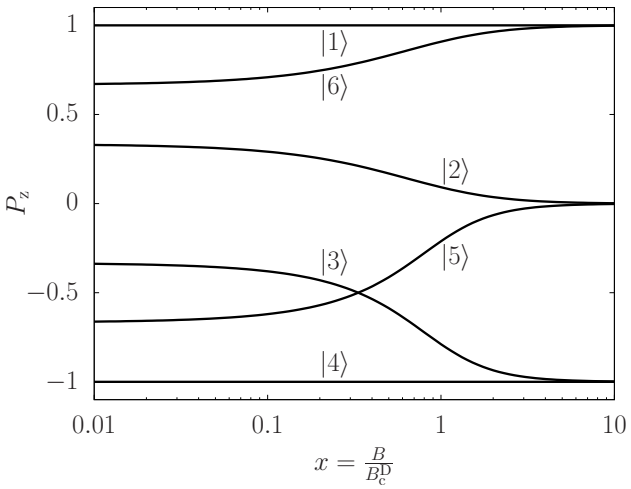

Figure 2.4 – *Nuclear polarization of the different hyperfine states $\langle k|P_{\mathrm{z}}|k\rangle$ for deuterium*

For the tensor polarization

$$
\begin{aligned}
\langle 1|P_{\mathrm{zz}}|1\rangle &= 1, \\
\langle 2|P_{\mathrm{zz}}|2\rangle &= -\frac{1}{2}(1+3\cos(2\theta_2)), \\
\langle 3|P_{\mathrm{zz}}|3\rangle &= -\frac{1}{2}(1-3\cos(2\theta_3)), \\
\langle 4|P_{\mathrm{zz}}|4\rangle &= 1, \\
\langle 5|P_{\mathrm{zz}}|5\rangle &= -\frac{1}{2}(1+3\cos(2\theta_3)), \\
\langle 6|P_{\mathrm{zz}}|6\rangle &= -\frac{1}{2}(1-3\cos(2\theta_2)).
\end{aligned}
$$

as shown in Fig. 2.5.

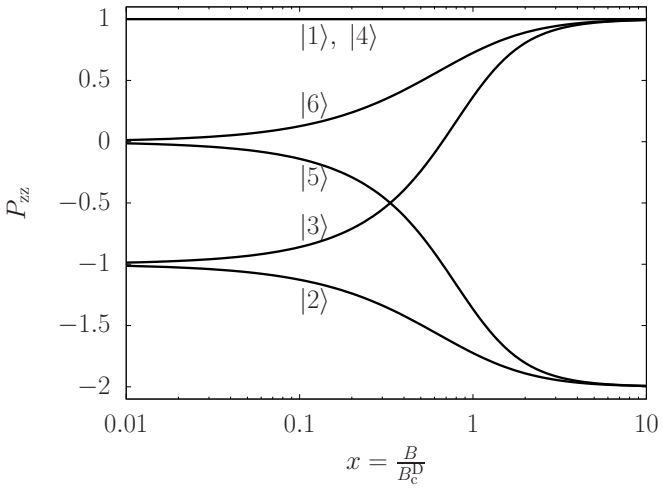

Figure 2.5 – *Tensor polarization of the different hyperfine states* $\langle k|P_{\mathrm{zz}}|k\rangle$ *for deuterium*

2.2 The Functionality of Polarized Internal Gas Targets

A target that is placed directly in the beam of a storage ring is called internal target. A polarized internal gas target consists of an Atomic Beam Source (ABS) and a gas-storage cell completed by a polarimeter to measure the polarization of the beam from the ABS or that of the target gas. A schematic picture of a setup is shown in Fig. 2.6.

The first part of an atomic beam source is a dissociator in which electrons are accelerated by an rf-field or by microwaves. The accelerated electrons loose kinetic energy, when they collide with the molecules and excite vibrations. If the energy transfer is higher than $8.8\,\mathrm{eV}$, the first molecular triplet state ${}^3\Sigma_{\mathrm{u}}^+$ can be excited, which leads to the dissociation of the molecule [14]. The electron bombardment heats the gas and a part of the atoms is ionized or excited from the ground state. In order to obtain a high degree of dissociation a small amount ($\approx 0.2\,\%$) of oxygen is added to the gas. At least two mechanisms were proposed for that. First, it was suggested that OH radicals bound to the wall increase the activation energy for hydrogen recombination thus lowering the recombination rate [27] and second, the OH radicals may contribute to the dissociation via the reaction OH+H_2 $\rightarrow$$H_2$O+H [28]. The dissociated gas with an admixture of undissociated molecular gas then flows through a cooled nozzle of typically $\approx 100\,\mathrm{K}$, in which it cools down and returns to the ground state. The produced water freezes at the surface and forms an ice layer which has to be removed after some days of running. On this ice layer some of the atoms recombine to molecules (compare with Sec. 2.3.1) and the recombination rate increases away from an optimum nozzle temperature thus limiting both the nozzle temperature and the degree of dissociation after the nozzle where the degree of dissociation, defined by the intensities of atoms, I_{a}, and molecules, I_{m}, is given by

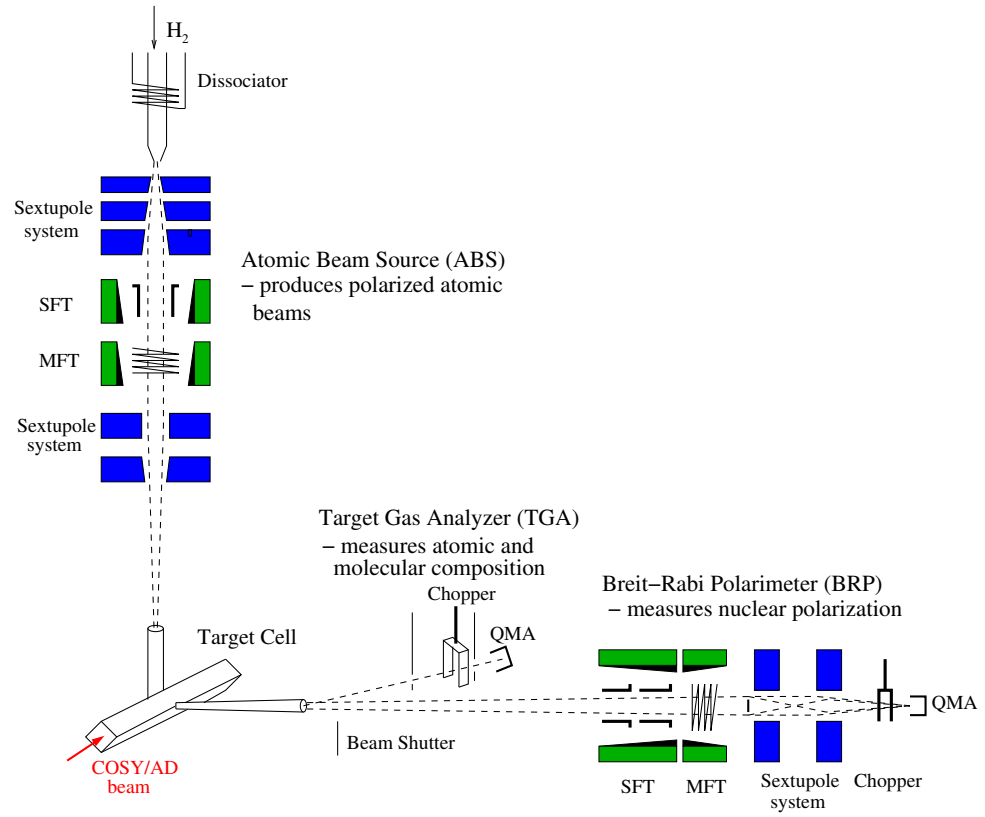

Figure 2.6 – *Schematic drawing of the PAX polarized target as installed at COSY and to be installed at the CERN AD. The figure shows the ABS in the vertical position, the storage cell around the beam axis, and the target diagnostic system in the horizontal position. The Target Gas Analyzer (TGA) determines the atomic to molecular fraction of an effusive beam from the storage cell and the Breit-Rabi Polarimeter (BRP) measures the intensities of the various hyperfine states to determine the polarization of the atomic storage-cell gas. The beam blocker in front of the sextupole magnets prevents particles near the beam axis from reaching the quadrupole mass analyzer (QMA), (from [8]).*

.

$$\alpha = \frac{I_\mathrm{a}}{I_\mathrm{a} + 2I_\mathrm{m}}. \tag{2.26}$$

The maximal degree of dissociation typically reaches about $80\,\%$ for low gas fluxes and decreases with increasing flux.

Behind the nozzle, the gas expands into the first of several differentially pumped vacuum chambers (five in the PAX-ABS) and forms a cold, supersonic beam, while the density rapidly drops and the flow regime changes from laminar to molecular flow. A skimmer and a collimator between nozzle and first sextupole magnet are used to cut a narrow beam with low divergence from the expanding gas. This beam passes through

a set of sextupole magnets, that exerts a force onto atoms in different hyperfine states as given by Eq. (2.11) and therefore focuses the beam of atoms in states with one orientation of the electron spin (e.g. states $|1\rangle$ and $|2\rangle$ for hydrogen), while it defocuses the beam of atoms in the other states. A set of medium-sized magnets is used for this task instead of only one large magnet in order to let the defocused atoms escape through the gap between the magnets. This reduces the gas density within the magnet bores and increases the transmission due to the decrease of scattering. The dimensions and strengths of the magnets have to be matched with the beam parameters (size, velocity distribution) in order to obtain maximal separation of the hyperfine states. In some cases it makes sense to use tapered magnets with conically shaped bores to reduce the pressure inside and to increase the transmission. A detailed description of the B field within the bore of the used Halbach type permanent sextupole magnets [29] can be found in appendix A.

Further important components of the ABS are the high frequency transition units, which are used to modify the occupation numbers of the hyperfine states in an rf-field. The necessary frequency for two states $|a\rangle$ and $|b\rangle$ depends on the energy difference $|E_a - E_b|$ and is given by

$$\nu = \frac{|E_a - E_b|}{h}, \tag{2.27}$$

which according to Eqs. (2.5) depends on the separation of the states by the magnetic field. In order to define the time that the atoms spend in the resonance region and for exactly matching ν and $|\vec{B}|$, the B field has a small gradient along the beam direction. The efficiency of this type of transition units is very high, above $98\,\%$ for some transitions. A mathematical description of such devices is given in [30] and it is found that, for an idealized case with only two hyperfine states (e.g. a free electron), the spin-flip probability p is given by

$$p = e^{-\pi \left| \frac{\mu_\mathrm{J} B_\mathrm{rf}^2}{\dot{B}_\mathrm{z} \hbar} \right|} \tag{2.28}$$

where μ_J is the magnetic moment of the electron, B_rf is the magnetic component of the high frequency field and $\dot{B}_\mathrm{z}$ is the dependence of the permanent field as seen by the particle. From this equation it follows that the spin-flip probability p is small for

$$\dot{B}_\mathrm{z} \ll \frac{2\mu_\mathrm{J}}{\hbar} B_\mathrm{rf}^2 \tag{2.29}$$

which is called the adiabaticity condition. In the classical view it means that, if the condition is fulfilled, the spins can follow the change of the B field, while otherwise the B field changes too fast and the spins cannot follow completely, which would lead to depolarization. For the case of atoms with mixed hyperfine states, no analytical

Table 2.2 – *List of hyperfine transitions in hydrogen and deuterium; Strong Field Transitions (SFTs) are characterized by $\Delta F = 1$ while Medium Field Transitions (MFTs) are characterized by $\Delta F = 0$. Δm_{F} is used to distinguish π ($\Delta m_{\mathrm{F}} = 1$) and σ ($\Delta m_{\mathrm{F}} = 0$) transitions and $\Delta m_{\mathrm{S}} = 1$ indicates a spin flip of the electron while $\Delta m_{\mathrm{I}} = 1$ indicates a transition of the nucleus (from [31]).*

Orientation	σ/π	Type	Δm_{F}	ΔF	Δm_{I}	Δm_{S}	Hydrogen	Deuterium
$B_{\mathrm{HF}} \parallel B_{\mathrm{z}}$	σ	SFT	0	1	1	1	$2 \leftrightarrow 4$	$2 \leftrightarrow 6$
$B_{\mathrm{HF}} \parallel B_{\mathrm{z}}$	σ	SFT	0	1	1	1		$3 \leftrightarrow 5$
$B_{\mathrm{HF}} \perp B_{\mathrm{z}}$	π	SFT	1	1	0	1	$1 \leftrightarrow 4$	$1 \leftrightarrow 6$
$B_{\mathrm{HF}} \perp B_{\mathrm{z}}$	π	SFT	1	1	0/2	1		$2 \leftrightarrow 5$, $3 \leftrightarrow 6$
$B_{\mathrm{HF}} \perp B_{\mathrm{z}}$	π	SFT	1	1	1	0	$3 \leftrightarrow 4$	$4 \leftrightarrow 5$
$B_{\mathrm{HF}} \perp B_{\mathrm{z}}$	π	MFT	1	0	1	0	$1 \leftrightarrow 2$	$1 \leftrightarrow 2$
$B_{\mathrm{HF}} \perp B_{\mathrm{z}}$	π	MFT	1	0	1	0		$2 \leftrightarrow 3$, $5 \leftrightarrow 6$
$B_{\mathrm{HF}} \perp B_{\mathrm{z}}$	π	MFT	1	0	0	1	$2 \leftrightarrow 3$	$3 \leftrightarrow 4$

equations for the transition probability exist but the situation is qualitatively the same as for the simple two-level system.

It should be mentioned that only transitions with $\Delta m_{\mathrm{F}} = 1$ (σ transition) and $\Delta m_{\mathrm{F}} = 0$ (π transition) are possible. In the first case the rf-field has to have a component parallel to the holding field, while in the second case the rf field has to be perpendicular to the holding field. For historical reasons the naming of the states is usually such that transitions with $\Delta F = 1$ are called Strong Field Transitions (SFT), while transitions with $\Delta F = 0$ are called Medium Field Transitions (MFT) or Weak Field Transitions (WFT) although some authors like Philpott in [30] use differing conventions. The WFTs are sometimes considered as a special way to run a cascade of MFTs in a multi-photon mode. Possible transitions are listed in Tab. 2.2.

For hydrogen, this first high frequency transition unit may interchange the population numbers of states $|2\rangle$ and $|3\rangle$ and in a second set of sextupole magnets the beam particles in hyperfine state $|3\rangle$ are defocused. In the ideal case, only atoms in state $|1\rangle$ are left in the atomic beam, which yields a highly polarized atomic gas target in a weak holding field. If the gas is injected into a region with high magnetic field, one can use two states at once, because both states are fully polarized in this limit (as can be seen in Fig. 2.2). The first transition unit is not used in this case and only at the end of the ABS behind the second set of magnets a final transition (not shown in Fig. 2.6) $|2\rangle \rightarrow |4\rangle$ or a cascade $|1\rangle \rightarrow |2\rangle \rightarrow |3\rangle$ is induced to obtain positive or negative vector polarization.

The polarized gas from the ABS is injected into a storage cell which is basically a system of tubes (compare Fig. 2.7). The beam of polarized atoms is injected by the feeding tube and diffuses to the outlets at the sides. Via the sampling tube a fraction of gas is taken from the cell to determine the polarization with the Breit-Rabi Polarimeter. During the

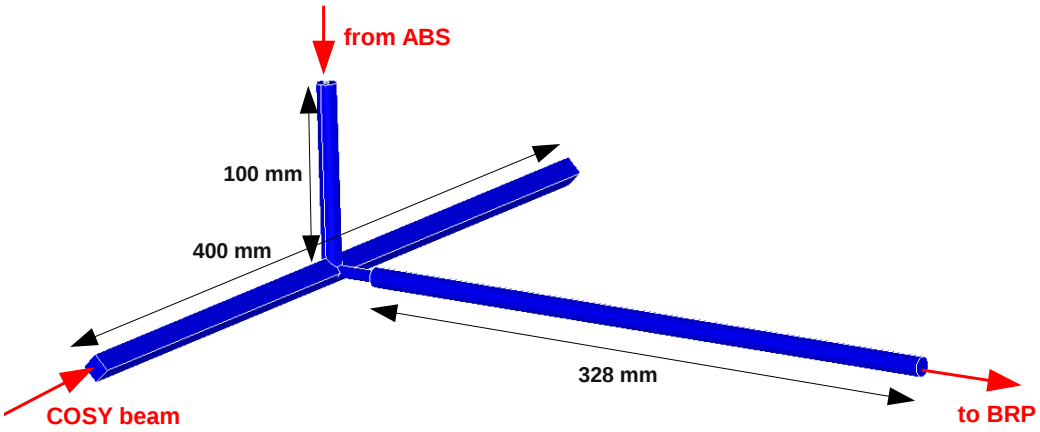

Figure 2.7 – *The PAX storage cell: The polarized beam from the ABS is injected from above by the feeding tube of* 10 *mm diameter into the quadratic interaction volume of* $10 \times 10\,mm^2$ *cross section. The diameter of the sampling tube for the BRP and TGA is* 6 *mm and then widened to* 10 *mm.*

diffusion process the atoms bounce back and forth inside the cell and such increase the probability of being hit by a particle from the accelerator beam by two orders of magnitude compared to a jet target. However, the wall collisions lead to recombination and depolarization (see Sec. 2.3) which (together with the detector size) limit the length of the cell and make it difficult to determine the gas polarization in the cell from a measurement with the BRP. The standard way of measuring the polarization is to take a fraction of the gas from the cell through a sampling tube and determine the relative intensities of the various hyperfine states with a BRP from which the polarization can be calculated according to Eq. (2.19). The BRP works essentially as the ABS, but it has an additional beam blocker on the beam axis. It stops particles with the wrong hyperfine state that leak through the system, because the sextupole magnets cannot deflect particles on their axis, where the gradient of the B field is zero. The beam intensity is measured with a Quadrupole Mass Analyzer (QMA). Another one is used in the Target Gas Analyser (TGA) to measure the molecular fraction.

2.3 Depolarization Mechanisms

There are three different mechanisms for depolarization – recombination, spin-exchange collisions and depolarization at the walls. All three effects will play only a minor role in the ABS itself, since there are almost no collisions between atoms, and particles desorbed from the wall, do most likely not reach the storage cell. However, recombination (without polarization loss) is important within the nozzle where it increases the molecular fraction. Within the storage cell mainly spin-exchange collisions

will take place although the mean free path length of the atoms is larger than the dimensions of the storage cell and hence only few spin-exchange collisions will occur. Measurements at HERMES showed that spin-exchange collisions reduce the nuclear polarization of hydrogen at the working point by about $3.3\,\%$, while depolarization at the walls causes a polarization loss of $2\,\%$ [31]. Although working conditions are different for the PAX target, depolarization at the walls is neglected in the the present state of the program.

2.3.1 Recombination

Hydrogen atoms are highly reactive and tend to recombine with other H atoms to hydrogen molecules. However, the atoms usually cannot recombine in a dilute gas, since in two body collisions both energy and momentum conservation cannot be satisfied. A third collision partner would be necessary for volume recombination, which is therefore highly unlikely. The recombination rate in hydrogen gas of atomic and molecular number density $n(\mathrm{H})$ and $n(\mathrm{H_2})$ with $n(\mathrm{H_2}) \ll n(\mathrm{H})$ is

$$\dot{n}(\mathrm{H_2}) = \frac{dn(\mathrm{H_2})}{dt} = k_{\mathrm{rec}} \cdot n^2(\mathrm{H}) \cdot n_3(\mathrm{H}), \tag{2.30}$$

where $k_{\mathrm{rec}} \approx 10^{-32}\,\mathrm{cm^6 s^{-1}}$ [32] and $n_3(\mathrm{H}) = n(\mathrm{H})$ is the number density of atomic hydrogen as third collision partner. In the storage cell the pressure is $\approx 5 \cdot 10^{-4}$ mbar or $n(\mathrm{H}) \approx 10^{13}$ atoms/cm^3, which yields $\dot{n}(\mathrm{H_2}) \approx 10^7$/cm^3·s. The relative rate $\dot{n}(\mathrm{H_2})/n(\mathrm{H})$ is of the order of $10^{-6}\,\mathrm{s^{-1}}$ and therefore volume recombination in the storage cell can be neglected. In the dissociator, however, the pressure is about 1 mbar. At this pressure $\dot{n}(\mathrm{H_2})/n(\mathrm{H}) \approx 0.1\,\mathrm{s^{-1}}$ affects the achievable degree of dissociation.

The situation is different at the walls of the containment, where atoms can be bound chemically or physically by Van-der-Waals forces. Atoms from the gas phase can recombine with these chemisorbed or physisorbed atoms. The description of this process is complicated and not well understood. There are basically two different reaction mechanisms which are dominant in different temperature regimes and describe different behaviors. The first one, called Eley-Rideal (E-R) mechanism, is a reaction between a gas phase atom colliding with an adsorbed atom and the second one, called Langmuir-Hinshelwood (L-H) mechanism, concerns two adsorbed atoms diffusing along the surface. Since the accommodation time at the surface decreases with increasing temperature and the mobility of the atoms decreases at low temperatures, the second mechanism mainly applies in the intermediate temperature range. In this temperature range the recombination probability $\gamma_{\mathrm{rec}}^{\mathrm{L-H}}$ of this process is [31]

$$\gamma_{\mathrm{rec}}^{\mathrm{L-H}} \propto e^{\frac{2E_{\mathrm{b}} - E_{\mathrm{j}}}{k_{\mathrm{B}} T}}, \tag{2.31}$$

where E_b is the binding energy of the atoms to the surface and E_j is the activation energy for surface diffusion. Contrary to that, the E-R mechanism works in all temperature ranges and the recombination probability $\gamma_\mathrm{rec}^\mathrm{E-R}$ can be approximated by [31]

$$\gamma_\mathrm{rec}^\mathrm{E-R} \approx k_1 e^{\frac{T_1}{T}} + k_2 e^{-\frac{T_2}{T}} \tag{2.32}$$

where the first term describes the increase in sticking time for physisorbed atoms at low temperatures and the second term describes an activation barrier for reactions with chemisorbed atoms. Another difference of the two recombination mechanisms is the energy distribution of the products. The energy of hydrogen molecules after recombination will be

$$E_{\mathrm{H}_2} = E_0 - E_\mathrm{b} + E_\mathrm{kin} - E_\mathrm{w} \tag{2.33}$$

where $E_0 = 4.5\,\mathrm{eV}$ is the binding energy of the H_2 molecule (measured at $T = 298\,\mathrm{K}$), E_b is the binding energy of the reactants to the surface, E_kin is the kinetic energy of the reactants, and E_w is the energy transferred to the wall. In L-H-type recombinations E_kin is approximately zero. Both atoms are bound to the surface and a large part of the energy E_0 is needed to free the molecule from the wall. Therefore, one can expect the molecules to be approximately in thermal equilibrium with the wall. In E-R type reactions on the other hand, there may be a considerable amount of kinetic energy and only one of the reactants is bound to the wall. This leads to molecules which are approximately in thermal equilibrium with the wall in the translational energy. They are, however, in very high rotational and vibrational excitated states such that the internal energy of the molecules is far from thermal equilibrium. They also retain parts of the incident transverse momentum leading to an asymmetric velocity distribution around the wall normal.

Modeling all these effects is extremely difficult, since the recombination probability depends on various microscopic and macroscopic parameters. The recombination probability depends strongly on the surface coverage, which in turn depends on the gas and wall temperatures, the gas density, and the surface properties which can be described by a surface potential depending on the material, the kinds of binding sites, geometry (steps, e.g., have a different potential than flat areas), orientation of the cutting plane through the crystal ('Miller indices') and surface impurities. As shown by experimental [33] and theoretical [34] work, a significant part of the reactions occurs via a so-called hot-atom pathway, in which the gas-phase atoms of the E-R mechanism do not hit the adsorbed atom directly but diffuse for a short time along the surface. This increases the cross section for this reaction type by an order of magnitude. Depending on the mass ratio of the adsorbed atoms to the wall, impinging light atoms lose little energy such that the hot-atom pathway is E-R-like.

For applications in internal gas targets not only the recombination of hydrogen is of interest but also the nuclear polarization of the atoms within the molecules. In an H_2 molecule, the electron spins have to follow the Pauli principle and point in opposite directions. On the other hand, the two relative orientations of the nuclear spins yield two energetically different kinds of hydrogen. The singlet state, where both nuclear spins point into opposite directions, is called parahydrogen and the triplet state, where both spins are parallel, is called orthohydrogen. Since the energy difference between these two forms of hydrogen are small, at room temperature the three times degenerate orthohydrogen occurs three times as often as parahydrogen. Which form is produced in a surface recombination event from a polarized gas, depends mainly on the surface properties of the wall and the polarizations of the reaction partners. The rate of depolarization depends on the external magnetic field strength. The recombination rate also depends on the electron polarization because of the Pauli principle.

One model for surface recombination is described in [35]. Based on rate equations for physisorption and chemisorption, for desorption of physisorbed atoms, and for chemical reactions the author builds a model which yields expressions for the recombination probabilities $\gamma^{\mathrm{L-H}}$ and $\gamma^{\mathrm{E-R}}$. Although the model is generic and describes the temperature and density dependence of the recombination probability, it depends on several (generally unknown) parameters which makes the model unhandy to use. In addition, no comparison of calculated rates with measurements is provided.

An equation for depolarization of molecules and its dependence on the external B field is given in [36]. The depolarization is interpreted as spin relaxation of orthohydrogen between wall collisions. The nuclear spins precess rapidly around an axis that is rotated to the external field by $\theta = \frac{B_{\mathrm{int}}}{B}$ where the internal B field for orthohydrogen in the $J = 1$ state is $B_{\mathrm{int}} = 5.4\,\mathrm{mT}$. Wall collisions randomize the direction of B_{int} and the component of the polarization along the external field decreases by $\frac{B_{\mathrm{int}}}{B^2}$ between two wall collisions. This leads to a remnant polarization of molecules after b wall collisions of

$$P(b) = P(0)\, e^{-b\left(\frac{B_{\mathrm{int}}}{B}\right)^2}. \tag{2.34}$$

Analog considerations apply for the deuterium atom and molecule. Differences occur due to (1) the higher mass, twice that of hydrogen, (2) the smaller magnetic moment of the deuteron, and (3) the fact that the deuteron with nuclear spin 1 is a boson. In the molecule the nuclear spins can couple to a total spin of 0, 1, and 2. Due to historical reasons, here molecules with with $I = 0$ and 2 are called orthodeuterium, those with $I = 1$ are called paradeuterium. They differ by the symmetry character of the total wave function. The moment of inertia of the deuterium molecule is twice that of hydrogen, whereas for both the nuclear distance is equal within $\approx 10^{-4}$. An example for the difference in kinematics is found in the velocity distribution of desorbed HD molecules. As shown in [33], the asymmetry of the velocity distribution of desorbed

HD molecules around the wall normal depends strongly on the mass ratio between the incoming and the adsorbed atom, such that there is a pronounced difference between the reactions $\mathrm{H} + \mathrm{D(wall)}$ and $\mathrm{D} + \mathrm{H(wall)}$.

2.3.2 Spin-Exchange Collisions

When two hydrogen atoms collide, they may temporarily form a bound state, during which the electrons interact and can be exchanged. Both effects look like exchange of angular momentum and consequently, the hyperfine population can change. Purcell and Field [37] developed a method to calculate the spin exchange in hydrogen and a calculation of the spin exchange probabilities is found in [38]. Two initial states $|a\rangle = a_\uparrow|\uparrow\ m_a - \frac{1}{2}\rangle + a_\downarrow|\downarrow\ m_a + \frac{1}{2}\rangle$ and $|b\rangle = b_\uparrow|\uparrow\ m_b - \frac{1}{2}\rangle + b_\downarrow|\downarrow\ m_b + \frac{1}{2}\rangle$ are considered and the probability of ending in states $|c\rangle$ and $|d\rangle$ is calculated. With the number N_i of particles in hyperfine state i and the total number density $n = 1/V \sum_i N_i$ in a volume V the the population rate of hyperfine state c is given by

$$\begin{aligned} \dot{N}_c &= 2n\,\overline{\sigma_{\mathrm{SE}} v_{\mathrm{r}}} \sum_{a,b} \left(|\langle cd|U|ab\rangle|^2 - |\langle cd|ab\rangle|^2 \right) N_a N_b && (2.35) \\ &= n\,\overline{\sigma_{\mathrm{SE}} v_{\mathrm{r}}} \sum_{a,b} M^c_{ab} N_a N_b, && (2.36) \end{aligned}$$

where $U = P_{\mathrm{s}} + e^{-i\Phi} P_{\mathrm{t}}$ is the time evolution operator of the model, where P_{s} is the projection operator for electron singlet states, P_{t} is the corresponding operator for electron triplet states, and

$$\Phi = \int \frac{V_{\mathrm{t}} - V_{\mathrm{s}}}{\hbar}\, dt \tag{2.37}$$

is the difference between the phase accumulations during the collision for singlet and triplet component of the electron spin function of the molecule. Here $V_{\mathrm{t}} - V_{\mathrm{s}}$ is the energy difference between the triplet and singlet electronic state of the molecule. Furthermore, σ_{SE} is the spin exchange cross section which is not much different from the cross section for momentum exchange σ_{T} [37]. The calculation yields

$$\begin{aligned} M^c_{ab} = \sum_d \Big[& -2\langle c|a\rangle\langle d|b\rangle \left(a_\uparrow b_\downarrow c_\uparrow d_\downarrow + a_\downarrow b_\uparrow c_\downarrow d_\uparrow\right) \\ & \langle m_c|m_a\rangle\langle m_d|m_b\rangle \left(a_\uparrow b_\downarrow c_\uparrow d_\downarrow + a_\downarrow b_\uparrow c_\downarrow d_\uparrow\right)^2 \\ & \langle m_c|m_a - 1\rangle\langle m_d|m_b + 1\rangle \left(a_\uparrow b_\downarrow c_\downarrow d_\uparrow\right)^2 \\ & \langle m_c|m_a + 1\rangle\langle m_d|m_b - 1\rangle \left(a_\downarrow b_\uparrow c_\uparrow d_\downarrow\right)^2 \Big]. \end{aligned} \tag{2.38}$$

Table 2.3 – *Pairs of hyperfine states for hydrogen and deuterium for which spin-exchange collisions are allowed by angular momentum conservation. Only final states within the same row as the initial state are allowed.*

$\sum_i m_\mathrm{F}$	Hydrogen	Deuterium
3		$\|11\rangle$
2	$\|11\rangle$	$\|12\rangle$, $\|16\rangle$
1	$\|12\rangle$, $\|14\rangle$	$\|13\rangle$, $\|15\rangle$, $\|22\rangle$, $\|26\rangle$, $\|66\rangle$
0	$\|13\rangle$, $\|22\rangle$, $\|24\rangle$, $\|44\rangle$	$\|14\rangle$, $\|23\rangle$, $\|25\rangle$, $\|36\rangle$, $\|56\rangle$
-1	$\|23\rangle$, $\|34\rangle$	$\|24\rangle$, $\|33\rangle$, $\|35\rangle$, $\|46\rangle$, $\|55\rangle$
-2	$\|33\rangle$	$\|34\rangle$, $\|45\rangle$
-3		$\|44\rangle$

Since the coefficients of the states depend on the magnetic field, the relaxation rate of the spins also depends on the magnetic field. With the time constant $\tau = \frac{1}{n\,\overline{\sigma_\mathrm{SE} v_\mathrm{r}}}$ one can write Eq. (2.36) as

$$\tau \dot{N}_c = \sum_{a,b} M^c_{ab} N_a N_b. \tag{2.39}$$

For atoms with $I = \frac{1}{2}$ the relaxation of the difference of the occupation number of hyperfine states 2 and 4, $N_2 - N_4$, follows an exponential with time constant

$$\tau_B = (1 + x^2)\tau \tag{2.40}$$

where $x = B/B_\mathrm{c} = g_\mathrm{S}\mu_\mathrm{B}B/E_\mathrm{HFS}$ as given in Sec. 2.1. For $I > \frac{1}{2}$ this time constant is still a useful approximation.

Due to angular momentum conservation, the states cannot interact arbitrarily. The possible interactions are listed in Tab. 2.3 and only final states within the same row of the table as the initial state can result from a spin-exchange collision. This is automatically included in Eq. (2.36).

For $t \to \infty$, the populations of the different hyperfine states will reach equilibrium values. The spin temperature equilibrium N_a^ste was introduced [26] to calculate the equilibrium values

$$N_a^\mathrm{ste} = \frac{e^{\beta m_a}}{\sum_a e^{\beta m_a}}. \tag{2.41}$$

If one defines $\eta = \tanh\left(\frac{\beta}{2}\right)$, one obtains for hydrogen and deuterium the values listed in Tab. 2.4 [31]. Note that the equilibrium values do not depend on the magnetic field. So these values depend only on the initial polarization of the ensemble of particles and will always be reached, whereas the rate of approach to the equilibrium will monotonically decrease with increasing B field.

Table 2.4 – *Population numbers and polarization of hydrogen and deuterium in spin temperature equilibrium.*

Value	Hydrogen	Deuterium
$\langle m_F \rangle$	η	$\eta \frac{11+\eta^2}{6+2\eta^2}$
β	$\ln\left(\frac{1+\eta}{1-\eta}\right)$	$\ln\left(\frac{1+\eta}{1-\eta}\right)$
n_1	$\frac{(1+\eta)^2}{4}$	$\frac{(1+\eta)^3}{6+2\eta^2}$
n_2	$\frac{1-\eta^2}{4}$	$n_2 = n_6 = \frac{(1+\eta)^2(1-\eta)}{6+2\eta^2}$
n_3	$\frac{(1-\eta)^2}{4}$	$n_3 = n_5 = \frac{(1-\eta)^2(1+\eta)}{6+2\eta^2}$
n_4	$\frac{1-\eta^2}{4}$	$\frac{(1-\eta)^3}{6+2\eta^2}$
P_e	η	η
P_z	η	$\frac{4\eta}{3+\eta^2}$
P_zz		$\frac{4\eta^2}{3+\eta^2}$

2.3.3 Spin Relaxation at Walls

When atoms strike a wall, they will be adsorbed due to Van-der-Waals interaction for a short time τ_s given by

$$\tau_\mathrm{s} = \tau_0 \, e^{\frac{E_\mathrm{b}}{k_\mathrm{B} T}} \tag{2.42}$$

with the high temperature sticking time τ_0 and the binding energy E_b. During that time they diffuse along the surface and the electron of the hydrogen or deuterium atom interacts with the local magnetic fields of the wall material. For polymers as wall material, where the atoms are bound together by covalent bindings, the electron spins do not contribute to the interaction with the adsorbed atom. Only the nuclear spins of the wall and possibly spin-orbit coupling can lead to depolarization. The theory for this was developed in [39] and [40] and was used for the investigation of the relaxation of optically pumped rubidium atoms on paraffin-coated walls [41] and for sodium on silicone surfaces [42]. Experiments with hydrogen and deuterium are described in [43] and [31]. The measurements in these two theses were performed at low temperatures ($\approx 100\,\mathrm{K}$) and for various holding field strengths. The effect is expected to decrease exponentially with increasing temperature because of the reduced sticking time τ_s and this was confirmed by measurements.

Generally, the effect of depolarization at the walls is expected to be smaller than the effects of spin-exchange collisions and it will be omitted in this work. Here, only a short theoretical description is provided following the discussion of Ref. [41].

For the derivation of relaxation rates, the Hamiltonian of the atom in a weak fluctuating B field $B(t)$ is given by

$$\mathcal{H} = \frac{g_{\mathrm{S}}\mu_{\mathrm{B}}}{\hbar} B(t) \cdot \mathcal{S}. \tag{2.43}$$

An exponential correlation function for the modulation of $B(t)$ is assumed with a correlation time τ_{c}. This correlation time can be interpreted as the time it takes for an adsorbed atom to diffuse from one place with a certain magnetic field to another place with the same field conditions. Then, the frequency dependence of the modulation is given by the Fourier transform of the correlation function

$$j(\omega) = \frac{1}{1 + \omega^2 \tau_{\mathrm{c}}^2}. \tag{2.44}$$

The relaxation rate for an isolated spin is given by

$$\frac{1}{T_{\mathrm{s}}} = C j(\omega_{\mathrm{S}}), \tag{2.45}$$

where ω_{s} is the Larmor frequency of the spin and the amplitude C is

$$C = \frac{2}{3} \frac{\tau_{\mathrm{s}}}{\tau_{\mathrm{s}} + \tau_{\mathrm{w}}} \left(\frac{g_{\mathrm{S}}\mu_{\mathrm{B}}}{\hbar} \right)^2 B^2 \tau_{\mathrm{c}} \tag{2.46}$$

where τ_{w} is the time between two wall collisions, and B is the amplitude of $B(t)$. Hyperfine structure and a static external magnetic field make the situation more complex and the rate of change of different hyperfine state populations can be written as

$$\frac{dN_i}{dt} = \sum_j R_{ij} N_j \tag{2.47}$$

with the transition rates R_{ij} given by

$$R_{ij} = C |\langle i|\mathcal{S}|j\rangle|^2 j(\omega_i - \omega_j). \tag{2.48}$$

Here $\omega_i - \omega_j$ is the hyperfine transition frequency which can be calculated with the use of Eqs. (2.5). The relaxation rates for hydrogen and deuterium were calculated in [44] and [45]. The solution for Eq. (2.47) is given by

$$N_i = \sum_{j=1}^{2(2I+1)} c_j V_j e^{-\lambda_j t}, \tag{2.49}$$

where V_j is the eigenvector of R_{ij} corresponding to the eigenvalue λ_j and the c_j are determined by the initial conditions. From this, one can calculate the polarization with

Eqs. (2.18) to (2.21). The result can be quite complicated but for the case of low external B field simplified formulas exist for the expectation values of $\mathcal{I}_z$ and $\mathcal{S}_z$:

$$\langle \mathcal{I}_z \rangle = \langle \mathcal{I}_z \rangle_0 e^{-\frac{t}{T_n}}, \tag{2.50}$$

$$\langle \mathcal{S}_z \rangle = \left(\langle \mathcal{S}_z \rangle_0 - \frac{2}{(2I+1)^2 - 2} \langle \mathcal{I}_z \rangle_0 \right) e^{-\frac{t}{T_e}} + \frac{2}{(2I+1)^2 - 2} \langle \mathcal{I}_z \rangle_0 e^{-\frac{t}{T_n}}. \tag{2.51}$$

With the electronic (nuclear) relaxation times T_e (T_n) the relaxation rates are given by

$$\frac{1}{T_e} = C \left(\frac{j(\omega_f) - j(\omega_{HFS})}{(2I+1)^2} + j(\omega_{HFS}) \right) \quad \text{and} \tag{2.52}$$

$$\frac{1}{T_n} = \frac{C}{(2I+1)^2} (j(\omega_f) + j(\omega_{HFS})). \tag{2.53}$$

Here $\omega_{HFS} = \frac{E_{HFS}}{\hbar}$ and $\omega_f = \frac{E_Z}{\hbar}$ with E_Z being the Zeeman splitting energy. This shows that the nuclear spin relaxes with a single rate, while the electron spin has two different rates. For the case with high external B field, where electron and nuclear spin are decoupled, the relaxation rate of $\langle \mathcal{S}_z \rangle$ is given by Eq. (2.45) and for intermediate field one has to use Eq. (2.49) to calculate the different relaxation rates. Reference [46] proposes a simpler model of atom polarization depending on the average collision age: The polarization of a set of atoms is given by

$$\langle P \rangle = \frac{\int_0^\infty P(b) n(b)\, db}{\int_0^\infty n(b)\, db} \tag{2.54}$$

where $n(b)$ is the number of atoms having undergone b wall collisions and $P(b)$ is the polarization after b wall collisions. The distribution of collision ages for a simple geometry is

$$n(b) = e^{-\frac{b}{N_0}}, \tag{2.55}$$

where N_0 is the average collision age of the atoms.

With the assumption of an exponential decrease of the polarization over time and the sticking time to the surface as given by Eq. (2.42)

$$P(b) = P(0)\, e^{-\frac{\tau_S}{\tau_R} b} \tag{2.56}$$

one can integrate Eq. (2.54) and finally obtains

$$\langle P \rangle = \frac{P(0)}{1 + \frac{N_0\, \tau_0}{\tau_R}\, e^{\frac{E_b}{k_B T}}}. \tag{2.57}$$

On metal surfaces, where electrons are only loosely bound, electron exchange can be important in the depolarization process. This or fluctuating local fields lead to the conclusion ([36]) that atoms, adsorbed to a copper surface, are completely depolarized and the molecules formed in an E-R-type recombination retain half the original nuclear polarization. Other groups ([47, 48, 49]) investigated other surfaces and showed that significantly higher nuclear polarizations in molecules are obtained from recombination on non-metallic surfaces.

2.4 Definition of Basic Gas Properties

N_j particles of type j with mass m_j and number density $n_j = \frac{N_j}{V}$ in a volume V within an equilibrium gas have a Maxwellian velocity distribution

$$\frac{dN_j}{N_j} d\,\vec{v} = \frac{dn_j}{n_j} d\,\vec{v} = \frac{\beta_j^3}{\pi^{\frac{3}{2}}} e^{-\beta_j^2 |\vec{v}'|^2} d\,\vec{v}, \tag{2.58}$$

where the thermal velocity $\vec{v}'$ is given by the difference of the total velocity $\vec{v}$ and the stream velocity $\vec{v}_0$, i.e., $\vec{v}' = \vec{v} - \vec{v}_0$, and β_j is the inverse of the most probable thermal velocity,

$$\beta_j = \frac{1}{|\vec{v}'_{\mathrm{m},j}|} = \sqrt{\frac{m_j}{2k_{\mathrm{B}}T}}. \tag{2.59}$$

Macroscopic properties of the gas are given for every point in space, but are defined microscopically by averaging over all particles within an extended volume element no matter if the gas is in equilibrium or not. For example the mean velocity of a gas, composed of different particle species, is the average over all velocities of all particles k_j of all particle species j:

$$\overline{\vec{v}} = \frac{1}{N} \sum_{j,k_j} \vec{v}_{j,k_j}. \tag{2.60}$$

The mean thermal speed $\overline{\vec{v}'}$ and the mean stream velocity $\overline{\vec{v}_0}$ are defined similarly as well as the mean kinetic (or translational) energy per particle

$$\overline{E_{\mathrm{t}}} = \frac{1}{N} \sum_{j,k_k} \frac{1}{2} m_j \vec{v}_{k_j}^2. \tag{2.61}$$

With this, the translational temperature T_{tr} can be defined by

$$\frac{3}{2} k_{\mathrm{B}} T_{\mathrm{tr}} = \frac{1}{2} \overline{m\vec{v}'} = \overline{E_{\mathrm{t}}} - \frac{1}{2} \overline{m} \overline{\vec{v}_0}^2. \tag{2.62}$$

This is actually the temperature used in Eq. (2.59) and it can also be used to express the ideal gas equation

$$p = k_{\mathrm{B}} n T_{\mathrm{tr}} \tag{2.63}$$

and to define the pressure p. If the molecules of species j have ζ_j internal degrees of freedom, one can also define an internal temperature by

$$\frac{\overline{\zeta}}{2} k_{\mathrm{B}} T_{\mathrm{int}} = \overline{E_{\mathrm{int}}}, \tag{2.64}$$

where $\overline{E_{\mathrm{int}}}$ is the mean internal energy. In the equilibrium state $T_{\mathrm{int}} = T_{\mathrm{tr}}$ and in general an overall temperature can be defined as the weighted average of the translational and internal temperatures as

$$T_{\mathrm{ov}} = \frac{3T_{\mathrm{tr}} + \overline{\zeta} T_{\mathrm{int}}}{3 + \overline{\zeta}}. \tag{2.65}$$

3 The Direct Simulation Monte Carlo (DSMC) Method

3.1 Motivation

Depending on various microscopic and macroscopic parameters, gases can have different properties which can be described by a few numbers, either dimensioned like viscosity and diffusion coefficients or dimensionless like the Reynolds number or the Knudsen number. The latter one is of particular interest here and is defined by $Kn = \frac{\lambda}{L}$, where λ is the mean free path length of the gas atoms or molecules and L is a characteristic length of the system like the diameter of a gas-flow channel. It can also be defined locally, e.g., as $L = \frac{\rho}{|\nabla\rho|}$ by the local density ρ and its gradient. Then the locally defined Knudsen number becomes

$$Kn = \frac{\lambda}{\frac{\rho}{|\nabla\rho|}}. \tag{3.1}$$

The Knudsen number can be used to define the validity ranges of equations that describe the gas behavior. The most general gas equation is the Boltzmann equation that is valid for all Knudsen numbers under the assumption of molecular chaos and the assumption that only two body collisions occur:

$$\frac{\partial}{\partial t}(nf) + \vec{v}\cdot\frac{\partial}{\partial\vec{r}}(nf) + \vec{F}\cdot\frac{\partial}{\partial\vec{v}}(nf) = \int_{-\infty}^{\infty}\int_{0}^{4\pi} n^2(f'f_1' - ff_1)v_\mathrm{r}\sigma\, d\Omega d\vec{v}_1. \tag{3.2}$$

This equation describes the flux of particles in and out of a volume element. The first term is the change of the particle number in a volume element, where n is the number density and f is the velocity distribution function of the particles. The second term describes the particle flux caused by the movement of the particles with velocity $\vec{v}$ and the third term is the additional flux caused by the force $\vec{F}$ per unit mass. The right hand side describes the particle flux caused by collisions of particles with relative velocity v_r and differential cross section σ. The velocity distribution functions f and f_1 are for particles of velocity $\vec{v}$ and $\vec{v}_1$ before collision, while the prime denotes the post-collision values. Solving this equation is extremely difficult. The easiest case is encountered for no collisions which corresponds to the limit $Kn \to \infty$. In the other limiting case $Kn \to 0$, the velocity distributions can be taken as the local equilibrium Maxwell distribution which leads to the Euler equations for inviscid flows. For small

Knudsen numbers the velocity distribution functions can be approximated by a perturbation to the Maxwellian distribution. For first order perturbations this leads to the Chapman-Enskog theory and the Navier-Stokes equations. The Navier-Stokes equations start to fail when the gradients of the macroscopic variables become so steep that their scale length is of the same order as the average distance traveled by the molecules between collisions, which is the case for $Kn \approx 0.1$. For larger Kn there are generally no analytical solutions to the Boltzmann equation available and this is especially the case in gases at very low densities or in shock structures.

Solving Eq. (3.2) numerically is very demanding and therefore people started to make use of the molecular structure of gases. The first molecular dynamics calculations for solving a many particle problem of interacting particles was reported 1959 [50]. Because of the huge number of particles, however, this original method is not suitable for simulating gas flows. Therefore, Bird [51] in 1963 used a probabilistic approach to solve this problem and the method was developed over the years to become the Direct Simulation Monte Carlo (DSMC) method.

3.2 The DSMC Method

The basic assumptions of the DSMC method are that the movement of particles and their collisions can be separated on small timescales, which is true in the limit of $\Delta t \to 0$, and that the gas properties can be derived by using only a relatively small number of stochastically interacting simulated particles whose properties are initially sampled from the local equilibrium distribution functions. Further, only binary collisions are assumed to occur which limits the method to gases of low densities. Then, the method proceeds in three steps: First, during the time step Δt, all particles are moved ballisticly along deterministic tracks. If a wall is encountered, the particle properties will be changed according to a certain wall interaction model. In the second step, possible collision partners are selected from a neighborhood of the particles and collisions are performed with a certain probability according to a collision model (see Sec. 3.3). This is assumed to happen instantaneously in the model, i.e., the simulated time is not advanced during this step. In the last step, the new macroscopic gas properties are sampled from the microscopic ensemble before the whole procedure starts again. Although the macroscopic gas properties are defined for every point in space, a reasonable value can only be defined by averaging over an ensemble of particles. Therefore the simulated region is usually split into small cells and the calculation of the macroscopic gas properties is done by averaging over all particles within such a cell, while possible collision partners can be chosen only from this cell.

This method empirically proved very useful for a wide range of situations, both equilibrium and nonequilibrium ones. Theoretically it was shown that the solution of a

DSMC simulation converges to a solution of the Boltzmann equation in the limit of vanishing cell size, time step, and infinitely large particle number [52]. In real simulations these conditions cannot be fulfilled exactly and therefore various discretization errors occur. Besides numerical errors and statistical fluctuations that decrease with the square root of the sample size, there are three error sources: Cell size, time step, and number of particles per cell. Various authors investigated these errors theoretically by means of the Green-Kubo theory [53] and by simulations. In these studies, simulation results are compared to exact theoretical values that can be obtained for simple problems. Errors in cell size occur since the collision partners are chosen from the whole cell and a too large separation of the collision partners leads to physically unrealistic collisions which tend to smear out gradients. In [54], [55] it was found that for the limiting case $\Delta t \to 0$ and $N_\mathrm{c} \to \infty$, where N_c is the number of simulated particles per cell, the discretization error caused by the finite cell size is proportional to $(\Delta\tilde{x})^2$ where $\Delta\tilde{x} = \frac{\Delta x}{\lambda}$ is the dimensionless cell size. Time discretization errors occur since the collision partners are only selected at the end of each time step although in reality the collisions would occur continuously. In the limit of vanishing cell size and particle number errors, the time step error was found proportional to the square of the dimensionless time step $\Delta\tilde{t} = \frac{t}{t_0}$, where t_0 is the mean collision time [56]. Finally, [57] reports that the particle number error is proportional to $\frac{1}{N_\mathrm{c}}$ in the limiting case of vanishing other discretization errors. A comprehensive study of the thermal conductivity of a gas between two differentially heated plates found [58] that the simulated thermal conductivity $\overline{K}$ divided by the theoretical value K can be described in the presence of all three errors by

$$\frac{\overline{K}}{K} = 1.0001 + 0.0287(\Delta\tilde{t})^2 + 0.0405(\Delta\tilde{x})^2 - \frac{0.083}{N_\mathrm{c}} + F\left(\Delta\tilde{t}, \Delta\tilde{x}, \frac{1}{N_\mathrm{c}}\right), \tag{3.3}$$

where $F\left(\Delta\tilde{t}, \Delta\tilde{x}, \frac{1}{N_\mathrm{c}}\right)$ is a function of higher order terms in the declared variables. However, it was also found that the convergence behavior depends on the exact DSMC method used, its implementation, the measure by which the convergence behavior is measured, and the problem under investigation. Generally it can be noted that the time step should be a fraction of the mean collision time, that the cell size should be a fraction of the mean free path length, and that the numbers per cell should be around 20. In real simulations with large pressure differences these conditions are hard to fulfill and the time step is determined by the largest pressure area. This may lead to time steps which are much too small for large parts of the domain which slows down the computation significantly.

The DSMC method is extensively described in [59] and is often called DSMC94 in the literature. Many improvements have been suggested to increase convergence and computational efficiency like the use of subcells for the selection of collision partners to reduce the mean collision separation or sampling twice, once before and once after the collision step. Some new developments, implemented in a code generally termed

DSMC07 [60], include the use of different collision selection mechanisms and adaptive, cell based time steps which makes the code faster, especially in simulations with large pressure differences. However, it was found in [61] that the new scheme introduces stringent requirements for the temporal discretization. It should further be noted that cell-based collision-partner schemes are not applicable to charged particles since their cross section is generally much larger than the cell and therefore possible collision partners cannot be selected. Therefore, Olson and Christlieb report about the development of a gridless DSMC code which is intended to become part of a plasma simulation tool [62].

3.3 Binary Collisions

In rarefied gas flows only two body or binary collisions are likely to occur while three body collisions are negligible. Realistic descriptions of binary collisions are very complicated in general, especially when inelastic collisions are considered where energy is redistributed between the translational and internal degrees of freedom. In general, the interaction potential of molecules cannot be given or would need excessive computational effort. Therefore one is limited to simple collision models. They are allowed to be as simple as possible as long as they yield appropriate results. Only when the results obtained from a simple model do not agree with measurements a more complex model should be used. In the following section some simple models will be described and formulas for the relevant quantities will be given. Much of this section closely follows Chap. 2 of [59]. All models considered here are purely classical. Quantum effects are expected to occur only at very low temperatures and can therefore be neglected in general. Quantum mechanical models using partial waves to calculate a cross section are also less efficient and it was found in [63] that the macroscopic gas properties calculated with quantum mechanical models agree well with the ones calculated with classical models although the differential cross sections differ in the two approaches.

The simplest collision models consider only elastic collisions and make use of a rotational symmetric molecular potential. In this case the collision has to obey the three conservation laws for momentum, energy and angular momentum. If the two colliding particles have masses m_1 and m_2, pre-collision velocities $\vec{v}_1$ and $\vec{v}_2$ and post-collision velocities $\vec{v'}_1$ and $\vec{v'}_2$ the first two conservation laws can be written as

$$m_1\,\vec{v}_1 + m_2\,\vec{v}_2 = m_1\,\vec{v'}_1 + m_2\,\vec{v'}_2 = (m_1+m_2)\,\vec{v}_\mathrm{m} \tag{3.4}$$

$$m_1\,v_1^2 + m_2\,v_2^2 = m_1\,v_1'^2 + m_2\,v_2'^2 \tag{3.5}$$

where $\vec{v}_\mathrm{m}$ is the velocity of the center of mass. Defining the relative velocities before and after the collision, $\vec{v}_\mathrm{r} = \vec{v}_1 - \vec{v}_2$ and $\vec{v'}_\mathrm{r} = \vec{v'}_1 - \vec{v'}_2$, one can write

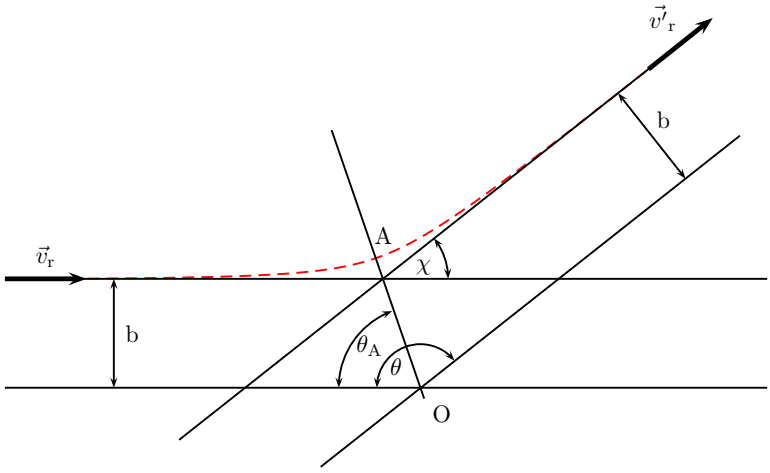

Figure 3.1 – *Collisions in the center of mass system; One particle is located at the origin O and the other one approaching with impact parameter b and velocity $\vec{v}_{\mathrm{r}}$ is deflected by an angle χ in its potential Φ. The problem is symmetric around the axis O-A.*

$$\vec{v}_1 = \vec{v}_{\mathrm{m}} + \frac{m_2}{m_1 + m_2}\,\vec{v}_{\mathrm{r}} \tag{3.6}$$

$$\vec{v}_2 = \vec{v}_{\mathrm{m}} - \frac{m_1}{m_1 + m_2}\,\vec{v}_{\mathrm{r}} \tag{3.7}$$

and

$$\vec{v'}_1 = \vec{v}_{\mathrm{m}} + \frac{m_2}{m_1 + m_2}\,\vec{v'}_{\mathrm{r}} \tag{3.8}$$

$$\vec{v'}_2 = \vec{v}_{\mathrm{m}} - \frac{m_1}{m_1 + m_2}\,\vec{v'}_{\mathrm{r}}. \tag{3.9}$$

$\vec{v}_{\mathrm{m}}$ will not change without external forces and in elastic collisions the magnitude of the relative velocities before and after the collision will be the same, i.e. $|\vec{v}_{\mathrm{r}}| = |\vec{v'}_{\mathrm{r}}|$. $\vec{v}_{\mathrm{m}}$ and $\vec{v}_{\mathrm{r}}$ can be calculated from the pre-collision velocities and hence the only task left is to determine the change in direction χ of the relative velocity vector during the collision.

Writing the position vector of particle i as $\vec{r}_i$, the equations of motion for both collision partners are

$$m_1\,\ddot{\vec{r}}_1 = \vec{F} \tag{3.10}$$

$$m_2\,\ddot{\vec{r}}_2 = -\vec{F} \tag{3.11}$$

which can be combined to an effective one body problem with the reduced mass $m_\mathrm{r} = \frac{m_1 m_2}{m_1+m_2}$ and the relative position $\vec{r} = \vec{r}_1 - \vec{r}_2$ in the force field of a fixed particle:

$$m_\mathrm{r} \ddot{\vec{r}} = \vec{F}. \tag{3.12}$$

The central force $\vec{F}(r)$ can be written as the derivative of a scalar potential Φ as

$$\vec{F}(r) = -\frac{d\Phi}{dr}\,\vec{e}_\mathrm{r} \tag{3.13}$$

and the equation of motion can be written in terms of the total energy

$$\frac{1}{2} m_\mathrm{r} \left(\dot{r}^2 + r^2\dot{\theta}^2\right) + \Phi = E = \frac{1}{2} m_\mathrm{r} v_\mathrm{r}^2 \tag{3.14}$$

where the last term comes from the asymptotic large separations of the two particles with no potential energy. In order to eliminate the time dependence, one can use angular momentum conservation $r^2\dot{\theta} = const = b v_\mathrm{r}$ with b being the impact parameter, to obtain

$$\left(\frac{dr}{d\theta}\right)^2 = \frac{r^4}{b^2} - r^2 - \frac{\Phi\, r^4}{\frac{1}{2} m_\mathrm{r}\, v_\mathrm{r}^2\, b^2}. \tag{3.15}$$

Using the dimensionless coordinate

$$W = \frac{b}{r} \tag{3.16}$$

this reduces to

$$\left(\frac{dW}{d\theta}\right)^2 = 1 - W^2 - \frac{\Phi}{\frac{1}{2}\, m_\mathrm{r}\, v_\mathrm{r}^2} \tag{3.17}$$

so that

$$\theta = \int_0^W \frac{1}{\sqrt{1 - W^2 - \frac{\Phi}{\frac{1}{2}\, m_\mathrm{r}\, v_\mathrm{r}^2}}}\, dW. \tag{3.18}$$

Now, one can make use of the symmetry of the problem. At the intersection of the orbit with the apse line OA (compare Fig. 3.1) it is

$$\theta = \theta_\mathrm{A} \tag{3.19}$$

and

$$\frac{dW}{d\theta} = 0. \tag{3.20}$$

Therefore,

$$\theta_{\mathrm{A}} = \int_0^{W_1} \frac{1}{\sqrt{1 - W^2 - \frac{\Phi}{\frac{1}{2}\, m_{\mathrm{r}}\, v_{\mathrm{r}}^2}}}\, dW \tag{3.21}$$

where W_1 is the positive root of the equation

$$1 - W^2 - \frac{\Phi}{\frac{1}{2}\, m_{\mathrm{r}}\, v_{\mathrm{r}}^2} = 0. \tag{3.22}$$

From this one can obtain the deflection angle of the relative velocity as

$$\chi = \pi - 2\theta_{\mathrm{A}} \tag{3.23}$$

The final task now is to get the post-collision velocities from this value. Angular momentum conservation requires that all velocities stay in one plane during the collision in the center of mass frame. This plane may be rotated by an angle ϕ versus some reference plane. Generally ϕ will be equally distributed in the interval $[0, 2\pi]$ and can be chosen randomly. In Cartesian coordinates $\vec{v'}_{\mathrm{r}}$ can be written as

$$\vec{v'}_{\mathrm{r}} = v_{\mathrm{r}} \begin{pmatrix} \cos\chi \\ \sin\chi\, \cos\phi \\ \sin\chi\, \sin\phi \end{pmatrix}. \tag{3.24}$$

This can now be used to calculate the post-collision velocities according to Eqs. (3.8) and (3.9).

The differential cross section of the collision can be obtained from

$$\sigma\, d\Omega = b\, db\, d\phi \tag{3.25}$$

with $d\Omega = \sin\chi\, d\chi\, d\phi$ being the solid angle around $\vec{v'}_{\mathrm{r}}$. This yields

$$\sigma = \frac{b}{\sin\chi} \left| \frac{db}{d\chi} \right|. \tag{3.26}$$

Finally, in order to get the total cross section, one has to integrate over all angles

$$\sigma_{\mathrm{T}} = \int_0^{4\pi} \sigma\, d\Omega = 2\pi \int_0^{\pi} \sigma\, \sin\chi\, d\chi. \tag{3.27}$$

For some more realistic molecular models this integral will diverge and one has to introduce effective or nominal cross sections. This can be achieved by using cutoffs on b or on χ, where in the case of a collision it should hold $b < b_0$ or $\chi > \chi_0$. In all other cases the interaction is considered too small to be counted as collision and it was shown in [64] that if quantum effects are taken into account, such glancing collisions cannot be properly defined. However, since these cutoffs are arbitrary, the resulting total cross section is not suitable for defining an efficient collision frequency or mean free path length. In order to define these properties, a nominal cross section $\sigma = \pi d^2/4$ may be defined, where the diameter d may depend on the relative velocity v_r of the collision partners.

According to the Chapman-Enskog theory, the viscosity and diffusion of a gas depend on the differential cross sections rather than the total cross section. This fact makes it possible to define models that lead to a diverging total cross section but still exhibit the correct viscosity and diffusion. Only the collision frequency and the mean free path length then depend on the total cross section, that can be chosen in a suitable way. These quantities will therefore be effective ones, i.e. they will be model dependent and not equal to the correct ones which one would obtain from measurements. For later use, the viscosity cross section σ_μ and the diffusion cross section σ_D in the Chapman-Enskog theory are defined as

$$\sigma_\mu = \int_0^{4\pi} (1 - \cos^2\chi)\, \sigma \, d\Omega = 2\pi \int_0^{\pi} \sigma \, \sin^3\chi \, d\chi \tag{3.28}$$

$$\sigma_\mathrm{D} = \int_0^{4\pi} (1 - \cos\chi)\, \sigma \, d\Omega = 2\pi \int_0^{\pi} \sigma \, (1 - \cos\chi) \, \sin\chi \, d\chi. \tag{3.29}$$

These integrals are convergent but usually very difficult to solve.

3.3.1 Binary Collision Models

Inverse Power Law Model

One of the simplest models is the inverse power law model. The force on a particle is simply given by

$$F = \frac{\kappa}{r^\eta} \tag{3.30}$$

with parameters κ and η. For $\eta > 1$ the corresponding potential is

$$\Phi^\mathrm{IPL} = \frac{\kappa}{(\eta - 1) r^{\eta - 1}}. \tag{3.31}$$

The ratio of the potential energy to the total energy can be written as

$$\frac{\Phi^{\mathrm{IPL}}}{\frac{1}{2}\,m_{\mathrm{r}}\,v_{\mathrm{r}}^2} = \frac{2}{\eta-1}\,\left(\frac{W}{W_0}\right)^{\eta-1} \tag{3.32}$$

where W_0 is a second dimensionless parameter defined by

$$W_0 = b\left(\frac{m_{\mathrm{r}}\,v_{\mathrm{r}}^2}{\kappa}\right)^{\frac{1}{\eta-1}}. \tag{3.33}$$

Equations (3.21) to (3.23) then yield for the deflection angle

$$\chi = \pi - 2\int_0^{W_1} \frac{1}{\sqrt{1-W^2-\frac{2}{\eta-1}\,\left(\frac{W}{W_0}\right)^{\eta-1}}}\,dW \tag{3.34}$$

where W_1 is the positive root of the equation

$$1 - W^2 - \frac{2}{\eta-1}\,\left(\frac{W}{W_0}\right)^{\eta-1} = 0. \tag{3.35}$$

The differential cross section for this model can be calculated from Eqs. (3.33) and (3.25) to

$$\sigma^{\mathrm{IPL}}\,d\Omega = W_0\,\left(\frac{\kappa}{m\,v_{\mathrm{r}}^2}\right)^{\frac{2}{\eta-1}}\,dW_0\,d\phi. \tag{3.36}$$

It is easy to see that the integration of this equation would lead to a divergent total cross section. Therefore, a cutoff is needed, preferably on χ. Since χ depends only on W_0, this is equivalent to setting the cutoff directly on W_0 where the cutoff value shall be denoted by $W_{0,\mathrm{m}}$. So

$$\sigma_{\mathrm{T}}^{\mathrm{IPL}} = \int_0^{2\pi}\int_0^{W_{0,\mathrm{m}}} W_0\,\left(\frac{\kappa}{m\,v_{\mathrm{r}}^2}\right)^{\frac{2}{\eta-1}}\,dW_0\,d\phi = \pi\,W_{0,\mathrm{m}}^2\,\left(\frac{\kappa}{m\,v_{\mathrm{r}}^2}\right)^{\frac{2}{\eta-1}} = \pi\,W_{0,\mathrm{m}}^2\,\left(\frac{\frac{1}{2}\kappa}{E_{\mathrm{t}}}\right)^{\frac{2}{\eta-1}}. \tag{3.37}$$

The important point here is the dependence of σ_{T} on the relative velocity:

$$\sigma_{\mathrm{T}}^{\mathrm{IPL}} \propto \frac{1}{v_{\mathrm{r}}^{\frac{4}{\eta-1}}}. \tag{3.38}$$

This dependence will later be used for the definition of the variable hard sphere model.

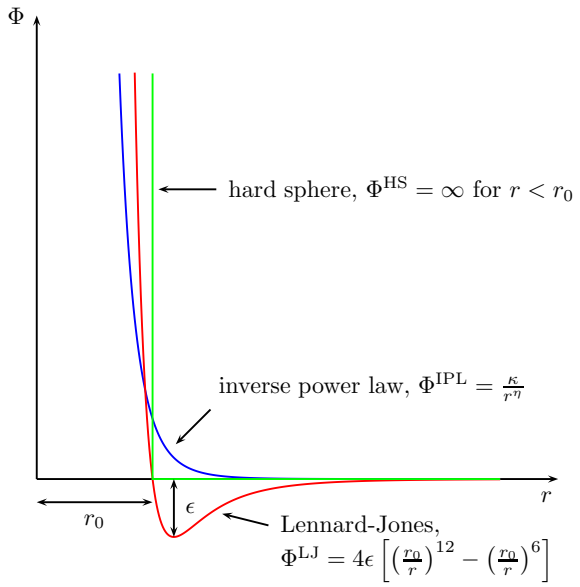

Figure 3.2 – *Simple molecular potentials used for binary collision models.*

Hard Sphere (HS) and Variable Hard Sphere (VHS) Model

In this very simple model the colliding particles are treated as hard spheres. This can be understood as the limiting case of the inverse power law model with $\eta = \infty$ (compare Fig. 3.2). It is clear that the force becomes effective at

$$r_0 = \frac{1}{2}(d_1 + d_2) = d_{12} \tag{3.39}$$

and the apse line is the line through the centers of the spheres. Therefore,

$$b = d_{12}\,\sin\theta_{\mathrm{A}} = d_{12}\,\cos\left(\frac{\chi}{2}\right) \tag{3.40}$$

and

$$\left|\frac{db}{d\chi}\right| = \frac{1}{2}\,d_{12}\,\sin\left(\frac{\chi}{2}\right) \tag{3.41}$$

so that Eq. (3.26) gives

$$\sigma^{\mathrm{HS}} = \frac{1}{4}\,d_{12}^2. \tag{3.42}$$

This equation is independent of χ meaning that all scattering angles are equally likely in the center of mass frame. The total cross section is

$$\sigma_{\mathrm{T}}^{\mathrm{HS}} = \int_0^{4\pi} \sigma^{\mathrm{HS}}\, d\Omega = \pi\, d_{12}^2 \tag{3.43}$$

and the viscosity and diffusion cross sections are

$$\sigma_{\mu}^{\mathrm{HS}} = \frac{2}{3}\sigma_{\mathrm{T}}^{\mathrm{HS}} \tag{3.44}$$

$$\sigma_{\mathrm{D}}^{\mathrm{HS}} = \sigma_{\mathrm{T}}^{\mathrm{HS}}. \tag{3.45}$$

The first approximation to the viscosity coefficient for a monatomic gas in the Chapman-Enskog theory is

$$\mu = \frac{\frac{5}{8}\sqrt{\pi m k_{\mathrm{B}} T}}{\left(\frac{m}{4k_{\mathrm{B}}T}\right)^4 \int_0^{\infty} v_{\mathrm{r}}^7\, \sigma_{\mu}\, e^{-\frac{m v_{\mathrm{r}}^2}{4k_{\mathrm{B}}T}}\, dv_{\mathrm{r}}} \tag{3.46}$$

which yields

$$\mu = \frac{5}{16}\,\frac{\sqrt{\pi m k_{\mathrm{B}} T}}{\sigma_{\mathrm{T}}^{\mathrm{HS}}} = \frac{5}{16}\sqrt{\frac{m k_{\mathrm{B}} T}{\pi}}\,\frac{1}{d^2} \tag{3.47}$$

for the hard sphere model. One can see that $\mu \propto \sqrt{T}$ in this case whereas for real gases the dependence is more like $\mu = \propto T^{\frac{3}{4}}$. Empirically it was found that a successful molecular model has to reproduce the viscosity coefficient of a gas and the dependence of it on temperature. The choice of the potential on which the model is based on was found to be much less important. The failure of the hard sphere model to describe the temperature dependence of the viscosity effectively prohibits its usage but the aforementioned facts led Bird ([65]) to the proposition of the variable hard sphere model which is a simple extension of the hard sphere model. It uses the same isotropic distribution for the scattering angle χ as the hard sphere model such that

$$\chi = 2\arccos\left(\frac{b}{d}\right) \tag{3.48}$$

and the nominal cross sections are taken from the hard sphere model as well:

$$\sigma_{\mathrm{T}}^{\mathrm{VHS}} = \pi d^2 \tag{3.49}$$

$$\sigma_{\mu}^{\mathrm{VHS}} = \frac{2}{3}\pi d^2 \tag{3.50}$$

$$\sigma_{\mathrm{D}}^{\mathrm{VHS}} = \pi d^2. \tag{3.51}$$

But now the diameter is allowed to vary with the collision energy. Usually it is chosen such that the temperature dependence of the viscosity matches that of the inverse power law model. Comparing viscosity and diffusion cross sections of both models yields

$$d^2 \propto E_\mathrm{t}^{-\nu} \tag{3.52}$$

with $\nu = \frac{2}{\eta-1}$. For practical usage one has to find or eliminate the constants κ and η of the inverse power law model. To this end one can use Eq. (3.47) which yields a diameter at some reference temperature and viscosity of

$$d_\mathrm{ref} = \sqrt{\frac{5}{16\,\mu_\mathrm{ref}}\sqrt{\frac{m k_\mathrm{B} T_\mathrm{ref}}{\pi}}}. \tag{3.53}$$

These reference values can be taken from experimental data as well as the dependence of μ on T. With the reference values it is

$$\frac{\sigma_\mathrm{T}^\mathrm{VHS}}{\sigma_\mathrm{T,ref}^\mathrm{VHS}} = \left(\frac{d}{d_\mathrm{ref}}\right)^2 = \left(\frac{v_\mathrm{r}}{v_\mathrm{r,ref}}\right)^{-2\nu} = \left(\frac{E_\mathrm{t}}{E_\mathrm{t,ref}}\right)^{-\nu}. \tag{3.54}$$

Using this and Eqs. (3.50) and (3.46) one obtains

$$\mu = \frac{\frac{15}{8}\sqrt{\pi m k_\mathrm{B}}\left(\frac{4k_\mathrm{B}}{m}\right)^\nu T^\omega}{\Gamma(4-\nu)\sigma_\mathrm{T,ref}^\mathrm{VHS}\, v_\mathrm{r,ref}^{2\nu}} \tag{3.55}$$

with

$$\omega = \nu + \frac{1}{2} = \frac{1}{2}\frac{\eta+3}{\eta-1} \tag{3.56}$$

yielding

$$d = \sqrt{\frac{\frac{15}{8}\sqrt{\frac{m}{\pi}}\left(k_\mathrm{B}T_\mathrm{ref}\right)^\omega}{\Gamma\left(\frac{9}{2}-\omega\right)\,\mu_\mathrm{ref}\,E_\mathrm{t}^{\omega-\frac{1}{2}}}}. \tag{3.57}$$

This procedure is suitable for a pure gas. If one wants to consider a gas mixture it may be more practical to use reference diffusion values for this procedure and one can obtain for a two component gas mixture

$$d_{12} = \sqrt{\frac{\frac{3}{8}(2k_\mathrm{B}T_\mathrm{ref})^{\omega_{12}}}{\Gamma\left(\frac{7}{2}-\omega_{12}\right)\,\sqrt{\pi\, m_\mathrm{r}}\, n\, D_{12,\mathrm{ref}}\,(2E_\mathrm{t})^{\omega_{12}-\frac{1}{2}}}}. \tag{3.58}$$

If such cross collision data is not available, one may want to use approximated values, e.g. the averages of diameters and temperature exponents, i.e.

$$d_{12} = \frac{1}{2}(d_1 + d_2) \tag{3.59}$$

$$\omega_{12} = \frac{1}{2}(\omega_1 + \omega_2). \tag{3.60}$$

Variable Soft Sphere (VSS) Model

The VHS model is already much more useful than the hard sphere model but one can find that in the VHS model it is

$$\frac{\sigma_\mu^{\mathrm{VHS}}}{\sigma_{\mathrm{D}}^{\mathrm{VHS}}} = \frac{2}{3} \tag{3.61}$$

like in the hard sphere model while in the inverse power law model $\frac{\sigma_\mu^{\mathrm{IPL}}}{\sigma_{\mathrm{D}}^{\mathrm{IPL}}}$ is a function of η and the ratio is not equal to $\frac{2}{3}$ except for $\eta = \infty$, i.e. if the parameters of the VHS model are matched with viscosity data, diffusion is expected to be off and vice versa. In order to match this ratio, the variable soft sphere (VSS) model was introduced by Koura and Matsumoto ([66]). Here, the law for the scattering angle is modified and can be written as

$$\chi = 2 \arccos\left(\left(\frac{b}{d}\right)^{\frac{1}{\alpha}}\right). \tag{3.62}$$

Since normal values for α lie in the range between 1 and 2, the scattering angle for this model is smaller than for the VHS model and this circumstance leads to the name „soft sphere“. Note that for $\alpha = 1$ the VSS and the VHS model are equivalent. The definite cross sections can be written as

$$\sigma_{\mathrm{T}}^{\mathrm{VSS}} = \pi d^2 \tag{3.63}$$

$$\sigma_\mu^{\mathrm{VSS}} = \frac{2}{3} S_\mu \pi d^2 \tag{3.64}$$

$$\sigma_{\mathrm{D}}^{\mathrm{VSS}} = S_{\mathrm{D}} \pi d^2 \tag{3.65}$$

with the softness coefficients

$$S_\mu = \frac{6\alpha}{(\alpha+1)(\alpha+2)} \tag{3.66}$$

$$S_{\mathrm{D}} = \frac{2}{\alpha+1}. \tag{3.67}$$

If the model is based on the inverse power law model, the diameter d of a particle has the same energy dependence as for the VHS model:

$$\pi d^2 = const \cdot E_{\mathrm{t}}^{-\nu} \tag{3.68}$$

with $\nu = \frac{2}{\eta - 1}$. For the exponent α one gets

$$\alpha = \frac{1}{\frac{\sigma_{\mathrm{D}}^{\mathrm{VSS}}}{\sigma_{\mu}^{\mathrm{VSS}}} - \frac{1}{2}} \tag{3.69}$$

and the viscosity and diffusion coefficients of the Chapman-Enskog theory become modified by the respective softness factors

$$\mu_{\mathrm{VSS}} = \frac{\mu_{\mathrm{VHS}}}{S_{\mu}} \tag{3.70}$$

$$D_{\mathrm{VSS}} = \frac{D_{\mathrm{VHS}}}{S_{\mathrm{D}}}. \tag{3.71}$$

VSS Model based on Lennard-Jones-Potential

So far, only models based on the simple inverse power law potential have been considered. The corresponding force is repulsive for all distances between the collision partners. However, in reality, the force is usually composed of a long range attractive and a short range repulsive term and the attractive term may change the scattering behavior at very low collision energies, i.e. at low temperatures. One of the most famous models describing the interaction between neutral, non-polar molecules is the Lennard-Jones-Potential

$$\Phi^{\mathrm{LJ}} = 4\epsilon_{\mathrm{LJ}} \left[\left(\frac{d_{\mathrm{LJ}}}{r} \right)^{12} - \left(\frac{d_{\mathrm{LJ}}}{r} \right)^{6} \right] \tag{3.72}$$

(compare Fig. 3.2) where the r^{-6}-term describes the Van-der-Waals attraction and the r^{-12}-term models the Pauli-repulsion due to overlapping of the electron orbitals at short distances. From this one can expect the model to work best for noble gases. Unfortunately, the model is very complicated to handle if one wants to calculate the scattering angle or the collision cross section which is the reason why it is not as commonly used as the inverse power law potential. However, the calculations have been performed numerically and some fitfunctions are provided in [67]. These can be used for the easy incorporation of the Lennard-Jones potential into the VSS model. The nominal cross section is

$$\sigma_{\mathrm{LJ}} = \pi d_{\mathrm{LJ}}^2 \tag{3.73}$$

and the exponent for the scattering angle is [66]

$$\alpha = \frac{1}{\frac{\sigma_{\mathrm{D}}^*}{\sigma_\mu^*} - \frac{1}{2}} = \frac{1}{\frac{\sigma_{\mathrm{D}}^{\mathrm{LJ}}}{\sigma_\mu^{\mathrm{LJ}}} - \frac{1}{2}} \tag{3.74}$$

where $\sigma_\mu^* = \frac{\sigma_\mu^{\mathrm{LJ}}}{\sigma_{\mathrm{LJ}}}$ and $\sigma_{\mathrm{D}}^* = \frac{\sigma_{\mathrm{D}}^{\mathrm{LJ}}}{\sigma_{\mathrm{LJ}}}$ are the reduced viscosity and diffusion cross sections. Fit functions for the two "reduced cross sections" $S^{(l)}(K)$ are given in [67] and are defined as

$$S^{(l)}(K) = \frac{4}{2 - \frac{1+(-1)^l}{1+l}} \int_0^\infty \left(1 - \cos^l\chi\right) \beta \, d\beta. \tag{3.75}$$

They are of the form

$$S^{(l)}(K) = \sum_i A_i^{(l)} \, K^{n_i^{(l)}} \, e^{-a_i^{(l)} K} \tag{3.76}$$

with $l = 1$ for diffusion and $l = 2$ for viscosity as well as $K = \frac{\frac{1}{2} m_{\mathrm{r}} v_{\mathrm{r}}^2}{\epsilon_{\mathrm{LJ}}}$ and the respective coefficients given in tables 3.1 and 3.2.

Table 3.1 – *Coefficients for fitfunction of $S^{(1)}(K)$ for Lennard-Jones potential*

i	$A_i^{(1)}$	$n_i^{(1)}$	$a_i^{(1)}$
1	1.212	-0.155	0
2	1.0782	-0.5	0.439
3	19.49	2.126	3.675
4	$-4.255 \cdot 10^{81}$	154	194
5	$-1.677 \cdot 10^{16}$	23.4	45
6	$1.705 \cdot 10^{5}$	4.425	29.1

Table 3.2 – *Coefficients for fitfunction of $S^{(2)}(K)$ for Lennard-Jones potential*

i	$A_i^{(2)}$	$n_i^{(2)}$	$a_i^{(2)}$
1	1.3719	-0.145	0
2	1.1812	-0.789	0.19
3	2.724	-0.697	0.67742
4	-899.76	3.01	6.7461
5	$3.162 \cdot 10^{56}$	118.6	132.2
6	$9.3325 \cdot 10^{188}$	363.5	443.25
7	$8.4723 \cdot 10^{54}$	82.51	140.42
8	$-5.0315 \cdot 10^{6}$	3.958	35.989

It should be mentioned that for numerical reasons it is much more convenient to write Eq. (3.76) as

$$S^{(l)}(K) = \sum_i e^{\ln(A_i^{(l)}) + \ln(K) \cdot n_i^{(l)} - a_i^{(l)} K} \tag{3.77}$$

and although the $S^{(l)}(K)$ are called "reduced cross sections", starting from Eqs. (3.29) and (3.28) and using Eq. (3.25), (3.75) and $\beta = \frac{b}{d_{\mathrm{LJ}}}$ one finds

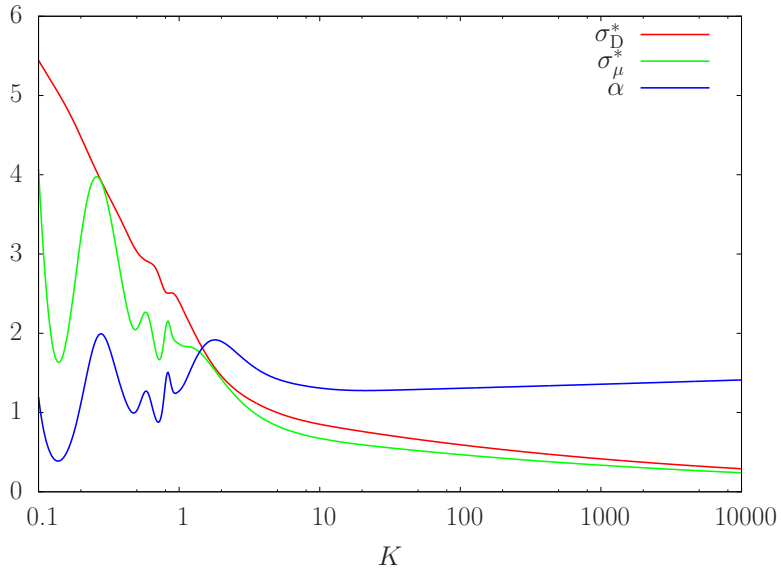

Figure 3.3 – *Reduced cross sections and exponent of the scattering angle for the VSS model based on the Lennard-Jones potential as a function of the reduced collision energy* $K = \frac{\frac{1}{2} m_r v_r^2}{\epsilon_{LJ}}$.

$$\sigma^*_D = S^{(1)}(K) \tag{3.78}$$

$$\sigma^*_\mu = \frac{2}{3} S^{(2)}(K) \tag{3.79}$$

These reduced cross sections together with the exponent for the scattering angle α are shown in Fig. 3.3. For the reduced total cross section one then obtains

$$\sigma^*_T = \frac{3}{2S_\mu}\,\sigma^*_\mu = \frac{\pi d^2}{\sigma_{LJ}} = \left(\frac{d}{d_{LJ}}\right)^2 \tag{3.80}$$

If the collision of two different particle species is considered, one may approximate the relevant cross collision values by

$$d_{LJ,12} = \frac{1}{2}(d_{LJ,1} + d_{LJ,2}) \tag{3.81}$$

$$\epsilon_{LJ,12} = \sqrt{\epsilon_{LJ,1} \cdot \epsilon_{LJ,2}} \tag{3.82}$$

3.3.2 Inelastic Collisions

So far only elastic collisions have been studied where the total kinetic energy of the collision partners is conserved during the collision. This is not the case if inelastic

collisions are considered where there is an exchange of kinetic energy E_t and internal energy E_i that may be stored in rotational or vibrational excitations. There are several different models which describe the mechanism of energy exchange more or less realistically. Many of them suffer from complex calculations or problematic results. One widely used phenomenological method is the Larsen-Borgnakke model in which only a fraction Λ of the collisions is regarded as inelastic. This allows Λ to be used as control parameter to steer the relaxation rate. Further, it has a low computational cost. A similar model which considers all collisions as inelastic but with a restricted energy exchange is less efficient and was found to lead not to exact equipartition of energy for internal and translational degrees of freedom whereas the Larsen-Borgnakke model satisfies this requirement. Another advantage of this model is, that it redistributes energy between the molecules and internal and translational modes but otherwise does not affect the scattering behavior, i.e. the scattering angle is not affected in inelastic collisions. This feature makes it easy to combine this model with the previously discussed collision models and therefore it will be the only model for inelastic collisions used in this work.

Larsen-Borgnakke Model

The average over an arbitrary quantity Q of a gas that depends only on velocity can be calculated by

$$\overline{Q} = \int_{-\infty}^{\infty} Q f \, d\vec{v} \tag{3.83}$$

where f is the velocity distribution function. For a gas in equilibrium this is given by

$$f = \frac{\beta^3}{\pi^{\frac{3}{2}}} e^{-\beta^2 v'^2} \tag{3.84}$$

where the thermal velocity $\vec{v}'$ is given by the difference of the particle velocity $\vec{v}$ and the stream velocity $\vec{v}_0$ and with $\beta = \sqrt{\frac{m}{2k_\mathrm{B}T}}$ being the inverse of the most probable molecular thermal speed. For example, the average of the j-th power of the relative velocity $v_\mathrm{r} = |\vec{v}_1 - \vec{v}_2|$ of two particles can be written as

$$\overline{v_\mathrm{r}^j} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} v_\mathrm{r}^j f_1 f_2 \, d\vec{v}_1 d\vec{v}_2. \tag{3.85}$$

If the integration variables are changed from $\vec{v}_1$ and $\vec{v}_2$ to $\vec{v}_\mathrm{m}$ and $\vec{v}_\mathrm{r}$ and the gas is in equilibrium, this can be written as

$$\overline{v_\mathrm{r}^j} = \frac{2(m_1 m_2)^{\frac{3}{2}}}{\pi (k_\mathrm{B}T)^3} \int_0^{\infty} v_\mathrm{m}^2 e^{-\frac{-(m_1+m_2)v_\mathrm{m}^2}{2k_\mathrm{B}T}} \, dv_\mathrm{m} \cdot \int_0^{\infty} v_\mathrm{r}^{j+2} e^{-\frac{m_\mathrm{r} v_\mathrm{r}^2}{2k_\mathrm{B}T}} \, dv_\mathrm{r} \tag{3.86}$$

$$= \frac{2}{\sqrt{\pi}} \Gamma\left(\frac{j+3}{2}\right) \left(\frac{2k_\mathrm{B}T}{m_\mathrm{r}}\right)^{\frac{j}{2}}. \tag{3.87}$$

For $j = 1$ this yields $\overline{v_\mathrm{r}} = \frac{2}{\sqrt{\pi}}\sqrt{\frac{2k_\mathrm{B}T}{m_\mathrm{r}}}$ which is for a simple gas with the reduced mass $m_\mathrm{r} = \frac{m}{2}$

$$\overline{v_\mathrm{r}} = 2\sqrt{\frac{2}{\pi}}\frac{1}{\beta} = \sqrt{2}\cdot\overline{v'}. \tag{3.88}$$

For particles with a fixed cross section this is also the mean relative speed in a collision. If the cross section depends on the relative speed of the colliding particles, the result will change and one has to include σ_T in the calculation, e.g. for the mean collision frequency

$$\nu = n\overline{\sigma_\mathrm{T} v_\mathrm{r}} \tag{3.89}$$

where $\overline{\sigma_\mathrm{T} v_\mathrm{r}}$ is given by

$$\overline{\sigma_\mathrm{T} v_\mathrm{r}} = \sqrt{\frac{2}{\pi}}\frac{m_\mathrm{r}}{k_\mathrm{B}T}\int_0^\infty \sigma_\mathrm{T} v_\mathrm{r}^3 e^{-\frac{m_\mathrm{r} v_\mathrm{r}^2}{2k_\mathrm{B}T}}\, dv_\mathrm{r}. \tag{3.90}$$

The mean of a quantity over all collisions is then given by

$$\overline{Q} = \frac{\int_0^\infty Q\sigma_\mathrm{T} v_\mathrm{r}^3 e^{-\frac{m_\mathrm{r} v_\mathrm{r}^2}{2k_\mathrm{B}T}}\, dv_\mathrm{r}}{\int_0^\infty \sigma_\mathrm{T} v_\mathrm{r}^3 e^{-\frac{m_\mathrm{r} v_\mathrm{r}^2}{2k_\mathrm{B}T}}\, dv_\mathrm{r}} \tag{3.91}$$

$$= \sqrt{\frac{2}{\pi}}\left(\frac{m_\mathrm{r}}{k_\mathrm{B}T}\right)^{\frac{3}{2}}\frac{1}{\overline{\sigma_\mathrm{T} v_\mathrm{r}}}\int_0^\infty Q\sigma_\mathrm{T} v_\mathrm{r} e^{-\frac{m_\mathrm{r} v_\mathrm{r}^2}{2k_\mathrm{B}T}}\, dv_\mathrm{r}. \tag{3.92}$$

For example, the mean kinetic energy $\overline{E_\mathrm{t}}$ of the particles in the center of mass system can be obtained by setting $Q = \frac{1}{2}m_\mathrm{r} v_\mathrm{r}^2$. Equation (3.92) can also be written in terms of the kinetic energy to give

$$\overline{Q} \propto \int_0^\infty Q\sigma_\mathrm{T}(E_\mathrm{t}) E_\mathrm{t} e^{-\frac{E_\mathrm{t}}{k_\mathrm{B}T}}\, dE_\mathrm{t} \tag{3.93}$$

from which one can deduce the distribution function of E_t through comparison with Eq. (3.83). The result is

$$f_{E_\mathrm{t}} \propto \sigma_\mathrm{T}(E_\mathrm{t}) E_\mathrm{t} e^{-\frac{E_\mathrm{t}}{k_\mathrm{B}T}}. \tag{3.94}$$

The total internal energy of the two collision partners is

$$E_\mathrm{int} = E_{\mathrm{int},1} + E_{\mathrm{int},2} \tag{3.95}$$

and the distribution function of the internal energy $E_{\mathrm{int},1}$ for a molecule with ζ internal degrees of freedom is

$$f_{E_{\mathrm{int},1}} \propto E_{\mathrm{int},1}^{\frac{\zeta}{2}-1} e^{-\frac{E_{\mathrm{int},1}}{k_{\mathrm{B}}T}}. \tag{3.96}$$

From this one can derive the distribution function for the total internal energy which is

$$f_{E_{\mathrm{int}}} \propto E_{\mathrm{int}}^{\zeta-1} e^{-\frac{E_{\mathrm{int}}}{k_{\mathrm{B}}T}}. \tag{3.97}$$

The total energy E_{c} in a collision is

$$E_{\mathrm{c}} = E_{\mathrm{t}} + E_{\mathrm{int}}. \tag{3.98}$$

The probability density of E_{t} and E_{int} is proportional to the product of $f_{E_{\mathrm{t}}}$ and $f_{E_{\mathrm{int}}}$, i.e.

$$f_{E_{\mathrm{t}}} f_{E_{\mathrm{int}}} \propto \sigma_{\mathrm{T}}(E_{\mathrm{t}}) E_{\mathrm{t}} E_{\mathrm{int}}^{\zeta-1} e^{-\frac{E_{\mathrm{t}}+E_{\mathrm{int}}}{k_{\mathrm{B}}T}} \tag{3.99}$$

or using Eq. (3.98)

$$f_{E_{\mathrm{t}}} f_{E_{\mathrm{int}}} \propto \sigma_{\mathrm{T}}(E_{\mathrm{t}}) E_{\mathrm{t}} (E_{\mathrm{c}} - E_{\mathrm{t}})^{\zeta-1} e^{-\frac{E_{\mathrm{c}}}{k_{\mathrm{B}}T}}. \tag{3.100}$$

The effective temperature T is defined by the total collision energy E_{c} and hence the exponential term may be regarded as constant. Then, the probability P of a certain value of the translational energy is

$$P = C\sigma_{\mathrm{T}}(E_{\mathrm{t}}) E_{\mathrm{t}} (E_{\mathrm{c}} - E_{\mathrm{t}})^{\zeta-1} \tag{3.101}$$

where C is a normalization constant. Sampling from this distribution is difficult, but if the maximum value $P_{\max}$ of this function can be calculated, the acceptance-rejection method can be used. For the VHS and VSS models based on the inverse power law potential one obtains

$$\frac{P}{P_{\max}} = \left(\frac{\zeta + \frac{1}{2} - \omega}{\frac{3}{2} - \omega} \frac{E_{\mathrm{t}}}{E_{\mathrm{c}}} \right)^{\frac{3}{2}-\omega} \left(\frac{\zeta + \frac{1}{2} - \omega}{\zeta - 1} \left(1 - \frac{E_{\mathrm{t}}}{E_{\mathrm{c}}} \right) \right)^{\zeta-1}. \tag{3.102}$$

With this relation, one can randomly choose a value for the post-collision translational energy $0 \leq E_{\mathrm{t}}' \leq E_{\mathrm{c}}$ and compare it with a random number $R \in [0, 1]$. If $R < \frac{P}{P_{\max}}$ the value for E_{t}' is accepted and otherwise rejected and a new value for E_{t}' is selected and tested until a value is accepted. Now, the post-collision internal energy $E_{\mathrm{int}}' = E_{\mathrm{c}} - E_{\mathrm{t}}'$ has to be distributed between the two molecules. This can be done in an equivalent way to the preceding analysis using Eq. (3.96) and by noting that

$$f_{E_{\mathrm{int},1}} f_{E_{\mathrm{int},2}} = f_{E_{\mathrm{int},1}} f_{(E_{\mathrm{int}}-E_{\mathrm{int},1})} \propto E_{\mathrm{int},1}^{\frac{\zeta}{2}-1} (E_{\mathrm{int}} - E_{\mathrm{int},1})^{\frac{\zeta}{2}-1}. \tag{3.103}$$

From this one obtains for $E'_{\mathrm{int},1}$

$$\frac{P}{P_{\max}} = 2^{\zeta-2} \left(\frac{E'_{\mathrm{int},1}}{E'_{\mathrm{int}}} \right)^{\frac{\zeta}{2}-1} \left(1 - \frac{E'_{\mathrm{int},1}}{E'_{\mathrm{int}}} \right)^{\frac{\zeta}{2}-1}. \tag{3.104}$$

Finally, the internal energy for the second molecule will be given by

$$E'_{\mathrm{int},2} = E_{\mathrm{int}} - E'_{\mathrm{int},1}. \tag{3.105}$$

So far, only collisions of one particle species have been considered. Different molecules can have a different number of internal degrees of freedom and different collision relaxation rates. The different number of degrees of freedom can easily be accounted for in the above formulas by using ζ_1 for molecule 1 and ζ_2 for molecule 2. The analysis then is completely equivalent except that some singularities have to be handled which is not important for understanding the method and therefore omitted here (see [59]). Accounting for different relaxation rates will be described shortly. First the general Larsen-Borgnakke distribution used in the code will be described. For that a parameter Ξ is defined as the sum of the average degrees of freedom:

$$\Xi = \frac{\zeta_{\mathrm{t}}}{2} + \frac{\zeta_{\mathrm{rot},1}}{2} + \frac{\zeta_{\mathrm{rot},2}}{2} + \frac{\sum \zeta_{\mathrm{vib},1}}{2} + \frac{\sum \zeta_{\mathrm{vib},2}}{2} \tag{3.106}$$

where the indices 1 and 2 are for the two particle species and ζ_{rot} stands for the rotational degrees of freedom while ζ_{vib} denotes the vibrational ones and ζ_{t} for the translational ones. ζ_{t} can be obtained by using the equipartition theorem

$$\overline{E_{\mathrm{t}}} = \frac{\zeta_{\mathrm{t}}}{2} k_{\mathrm{B}} T \tag{3.107}$$

and $\overline{E_{\mathrm{t}}}$ can be calculated by setting $Q = \frac{1}{2} m_{\mathrm{r}} v_{\mathrm{r}}^2$ in Eq. (3.92). For the VHS and VSS model based on the inverse power law this yields

$$\frac{\zeta_{\mathrm{t}}}{2} = \frac{5}{2} - \omega_{12} \tag{3.108}$$

and for the VSS model based on the Lennard-Jones potential it is

$$\frac{\zeta_{\mathrm{t}}}{2} = \frac{\epsilon_{\mathrm{LJ},12}}{k_{\mathrm{B}} T} \frac{\sum_i \frac{A_i^{(2)} \Gamma\left(3+n_i^{(2)}\right)}{\left(\frac{\epsilon_{\mathrm{LJ},12}}{k_{\mathrm{B}} T} + a_i^{(2)}\right)^{3+n_i^{(2)}}}}{\sum_i \frac{A_i^{(2)} \Gamma\left(2+n_i^{(2)}\right)}{\left(\frac{\epsilon_{\mathrm{LJ},12}}{k_{\mathrm{B}} T} + a_i^{(2)}\right)^{2+n_i^{(2)}}}} \tag{3.109}$$

with the parameters from Tab. 3.2. The rotational degrees of freedom become excited at low temperatures, generally below a few dozen degrees Kelvin, and are usually considered fully excited. The vibrational degrees of freedom become excited only above a few thousand degrees and in most cases cannot be considered fully excited. Therefore they need special treatment but since the characteristic temperature for H_2 is $6159\,\mathrm{K}$ which is much higher than the temperatures in the ABS, the vibrational modes can be neglected here.

Now, let Ξ_a be one or more terms of Eq. (3.106) and Ξ_b the remaining degrees of freedom that take part in the division of energy. Let E_a be the energy of the first group of modes and E_b the energy of the second group. Then, the available energy for redistribution is $E_\mathrm{c} = E_\mathrm{a} + E_\mathrm{b}$ and the distribution functions for the energies are

$$f\left(\frac{E_\mathrm{a}}{E_\mathrm{c}}\right) = f\left(\frac{E_\mathrm{b}}{E_\mathrm{c}}\right) = \frac{\Gamma(\Xi_\mathrm{a}+\Xi_\mathrm{b})}{\Gamma(\Xi_\mathrm{a})\Gamma(\Xi_\mathrm{b})}\left(\frac{E_\mathrm{a}}{E_\mathrm{c}}\right)^{\Xi_\mathrm{a}-1}\left(\frac{E_\mathrm{b}}{E_\mathrm{c}}\right)^{\Xi_\mathrm{b}-1}. \tag{3.110}$$

From that one can derive the probability of a particular value of E_a compared to the maximum probability to be used with the acceptance-rejection method:

$$\frac{P}{P_\mathrm{max}} = \left(\frac{\Xi_\mathrm{a}+\Xi_\mathrm{b}-2}{\Xi_\mathrm{a}-1}\,\frac{E_\mathrm{a}}{E_\mathrm{c}}\right)^{\Xi_\mathrm{a}-1}\left(\frac{\Xi_\mathrm{a}+\Xi_\mathrm{b}-2}{\Xi_\mathrm{b}-1}\left(1-\frac{E_\mathrm{a}}{E_\mathrm{c}}\right)\right)^{\Xi_\mathrm{b}-1}. \tag{3.111}$$

This general Larsen-Borgnakke distribution contains all previous distributions as special cases and is easier to handle. It is also useful if one wants to consider different relaxation rates for different gas components. The relaxation rate in the Larsen-Borgnakke model is controlled via the fraction Λ of inelastic collisions. Therefore different relaxation rates are only possible with different Λ's for different particle species. However, if the whole collision is considered as inelastic, i.e. if the energy is redistributed between translational and internal degrees of freedom of both molecules, this formalism is not possible in cross-collisions. In order to resolve this problem, the energy can get subsequently distributed between translational and internal degrees of freedom for molecule i if a random number $R_i \in [0,1]$ is bigger than Λ_i. It was also shown in [59] that this method leads to the expected result whereas the energy distribution between both molecules at once only agrees qualitatively but with another relaxation rate.

Now, only the parameters Λ_i have to be determined. Usually the rate of approach to equilibrium of the internal energy of a gas is modeled by the following rate equation

$$\frac{dE_\mathrm{int}}{dt} = \frac{E_\mathrm{int}(T_\mathrm{tr}) - E_\mathrm{int}(t)}{\tau_i} \tag{3.112}$$

where $E_{\mathrm{int}}(T_{\mathrm{tr}}) = \frac{\zeta}{2k_{\mathrm{B}}T_{\mathrm{tr}}}$ is the instantaneous equilibrium internal energy with ζ internal degrees of freedom and the time-dependent translational temperature $T_{\mathrm{tr}}(t)$. τ_i is the characteristic relaxation time of the system. It is customary to express τ_i as

$$\tau_i = Z_i \tau_{\mathrm{c}} \tag{3.113}$$

where τ_{c} is the mean collision time, which is a function of $T_{\mathrm{tr}}(t)$ as well. The collision relaxation number Z_i is the average number of collisions per molecule that is necessary to bring the gas back to equilibrium. Then, the fraction of inelastic collisions in the Larsen-Borgnakke model can be written as

$$\Lambda = \frac{\tau_{\mathrm{c}}}{\tau_i} = \frac{1}{Z_i}. \tag{3.114}$$

Z_i is also dependent on the temperature and generally increases with temperature. In this work, hydrogen is simulated and this gas has quite unusual properties since the molecules are so light. This leads to a very low moment of inertia I which in turn leads to a high characteristic rotational temperature $T_{\mathrm{rot,c}} = \frac{\hbar^2}{2k_{\mathrm{B}}I}$. For H_2 this is $85.4\,\mathrm{K}$, for D_2 it is $43\,\mathrm{K}$ and it reduces further with mass and yields $0.35\,\mathrm{K}$ for Cl_2. Therefore, the energy levels are widely spaced in hydrogen and only the lowest energy levels are excited at room temperature. For temperatures below the characteristic temperature the molecules behave like atoms, i.e. they are mainly in the rotational ground state and do not take part in the energy redistribution. Calculations in [68] of Z_i for para-hydrogen show a steep increase of Z_i for temperatures below $300\,\mathrm{K}$ and a gradual increase for higher temperatures. This general behavior is expected for other molecules as well, but with the minimum shifted to much lower temperatures and possibly below the boiling point. It was further found that Z_i depends on the distance from equilibrium and the direction to equilibrium. Generally it holds $Z_i(\Delta T < 0) > Z_i(\Delta T > 0)$ with $\Delta T = T_{\mathrm{rot},0} - T_{\mathrm{tr},0}$, i.e. the temperature difference between internal and translational degrees of freedom at $t = 0$. Additionally, higher rotational excitation levels decay slower than lower ones, which leads to an energy level distribution that is not the Boltzmann distribution and makes the system depend on its history. This also has an effect on the measured value that therefore depends on the measurement method. Measured values, e.g. in [69] and [70] agree well with the calculations of [68] but others like [71] which agrees qualitatively but with lower Z_i and a higher temperature for the minimum, differ significantly. Minimum values for H_2 at around $300\,\mathrm{K}$ for gas expansion are around $Z_i = 300$ and increase with temperature to $Z_i(T \approx 1000\,\mathrm{K}) \approx 500$. For temperatures below room temperature, the value increases rapidly to around $Z_i(T \approx 100\,\mathrm{K}) \approx 700$. D_2 needs only about half that number of collisions to relax and HD needs only a tenth of the H_2 value. These are still very high numbers for the current simulation and no significant effect can be expected from inelastic collisions, i.e. the rotational temperature will stay fixed after the nozzle which was also found in [72]. In simulations where the effects of rotational relaxation matter it was found in [73] that a suitably chosen constant Z_i can yield relatively good results

in DSMC calculations for nitrogen where Z_i is about two orders of magnitude smaller than for hydrogen and already a simple model for the temperature dependence of Z_i can drastically improve the results.

3.3.3 Selection of Collision Partners in DSMC Calculations

As mentioned above, there exist various collision selection mechanisms to reduce the mean collision distance. Here, only the method used in the DSMC94 method will be described. The probability P of two simulated particles that represent F_N real particles to collide in a cell of volume V_c is proportional to the volume swept over by their total cross section moving at the relative speed v_r of the two particles:

$$P = \frac{F_\mathrm{N}\sigma_\mathrm{T}v_\mathrm{r}\Delta t}{V_\mathrm{c}}. \tag{3.115}$$

Testing all possible collision pairs in a cell is very inefficient for a larger number of particles per cell. To circumvent this problem, Bird proposed the no-time counter (NTC) method [74]. In this method, the number N_s of selected collision pairs is calculated before the selection and is

$$N_\mathrm{s} = \frac{1}{2}\frac{N\overline{N}F_\mathrm{N}(\sigma_\mathrm{T}v_\mathrm{r})_\mathrm{max}\Delta t}{V_\mathrm{c}} \tag{3.116}$$

where $\overline{N}$ is the time averaged number of simulated particles per cell and $(\sigma_\mathrm{T}v_\mathrm{r})_\mathrm{max}$ is the maximum value of $\sigma_\mathrm{T}v_\mathrm{r}$ in a cell. Using this value, one can also write the maximum probability of a collision between two particles in the cell as

$$P_\mathrm{max} = \frac{F_\mathrm{N}(\sigma_\mathrm{T}v_\mathrm{r})_\mathrm{max}\Delta t}{V_\mathrm{c}}. \tag{3.117}$$

Using this, N_s possible collision pairs are selected and the collision is performed with probability

$$P = \frac{\sigma_\mathrm{T}v_\mathrm{r}}{(\sigma_\mathrm{T}v_\mathrm{r})_\mathrm{max}}. \tag{3.118}$$

This method is linear in N_c and since $(\sigma_\mathrm{T}v_\mathrm{r})_\mathrm{max}$ occurs in the numerator of Eq. (3.116) and in the denominator of Eq. (3.118) the result does not depend on its exact value. The only drawback of this method is, that the values for $(\sigma_\mathrm{T}v_\mathrm{r})_\mathrm{max}$ have to be stored for every cell and should be updated during the simulation. If subcells are used for the selection of collision pairs, the pairs are first selected from within one subcell but if there is only one particle in a subcell, a second collision partner will be chosen either from neighboring subcells or from the whole cell.

4 OpenFOAM

The whole work is based on OpenFOAM version 1.7.1. This is a fully parallelized open source C++ library which is freely available under the *GNU general public license* from [75]. It is made for Linux operating systems and installation instructions can be found on the same web page[1]. The source pack is delivered with a user manual [76] and a short programmer's guide [77]. Additional information can be found at [78].

OpenFOAM, previously FOAM for Field Operation And Manipulation, was developed as a solver for computational continuum mechanics problems but could be used as a solver for all systems of time-dependent partial differential equations [79], [80]. It makes heavy use of the object oriented programming features of C++ to make code fragments reusable, hide low level code from top level applications and to provide easy interfaces for new developments. It also provides a metalanguage for the user to provide partial differential equations in an intuitive way to the program. This design around a common central infrastructure with simple interfaces as well as the open source nature of the code led to the development of many different additional solvers and features over time. Beginning with OpenFOAM version 1.6 a solver for DSMC problems called *dsmcFoam* implementing Birds DSMC94 method is included and successful testing is reported in [81].

4.1 Description of the Common Infrastructure

OpenFOAM makes heavy use of the full functionality of C++. The program is centered around a generic core that provides a common infrastructure which can be interfaced by the various solvers. The core itself consists of many classes with a complicated inheritance structure and intensive use of templates. The low level structure is completely transparent for the top levels and in general does not need to be considered by an application programmer. In this section some of the frequently used parts of the OpenFOAM infrastructure shall be explained and some terminology is introduced.

[1] Note that as a basis for the further work the source pack has to be installed which requires the additional Linux packages *flex, cmake* and *binutils-dev*. The recommended compiler version is gcc-4.4.x but with some additional work (including many "this->" statements in the source code and linking additional libraries) it was also compiled successfully with gcc-4.7.1 under OpenSuse 12.2.

4.1.1 Mesh

One of the most important preconditions for numerically solving a system of partial differential equations or to perform a DSMC calculation is a mesh. This term describes the entity formed by all points, lines, faces, cells and groups thereof and splits the computational volume into small cells. An example is shown in Fig. 4.2. The mesh in OpenFOAM can consist of arbitrary convex polyhedrons and is unstructured, i.e. the cells are not ordered such that an easy scheme to find the neighbors of a cell does not exists. The most basic representation in OpenFOAM is as follows: First, there is a list of *points* (*vectors*) defining the position of every corner of every mesh *cell*. Then, there is a list of *cell faces* (compare Fig. 4.1). Every *face* is in turn represented by an ordered list of all its corner points which are referenced by the index in the list of points. After that, a similar list can be created to define the cells as a list of cell faces. Now, the computational domain is filled by cells, but for the handling of the mesh additional lists are necessary. One list indicates for every cell face by which cell it is owned. This can be only one cell for every face and the ordering of the points is such that the face normal calculated from them will point outwards of the owner cell. For all *internal faces* there is another list which gives the list index of the neighbor cell. All *boundary faces* can be grouped to *patches* and will appear in the faces list in an ordered way after all the internal faces. For the boundary patches certain types can be defined determining their behaviour in the computation like `wall` or `symmetryPlane`. All this information will be stored in a folder `polyMesh` (compare figure B.1) in which the file `boundary` is the only one that the user has to modify from time to time in order to adapt the boundary conditions. Note that almost arbitrary boundary planes can be combined to a *patch*. In figure 4.2 the front side could form a patch, but also e.g. the front and back together or even all sides together. Except for some physical *patch types* like `cyclic` or `wedge`, there are no restrictions of the grouping.

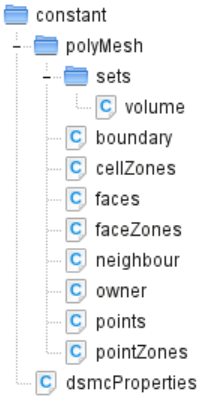

Figure 4.1 – *File structure for the mesh in OpenFOAM, the structure for the full DSMC case is shown in Fig. B.1*

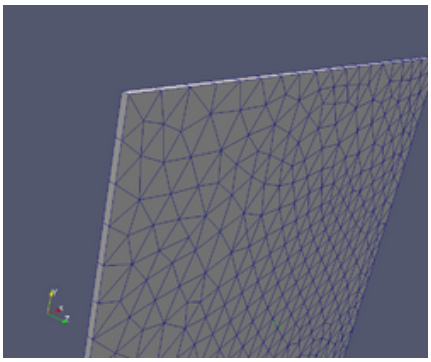

Figure 4.2 – *Example of a simple Delauny mesh*

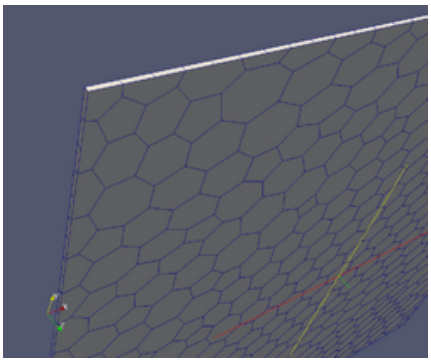

Figure 4.3 – *Example of a Voronoi mesh created with* polyDualMesh *from the mesh in Fig. 4.2.*

The creation of such a complicated set of lists cannot be done by hand. Therefore OpenFOAM provides a utility called *blockMesh* to generate simple meshes. As users might want to handle more complicated geometries or use geometries created with CAD tools, OpenFOAM also offers tools to convert various other mesh formats into the native format. Additionally, the program offers many tools to handle and manipulate meshes and calculate basic properties like cell centers, cell volumes and face normal vectors whose magnitude is giving the face area. There is even an application (*polyDualMesh*) to calculate the dual of a mesh creating a Voronoi mesh (compare Fig. 4.3) from a Delauny mesh. Voronoi meshes are found to have very favorable properties for DSMC calculations [82]. The mesh in OpenFOAM can also be changed during runtime allowing for moving parts in the simulation.

There is further the possibility to select parts of a mesh with the help of so called *topoSetSources*. This is a base class for classes like *cylinderToCell*, *boxToCell* etc. which allow to select cells or faces that have their center within a simple geometric entity that can be described by a few parameters in a *dictionary* (Sec. 4.1.3).

OpenFOAM also offers tools to decompose the mesh, i.e. to split it into smaller parts that are each assigned to a processor for separate calculations. Simple algorithms for decomposition are directly implemented in OpenFOAM but there is also more complex third party software delivered together with OpenFOAM like Scotch and Metis. These algorithms are interfaced with OpenFOAM and can be called via the configuration file `decomposeParDict`.

4.1.2 Parallelization

OpenFOAM is fully parallelized. The method used for all solvers is domain decomposition, meaning that the computational domain is split between different processors. The inter-processor communication is handled by the *Message Passing Interface (MPI)*. This allows one to spread calculations over an arbitrary number of processors within one machine or even over several computers. The parallel performance of two solvers was tested in [83] and it was found that the performance is reasonable up to a thousand tasks on an Intel NUMA architecture. Above that a bottleneck in core linear algebra libraries exists, which however may not affect the DSMC solver that was not tested. As usual, OpenFOAM provides all the necessary tools to decompose, handle and recombine calculations.

4.1.3 Data Types

OpenFOAM defines all basic data types by its own. For example, the C++ type *int* is called *label* while *double* (or *float*, depending on the settings) is called *scalar*. This is

done to provide the data types with a set of suitable operators, define some useful constants and provide some additional functionality like a function *inv()* for calculating the inverse. Most important for all data types is a common framework for input/output (I/O) operations. Starting from these simple data types, more and more complex types are defined. Scalars are extended to vectors and tensors which in turn are combined with a dimension set to allow dimension checking in calculations.

A very important data type, the *GeometricField*, is created through a rather complicated inheritance structure. It consists of several lists, one for the internal field which holds a field value for every mesh cell, and a list of boundary patches each holding a list of field values for every cell face. Since the list class is a template, the *GeometricField* is one as well and hence can hold different data types like *scalars*, *vectors* or *tensors*. Several template specializations are explicitly defined in the program and are called *volScalarField*, *volVectorField* and *volTensorField*. The *GeometricField* class also defines operators for handling the field, such that mathematical operations can be used for fields and functions for reading and writing fields from and to files are present as well.

Another important data type is the *dictionary*. Like in a real dictionary, it has two columns. One for the name of the entry and one for the data. But opposed to real dictionaries, the data does not have to be a word as well but can be of rather arbitrary type. Together with its I/O-operations this makes the dictionary a perfect class to be used in configuration files. For example, a dictionary entry could be

```
velocity (10 0 0);
```

Configuration files built from such *dictionary entries* are much easier to read and write by the user and no special order of the entries has to be maintained as the underlying hash table offers efficient lookup functions. Further, being of arbitrary data type, the data field of an entry can be of type *dictionary* as well. This allows to build hierarchies of *dictionaries* and *subdictionaries* which can be used to structure data for better readability and to use various *keywords* in several hierarchy levels.

4.1.4 Object Registries

OpenFOAM uses a hierarchical database were various objects like fields register themselves. This is used as a standardized interface to manage the communication between various solvers and their data. At the top of this database called the *objectRegistry* is always an instance of the *Time* class, usually called *runTime*. The *mesh* class registers itself at a sublevel of *runTime* and the fields are registered as sublevels of the *mesh*. The *objectRegistry* is complemented by the classes *IOobject* and *regIOobject*. *IOobject* is a class that provides standardized input/output support, as well as giving access to *runTime*.

regIOobject automatically manages the registration and deregistration of objects to the *objectRegistry*. The *objectRegistry* class is derived from *regIOobject* making it a subgroup of *regIOobject* and allowing for the hierarchical structure which is very useful for writing. Usually, the solver data has to be written to file at some time, which is triggered by *runTime*. This class tells all its registered objects to write themselves to files. If the registered object is a database itself, it cycles through all its registered objects as well to tell them to write themselves. Reading from file is usually performed only when needed. Then the object registry tells the solver where to look for the file and if it was modified, i.e. if it really has to be read again.

4.1.5 Virtual Constructors

C++ allows to call member functions of an object through its base class. These functions are called virtual functions. However, for constructors this mechanism isn't allowed although it could be of great use. For example, it would allow to create a derived class by creating the parent class with a certain parameter. Such a behaviour would in turn allow to build a common interface to all child classes were new models can easily be added at a later time. This is exactly what OpenFOAM wants to do and so it uses a workaround which is called *runTimeSelection* mechanism in the program and which is generally known as virtual constructors. Much of the implementation is hidden in a set of macros that define a hash table with pointers to the various constructors of the child classes before the execution of the program. Then, the correct constructor can be selected via its *typeName* at runtime. An example for the usage of this concept in the DSMC solver is the selection of a *BinaryCollisionModel* through a configuration file. In this case, the *runTimeSelection* mechanism also allows the user to easily add new *BinaryCollisionModels* without taking care of the interface to the calling class. Only model specific functions like *collide()* have to be implemented.

4.1.6 Pre- and Postprocessing

Before a run can be started, a lot of preparation has to be done and after the run has finished, the results should be processed in some way to extract the relevant data. OpenFOAM offers various tools for pre- and postprocessing. For example, there are tools for mesh import or creation, mesh manipulation, field initialization, case decomposition and initialization as pre-processing tools. The main tool for postprocessing is *ParaView* [84], a third party visualization and data analysis software to which OpenFOAM delivers an interface. There are also tools to convert the results to other file formats used by some common third party programs. Additionally there is a possibility to load functions from certain libraries through the configuration file `controlDict` to be executed at certain time steps during the calculation. One example is the function *fieldAverage* that calculates time averages of the specified fields. Note that not all tools are written in C++. Some, like *foamLog* that extracts relevant data from a logfile, are shell scripts that are delivered together with OpenFOAM. The required file structure of an executable case together with some description can be found in appendix B.

4.2 Description of the Solver *dsmcFoam*

The solver for DSMC calculations is called *dsmcFoam* in OpenFOAM. Besides the classes from the common infrastructure, there are only a few new special ones for the solver that are mainly derived from some generic parent class. The two most important ones are the template classes *DsmcParcel<ParcelType>* and *DsmcCloud<ParcelType>*. Additionally, there are the templated base classes *BinaryCollisionModel<CloudType>*, *InflowBoundaryModel<CloudType>* and *WallInteractionModel<CloudType>* together with their specific models like e.g. *VariableHardSphere* for the binary collisions. These classes will be described in the following section while some helper classes will not be explained in detail. As a convention, private or protected member variables of classes are marked by an underscore at the end of the variable name in OpenFOAM. This underscore will be omitted in the following such that *variableName_* will be the same as *variableName*. Further, template arguments will be omitted from time to time as well as function arguments to increase readability. Finally, all setter and getter functions as well as constructors and destructors are never shown together with most operators.

4.2.1 Particles

Every simulated particle in the program is represented as an instance of a *dsmcParcel* which is a child class and a type definition of *DsmcParcel<ParcelType>* specializing the template to *DsmcParcel<dsmcParcel>*, compare Fig. 4.4. This complicated design with the template parameter being the class itself is used for maximal efficiency in some function calls avoiding virtual functions by using static casts. All the functionality of the *dsmcParcel* is already implemented in the parent class *DsmcParcel* which in turn inherits most of its functionality from its parent class *Particle* from the OpenFOAM core. The *Particle* class handles the sorting of particles into cells, does the tracking along straight lines to cell *faces*, and handles interactions with different *patch* types. For this, it needs the attributes shown in the UML diagram below, indicating for example at which position and in which cell the particle is, at which *face* it is, and to which processor it is assigned. The *DsmcParcel* class overloads some of the patch interaction functions and has a function *move()*, which uses the tracking functions from its parent class to move the particle for a certain time interval. As additional attributes it holds the velocity, internal energy, and a *label* indicating the particle type which is equal to the position in the `typeIdList` in the configuration file `dsmcProperties`. The other classes shown on the right side of Fig. 4.4 are helper classes. *iNew* is used to read-construct particles when they are transferred from one processor to another. *trackData* is a class that provides data to some tracking functions and *constantProperties* holds the constant properties of a particle like its mass and its diameter d. This class is only structurally a member of *DsmcParcel* but is not instantiated for every *dsmcParcel*. In fact, it is only instantiated once for every particle type by the *DsmcCloud*.

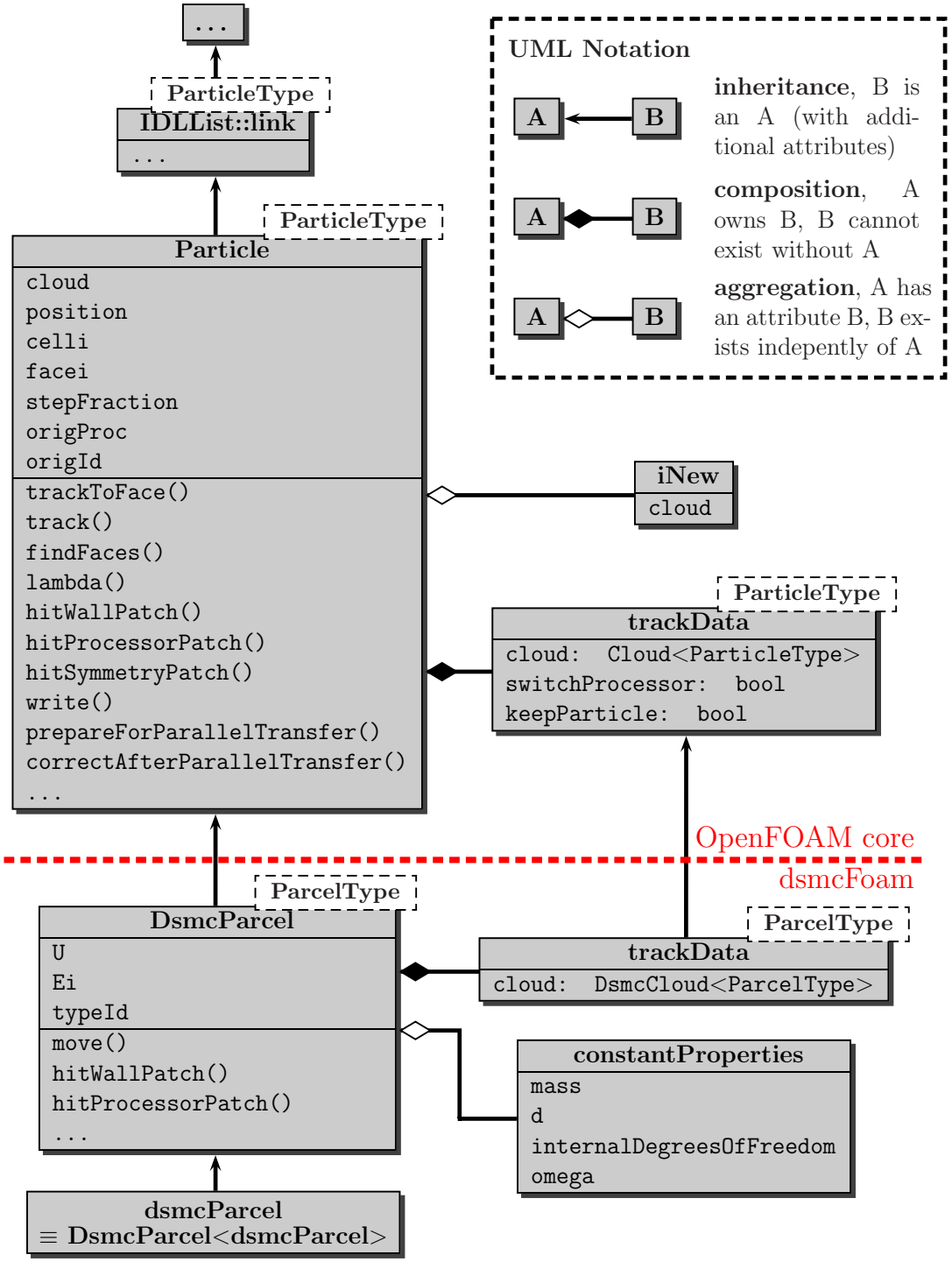

Figure 4.4 – *Simplified UML diagram for particles in OpenFOAM; All class attributes are shown while several member functions are omitted. The arrows indicate inheritance and the classes upstream of* Particle<ParticleType> *are omitted. Every particle has as an attribute the class* trackData *for which the data type of the attributes is shown for clarification. All classes above the dashed red line belong to the OpenFOAM core and are used by other solvers as well while the classes below the red line are specific to the* dsmcFoam *solver.*

4.2.2 DsmcCloud

The second major class of the solver is the class *DsmcCloud*. From the most basic point of view it is a list of all particles, which is indicated in the UML diagram (Fig. 4.5) by the inheritance from *IDLList* and the inheritance of *DsmcParcel* from *IDLList::link*. But the cloud is much more. It is the first solver specific class that gets instantiated by the program and holds all the data of the solver, either directly by its own or indirectly via member classes. It also performs the whole computation within its *evolve()* function. Before, the cloud may have to be initialized, i.e. the mesh cells have to be filled with simulated particles with properties such that the initial conditions specified by the user are obeyed. This task is done by the *initialise()* function. Then, particles have to be added at inflow patches, which can be done according to different models. Like *BinaryCollisionsModels* and *WallInteractionModels*, the *InflowBoundaryModels* are implemented with the *runTimeSelection mechanism*, where the *DsmcCloud* holds only a (auto-)pointer to the abstract base class of the submodel like in Fig. 4.6. The correct model to be used during the calculation is chosen by the user via the configuration file `dsmcProperties`. After the *evolve()* function has called the suitable *inflow()* function, it calls the *move()* function from the parent class *Cloud*. This function calls the *move()* function of every particle and handles the assignment of the particles to the correct processor. After that, the function *collisions()* is called to select collision partners within (sub-)cells of the mesh and perform collisions according to the chosen *BinaryCollisionModel*. The last step in the loop is to calculate the new field properties from the particle properties, which is done by the function *calculateFieldProperties()*. The loop around *evolve()* is steered during run time by the entries of the file `controlDict`.

There are several member functions of *DsmcCloud* that calculate some macroscopic properties of the system, like *massInjected()*, and functions that calculate average properties of particles from the Maxwell distribution. The functions *equipartitionLinearVelocity()* and *equipartitionInternalEnergy()* create a random sample value from the suitable distribution of the velocity and internal energy respectively, using the random number generator hold by the class attribute *rndGen*. A list of *constantProperties* for each particle type is stored in *constProps* and the *DsmcCloud* holds all fields as member variables. Finally, the function *autoMap()* of the parent class *Cloud* is used to sort the particles into the correct mesh cell after the mesh has changed. This parent class also implements the main input/output functions. The second parent class of *DsmcCloud*, namely *DsmcBaseCloud*, only disallows standard bitwise copy construction and bitwise assignment by declaring these operations as private.

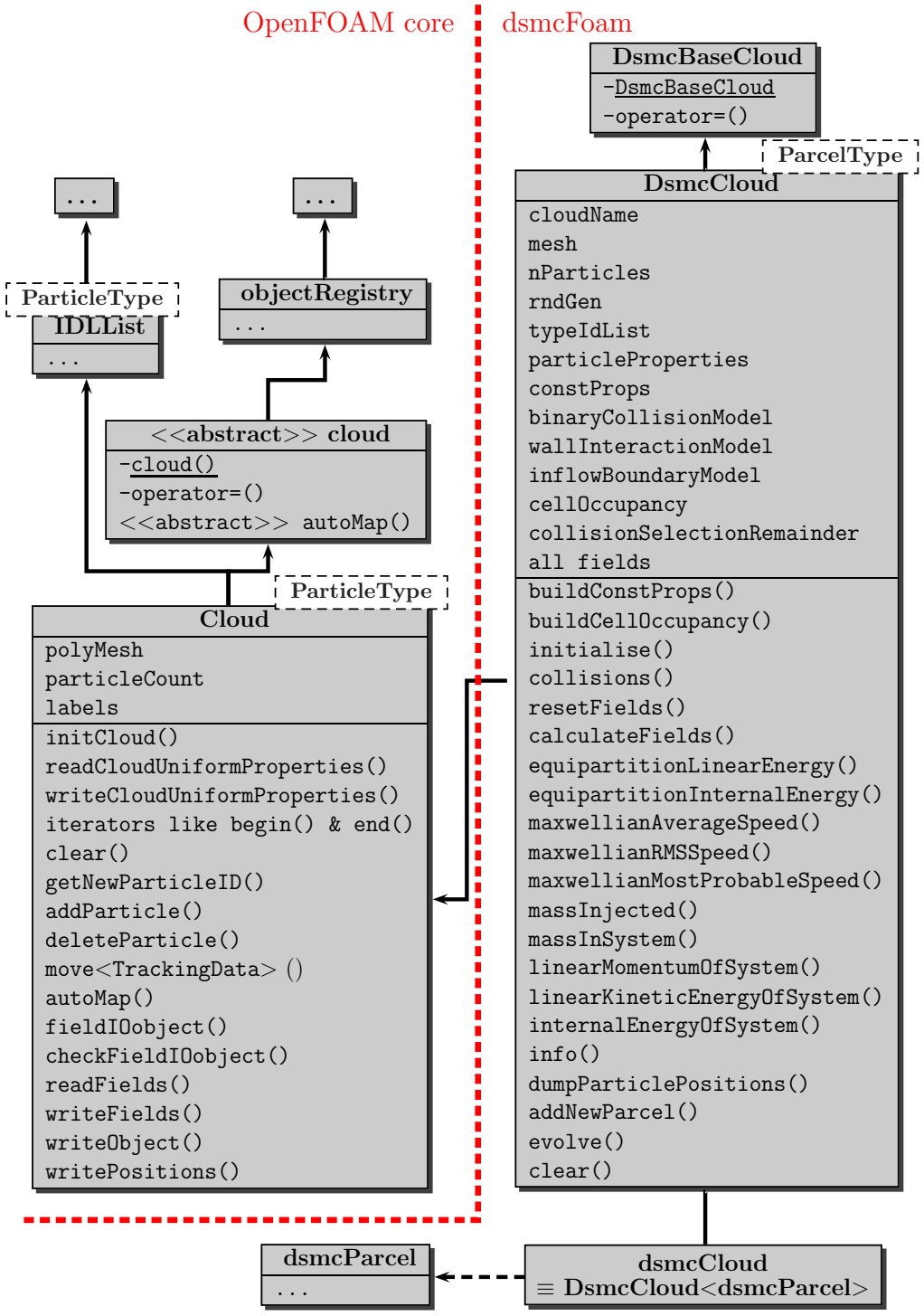

Figure 4.5 – *Simplified UML diagram for* clouds *in OpenFOAM; All class attributes are shown while several member functions are omitted, e.g. all setter and getter functions. The arrows indicate inheritance and the classes upstream of* IDLList<ParticleType> *and* objectRegistry *are omitted. The classes* cloud *and* DsmcBaseCloud *are only used to disallow bitwise copy operations and copy construction. All classes left of the dashed red line belong to the OpenFOAM core and are used by other solvers as well while the classes right to the red line are specific to the* dsmcFoam *solver.*

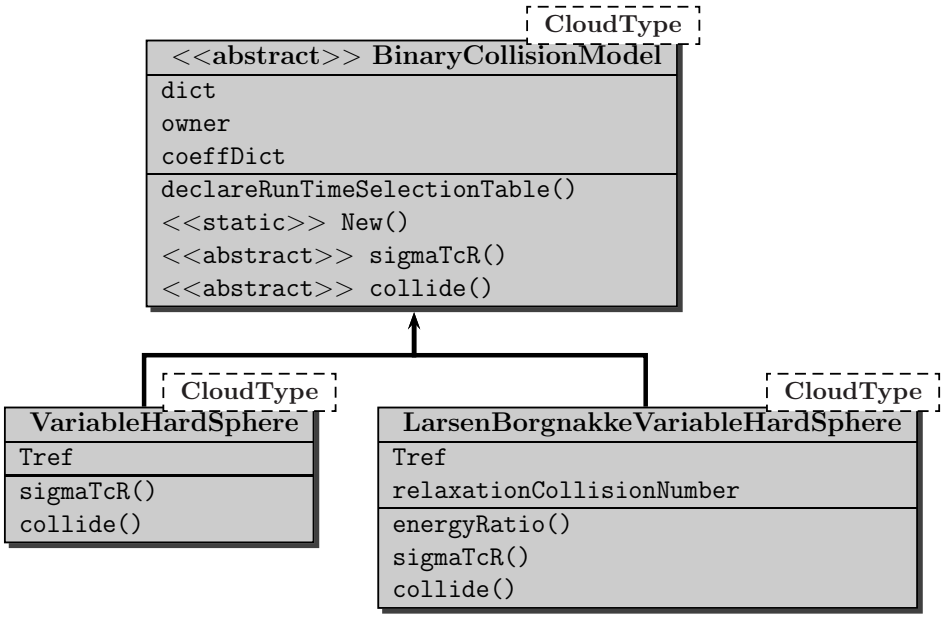

Figure 4.6 – *Simplified UML diagram for* BinaryCollisionModels *in OpenFOAM; All class attributes are shown while all setter and getter functions are omitted. The arrows indicate inheritance.*

5 Changes to the Standard DSMC Solver

In the standard OpenFoam DSMC solver *dsmcFoam* it is not possible to simulate pressure driven flows. One has only the option between the two inflow models `FreeStream` and `NoInflow`. For pressure driven flows and as basis for all further work the extended solver *dsmcModFoam* is used, which is available online ([85]). In this code, the scalar fields *rhoN* and *rhoM* are changed to *volVectorFields*, to allow three particle species to be placed anywhere in arbitrary quantities in the simulated region at the beginning. Each element of the vector represents the particle density of a certain particle species. The new scalar fields *spaceAverageRhoN* and *spaceAverageRhoM* take the role of the old fields *rhoN* and *rhoM* respectively as the average (particle) density in a cell. For the setup of a simulation it is required to write the files `0/rhoN` and `0/rhoM`. The files `0/spaceAverageRhoN` and `0/spaceAverageRhoM` can be created with the new utility *SpaceAverageRhoN*. It has to be taken into account that the initialization procedure uses the boundary values for the particle density given in the file `0/rhoN` but for the interior of the domain the values from the dictionary `system/dsmcInitialiseDict` are used. So these values should be set to compatible values.

In this work, the new solver *dsmcSpinModFoam* was created which is an extended version of the *dsmcModFoam* solver. The differences between *dsmcModFoam* and the original solver *dsmcFoam* are relatively small and all the UML diagrams in Sec. 4.2 are valid for *dsmcModFoam* as well. The current work considerably extends the functionality of the solver. Several new classes are added and many existing functions are extended or overloaded.

5.1 Magnetic Forces

An important part of this thesis was the integration of magnetic fields into the simulation. The force $\vec{F}$ on a neutral particle with magnetic moment $\vec{\mu}$ can be calculated according to Eq. (2.11), where the gradient of the field has to be calculated according to Eq. (2.16). Hence, one needs hyperfine states in the simulation, magnetic fields, and a description of the region where they are active, i.e. nonzero. If more than one force is acting on a particle, one needs a calculation of the combined force and the effect on the particle and last but not least, some kind of output and a calculation of the polarization in homogeneous magnetic fields to see and check the effects of magnetic fields. In this section the various parts of the implementation of magnetic fields and their effects will be described together with some problems and their solutions that occurred during the work.

5.1.1 Hyperfine States

Every particle needs variables for describing its hyperfine state. There are several possibilities of describing a certain state, e.g. with one variable for the number of the state or with two variables for electron and nuclear spin. But since an analytic description of the hyperfine splitting energy is only possible for $S = 1/2$ and therefore the whole simulation is restricted to this case, a slightly different approach was chosen. A first variable *whichHalf* with values 1 or 2 describes the upper and lower set of states in the Breit-Rabi-Diagram or, correspondingly, the electron spin state. The second variable *whichState* with values $1 \dots 2I + 1$ basically matches the index j in Chap. 2.1. The only difference is that j has a range from 2 to $2I + 1$ and the additional state *whichState* $= 1$ describes the pure state such that one has the following relationship between the used labeling and the standard labeling of hyperfine states:

$$|\text{whichHalf},\ \text{whichState}\rangle = \begin{cases} |1, \text{whichState}\rangle & = |\text{whichState}\rangle & = |1\rangle,\ |2\rangle \\ |2, \text{whichState} = 1\rangle & = |2I + 2\rangle & = |3\rangle \\ |2, \text{whichState} > 1\rangle & = |4(I + 1) - \text{whichState}\rangle & = |4\rangle \end{cases} \tag{5.1}$$

where the last column is specifically for hydrogen (nuclear spin-1/2 particles). This choice allows for a particular easy force calculation and limits the use of case-by-case analysis but requires twice as much memory space. However, the additional memory usage of a few MB at maximum might well be outweighed by a possible speedup of the calculation.

5.1.2 "Field" of Hyperfine States

For calculating a polarization value values of the relative population numbers of each hyperfine state are necessary. On the finite volume mesh, this can be defined for every cell as the number of particles of a certain species in a certain hyperfine state divided by the total particle number of this species within this particular cell. In the source code a *volTensorField* structure was chosen to hold these values. There is one field for every particle type with names *hfsPopulationNumbers0* to *hfsPopulationNumbers2* where the last number is the *typeId* of the particle type, i.e. the position of the particle species in the `typeIdList` in the file `dsmcPropertiesDict`. The number for hyperfine state i is saved in the tensor element i when the tensor is represented as a one-dimensional list. This limits the maximum number of hyperfine states per particle to 9, i.e. $I \leq \frac{3}{2}$ which is sufficient for our purpose. The clear advantage of this choice is that the standard I/O-functions for fields and the postprocessing including field averaging and interfacing with ParaView can be used.

The time averaging of the three fields could be done in the standard way, i.e. calculating the average of a number of fields calculated for different times. However, this method has the problem that there is no guarantee that there are particles within each

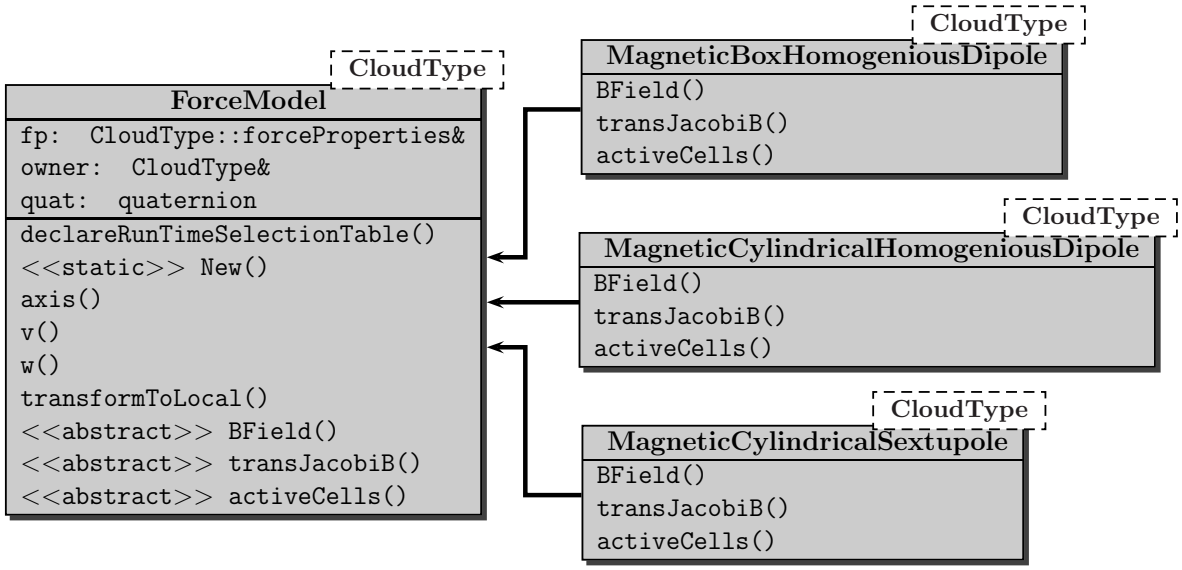

Figure 5.1 – *UML diagram for* ForceModels *in* dsmcSpinModFoam*; All class attributes are shown while setter and getter functions are omitted. The arrows indicate inheritance.*

cell at every time step, i.e. the field value can be undefined from time to time. Simply setting the undefined values to zero may create serious artifacts if they are too numerous. That's why another method was chosen. During the averaging time, the place in the tensor for a certain hyperfine state will just hold the sum of particles in this particular hyperfine state during the time interval. Averaging then means just dividing by the sum over all components of the tensor. If there were always particles in every cell, these two methods would be equivalent, but so the second one has a clear advantage in avoiding artifacts by increasing the likelihood that there is a particle in the cell by a factor equivalent to the number of time steps over which the average stretches. This alternative averaging method for these three fields requires some small changes to the `controlDict` as can be seen in appendix D which are explained in appendix B.

5.1.3 Magnetic Fields

For the calculation of the force on a particle a description of the magnetic field at the position of the particle is a prerequisite. Possible are an analytical or a numerical description. The latter one has the advantage that very complex fields can be specified but the major disadvantages are that it can only be specified at certain grid points and has to be interpolated in between and that it has to be recalculated for every change of the geometry. Therefore, for standard magnets the analytical description is more favorable. It is implemented with the runtime selection mechanism in a similar way as other *submodels*, for example the binary collision models, see Fig. 5.1.
The base class *ForceModel* declares three abstract functions which have to be defined by every child class. Every such child class represents a certain type of magnet that has to

return the B field at a specified location, the transpose of the Jacobi matrix and a set of mesh cells where the field is active, i.e. nonzero. All values should be defined in a local coordinate system of the magnet such that the magnet can be placed anywhere in the computational domain and in arbitrary orientations. This requires global coordinates transformed to local ones in the calculation of field properties at a particles position which is a combination of a translation and a rotation. The whole transformation is performed by the function *transformToLocal()* where the rotational part is described by a quaternion, a data type inherent to OpenFOAM. The quaternion can be created with a vector for the rotation axis v and a scalar $w = \cos\left(\frac{\theta}{2}\right)$ where θ is the rotation angle. These values can be obtained when the magnet axis is known and it is assumed that this axis defines the x-axis of the local coordinate system. The axis $\vec{a}$ is defined by two points, *origin* and *P2*. When $\vec{a} = (a_1, a_2, a_3)$ is normalized, it is

$$v = \hat{e}_1 \times \hat{a} = (0, -a_3, a_2) \tag{5.2}$$

$$w = \cos\left(\frac{\theta}{2}\right) = \sqrt{\frac{1}{2}(1 + \cos(\theta))} = \sqrt{\frac{1}{2}(1 + \hat{e}_1 \cdot \hat{a})} = \sqrt{\frac{1}{2}(1 + a_1)}. \tag{5.3}$$

These values, together with the transformation, are the same for all magnets and hence are located in the base class *ForceModel*.

The types, dimensions and orientations of magnets can be defined as a dictionary in the file `dsmcProperties`. Additionally, as for the particle properties a list of the names of all magnets has to be provided, called `forceIdList`. The properties for every magnet will then be stored in a class *forceProperties* (compare Fig. 5.2) that can hold several parameters of which the following are compulsory

type Magnet type, e.g. MagneticCylindricalSextupole

origin Local origin given in global coordinates

P2 Second point given in global coordinates, used to define magnet axis and size

The other parameters in the *forceProperties* class were originally thought for tempered sextupole magnets were *r1Origin* is the inner radius of the magnet at the origin, *r1End* is the inner radius at *P2* and *r2* is the outer radius. *B0* is the pole tip field and *vectorB0* was later introduced to describe the B field of homogeneous magnets. The angle *phi*, measured in degrees according to the right hand rule, defines a rotation of the magnet around its axis. Although the parameter names may not be suggestive for other magnet geometries, they are currently used for them as well. Which parameters are needed and their meaning can be found in the comments of the source code.

At initialization of the extended *dsmcCloud* class, a list of *forceProperties* of all magnets will be created as well as a list of pointers to all instances of magnets. When all magnets are initialized, a lookup table called *forceActiveList* will be created by the function *makeForceActiveList()* which shows for every mesh cell which magnet has a nonzero

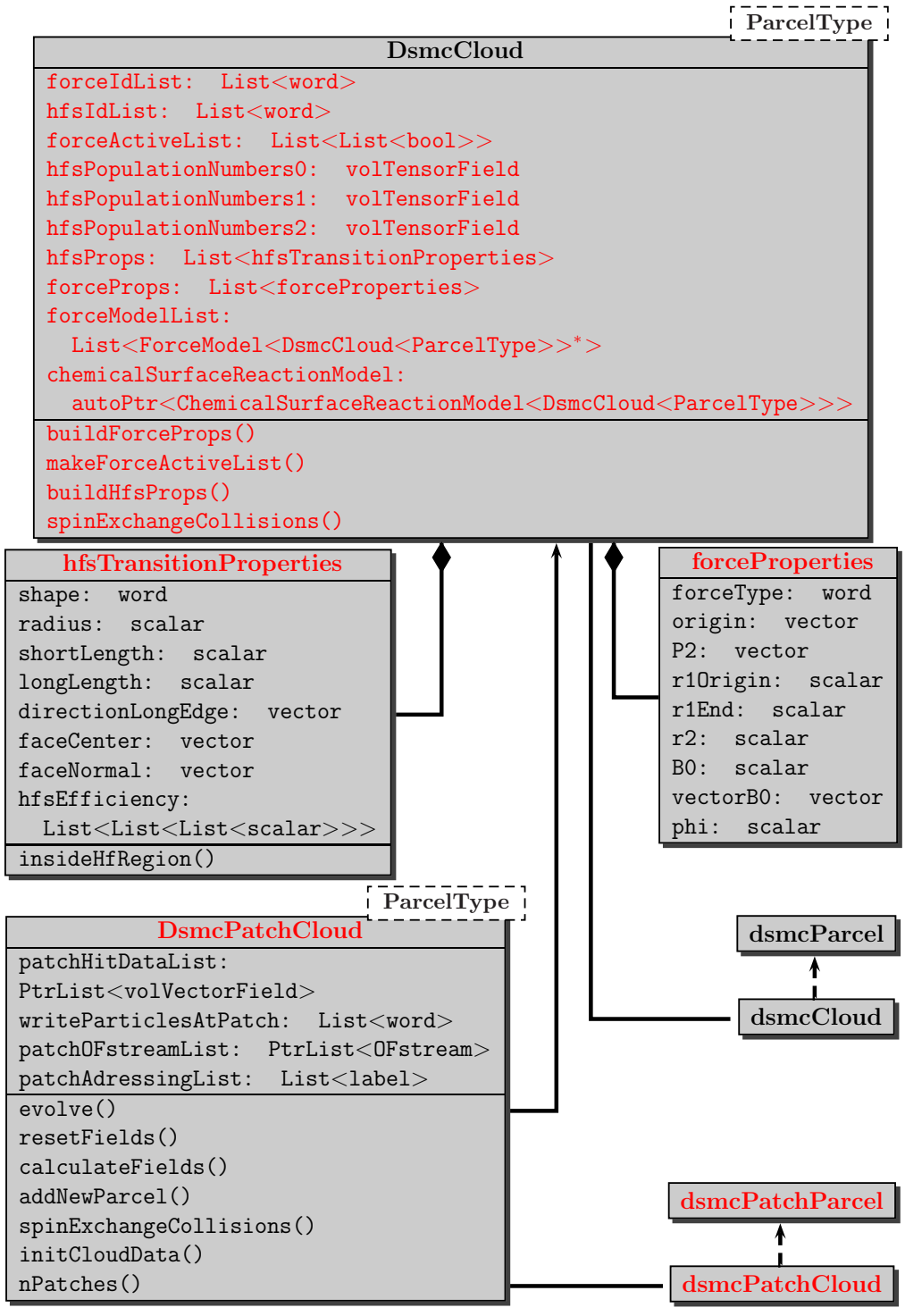

Figure 5.2 – *UML diagram for the* dsmcCloud *in* dsmcSpinModFoam*; Class attributes are shown with their type and setter and getter functions are omitted. The diagram is meant as an extension to the diagram of the* dsmcCloud *class from the standard* dsmcFoam *solver in Fig. 4.5. Attributes and functions printed in red are new in* dsmcSpinModFoam *as well as complete classes of which only the class name is printed red. The class* dsmcPatchCloud *will be used for particles counting the wall collisions, compare Sec. 5.4.*

field value in this cell. The table has one column more than there are magnets. In this last column it is indicated if any force is active in this particular cell, which helps avoiding loops over all magnets in cases where there is no field in a cell. For setting the list, the function *makeForceActiveList()* cycles through all magnet instances and calls their function *activeCells()*. In turn, these functions make use of classes derived from *topoSetSource* like e.g. *cylinderToCell* provided by OpenFOAM to obtain all cells whose cell center is within a simple geometric volume with only a few parameters saved in the corresponding *forceProperties*.

Calculating the B field for a single magnet is straightforward. One has only to remember to transform the vector back to global coordinates before it is returned. For the Jacobi matrix the situation is a bit more involved because the simplified analytical descriptions specifies the field only inside the magnet and neglects fringe fields. This causes a step in the B field which entails a gradient in the form of the Delta function. A small argument shows, that although the time spent crossing this surface is infinitely small, the effect of this step cannot be neglected in general. Consider the force on a particle caused by the field gradient (compare with Eq. (2.11))

$$\vec{F} = -\mu \nabla |\vec{B}| = -\nabla E = m\dot{\vec{v}}. \tag{5.4}$$

The change in the B field leads to a change in potential energy equal to the energy shift of the hyperfine states (Eq. (2.5)) that in turn leads to a change in velocity. Because $\nabla|\vec{B}|$ is normal to the boundary surface, only the velocity component $v_\perp$ normal to the surface will be changed. The velocity change $\Delta v_\perp$ can be computed from energy conservation to be

$$\Delta v_\perp = \sqrt{v_{1\perp}^2 - \frac{2\Delta E}{m}} - v_{1\perp} \tag{5.5}$$

where $v_{1\perp}$ is the velocity normal to the field step outside of the magnetic field. In most cases the effect is very small which shows that the fringe fields shouldn't affect the hyperfine state separation efficiencies of the magnets. Nonetheless, this effect is realized in the program and is implemented in the function *DsmcParcel::hitFace()* where it is checked whether a field is active on one side of a cell face but not on the other. If appropriate, the velocity correction is applied with the normal direction of the field taken as the normal vector of the cell face. In cases where this effect is big enough to have any effect, one should therefore build the geometry and select the cells with magnetic field appropriately.

5.1.4 Particle Motion in Magnetic Fields

In the original code of the *dsmcFoam* solver, the translation of the particles is performed as follows: From the current position and velocity and the time step `deltaT` ($= \Delta t_{\mathrm{cD}}$) provided in the file `controlDict` the next position is calculated by $\vec{x}_{n+1} = \vec{x}_n + \vec{v}_n \, \Delta t_{\mathrm{cD}}$

where $\vec{x}_n$ and $\vec{v}_n$ may be restricted to a certain plane depending on the dimensionality of the simulation. The connection line between the current position $\vec{x}_n$ and the position at the end of the time step $\vec{x}_{n+1}$ is searched for intersections with the cell faces of the mesh (see Sec. 5.1.5). If there is no intersection, the particle is moved to $\vec{x}_{n+1}$ otherwise it is moved to the first cell face along the track. On internal faces the particle is simply moved to the next cell, i.e. its variable *celli* is changed. On processor boundaries the particle is appended to a list which shows which particle has to be transferred to which processor. If the particle hits a wall patch, the velocity and possibly other particle properties like the internal energy are changed according to a *wallInteractionModel*. For other patches like inlets or symmetry patches, there are specific actions as well. At all face hits the particle attribute *stepFraction* holding the fraction of the current time step completed, is set. After transferring the marked particles to the new processors, the procedure is repeated with a reduced time step $dt = \Delta t_{\mathrm{cD}}(1 - \texttt{stepFraction})$ until all particles have stepFraction $= 1$. Afterwards the collisions are performed and the whole procedure is repeated.

If magnetic fields are allowed, the particles will generally be accelerated and move along curved tracks. Hence, calculating $\vec{x}_{n+1}$ is not as easy as for the case without forces. The equation of motion describes the particle behaviour, but for arbitrary fields there is no analytical solution. However, a numerical solution can always be found. The question is, which numerical method is suitable by means of accuracy, stability and computational effort.

Only single step methods are useful since there is no information of the previous time steps at the time of calculation of $\vec{x}_{n+1}$. Another restriction is to use only explicit methods since solving a nonlinear system of equations is computationally costly. So it is natural to consider the simplest explicit scheme, namely the Euler method, first. The equation of motion can be written in the form

$$\begin{aligned} \dot{\vec{x}} &= \vec{v} \\ \dot{\vec{v}} &= \frac{\vec{F}}{m}. \end{aligned} \tag{5.6}$$

The standard Euler method is to approximate the solution by

$$\begin{aligned} \vec{x}(t + \Delta t) &= \vec{x}(t) + \vec{v}(t)\Delta t \\ \vec{v}(t + \Delta t) &= \vec{v}(t) + \frac{F(\vec{x}(t))}{m}\Delta t. \end{aligned} \tag{5.7}$$

This method converges in order $O(\Delta t)$ but it can be numerically unstable for some (stiff) problems and it is not symplectic meaning it does not preserve energy. Only a small variation of the method is needed to overcome the problems with instability and symplecticity. The new method is called Semi-implicit Euler method and the only

change versus the standard Euler method is, that it uses already the newly calculated position in the calculation of the new velocity:

$$\begin{aligned} \vec{x}(t+\Delta t) &= \vec{x}(t) + \vec{v}(t)\Delta t \\ \vec{v}(t+\Delta t) &= \vec{v}(t) + \frac{F(\vec{x}(t+\Delta t))}{m}\Delta t. \end{aligned} \tag{5.8}$$

This method still has convergence order one. The question arises if this is sufficient for a general DSMC simulation or whether a higher order scheme like the fourth order Runge-Kutta has to be used. Since the time step $\Delta t =: \Delta t_{\mathrm{ODE}}$ used for the solution will be the same as Δt_{cD} which should be much smaller than the mean collision time Δt_{Col} for the particles, there is no trivial answer. But one can estimate the two values of Δt_{ODE} and Δt_{Col}: For the VHS and VSS collision model the mean collision rate can be written as ([59])

$$\nu_0 = n\,\sigma_{\mathrm{T,ref}}\,v_{\mathrm{r,ref}}^{2\omega-1}\frac{2}{\sqrt{\pi}}\Gamma\left(\frac{5}{2}-\omega\right)\left(\frac{2k_{\mathrm{B}}T}{m_{\mathrm{r}}}\right)^{1-\omega} \tag{5.9}$$

$$= \frac{5(\alpha+1)(\alpha+2)}{\alpha(5-2\omega)(7-2\omega)}\frac{p}{\mu} \tag{5.10}$$

$$= \frac{1}{\Delta t_{\mathrm{Col}}} \tag{5.11}$$

where $\sigma_{\mathrm{T,ref}}$ and $v_{\mathrm{r,ref}}$ are reference values for the collision cross section and the relative velocity of the collision partners. $m_{\mathrm{r}} = \frac{m_1 m_2}{m_1+m_2}$ is the reduced mass while ω and α are parameters describing the dependence of the particle diameter on the relative velocity and the scattering angle on the impact parameter b respectively as defined in Eqs. (3.56) and (3.62). μ is the coefficient of viscosity at reference conditions. The important point is the linear dependence on the pressure p and the weak dependence on the temperature. For common ω between 0.65 and 1.1 the maximal change in collision frequency in the temperature range from $10\,\mathrm{K}$ to $10\,000\,\mathrm{K}$ is one order of magnitude, while for $\omega = 1$ the collision rate is independent of temperature. Therefore it is sufficient to estimate the collision rate by considering only Eq. (5.10) for the reference values at $T = 273\,\mathrm{K}$ and allow a factor 5 for temperature dependence. As α generally lies between 1 and 2, the first fraction in Eq. (5.10) is always bigger than about 0.7. The highest viscosity coefficient listed in [59] is for neon with $\mu = 2.975 * 10^{-5}\,\mathrm{N\,s\,m^{-2}}$ at standard conditions. For these adversely chosen parameters one can conclude that for all particles within the assumed parameter range $\omega > 0.65$ and $\alpha > 1$ and for all temperatures $\Delta t_{\mathrm{Col}} \leq \frac{10^{-5}\mathrm{Pa\cdot s}}{p}$. Thus the maximal time step Δt_{cD} that can be chosen for the DSMC method is about an order of magnitude smaller, say

$$\Delta t_{\mathrm{cD}} \leq \frac{10^{-6}\mathrm{Pa\cdot s}}{p}. \tag{5.12}$$

From this one can see, that for very low pressures the mean collision time and the time step for calculation can easily be in the order of several seconds or even minutes. With an assumed particle velocity of $1000\,\frac{\mathrm{m}}{\mathrm{s}}$ this yields a traveled distance between two calculation steps in the order of kilometers which is ways too long independent of the numerical scheme. Thus, for very low pressures the chosen time step Δt_{cD} will not be limited by the collision frequency but rather by the dimensions of the simulated area and the numerical scheme. With a mean thermal speed of a particle of $\bar{v'} = 2\sqrt{\frac{2k_{\mathrm{B}}T}{\pi m}}$ easily in the range of hundreds to thousands meters per second, the particle can move several millimeters per microsecond without even considering the flow speed. Depending on the size of the magnets $\Delta t_{\mathrm{cD}} = 10^{-6}\,\mathrm{s}$ seems to be the absolute maximum for simulations with B field. In order to support relatively large stepsizes, a numerical scheme with good convergence is needed while too much computational effort should be avoided. Therefore the fourth order Runge-Kutta method is selected. It requires the evaluation of the force at four different locations per step and is of convergence order four for an ordinary differential equation. A single time step has error $O\left((\Delta t)^5\right)$ and since the particle velocities are randomized after a random number of time steps by collisions, the total convergence order in this case is unclear. For the differential equation

$$\frac{dy(t)}{dt} = f(t, y(t)) \tag{5.13}$$

one way to express the method is

$$\begin{aligned} y_{n+1} &= y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \\ k_1 &= \Delta t\; f(t_n,\; y_n) \\ k_2 &= \Delta t\; f\left(t_n + \frac{\Delta t}{2},\; y_n + \frac{k_1}{2}\right) \\ k_3 &= \Delta t\; f\left(t_n + \frac{\Delta t}{2},\; y_n + \frac{k_2}{2}\right) \\ k_4 &= \Delta t\; f(t_n + \Delta t,\; y_n + k_3) \end{aligned} \tag{5.14}$$

with $y_n = y(t_n)$ and an initial condition $y(t_0) = y_0$. For the equation of motion it is $y = \begin{pmatrix} x \\ v \end{pmatrix}$ and $f(t, y(t)) = \begin{pmatrix} v \\ \frac{F(x)}{m} \end{pmatrix}$. This yields

$$
\begin{aligned}
\begin{pmatrix} \vec{x}_{n+1} \\ \vec{v}_{n+1} \end{pmatrix} &= \begin{pmatrix} \vec{x}_n + \vec{v}_n\,\Delta t + \frac{\Delta t}{6}\frac{\Delta t}{m}(\vec{F}_1 + \vec{F}_2 + \vec{F}_3) \\ \vec{v}_n + \frac{\Delta t}{6\,m}(\vec{F}_1 + 2\,\vec{F}_2 + 2\vec{F}_3 + \vec{F}_4) \end{pmatrix} \\
\vec{F}_1 &= \vec{F}(\vec{x}_n) \\
\vec{F}_2 &= \vec{F}\left(\vec{x}_n + \frac{\Delta t}{2}\vec{v}_n\right) \\
\vec{F}_3 &= \vec{F}\left(\vec{x}_n + \frac{\Delta t}{2}\left(\vec{v}_n + \frac{\Delta t}{2}\,\frac{\vec{F}(\vec{x}_n)}{m}\right)\right) \\
\vec{F}_4 &= \vec{F}\left(\vec{x}_n + \vec{v}_n\,\Delta t + \frac{(\Delta t)^2}{2\,m}\vec{F}\left(\vec{x}_n + \frac{\Delta t}{2}\,\vec{v}_n\right)\right).
\end{aligned}
\tag{5.15}
$$

Since the force should not explicitly depend on t, one has an autonomous ordinary differential equation that conserves energy. The explicit Runge-Kutta methods however are not symplectic meaning they do not conserve energy. But since the method has a high convergence order and no long term behaviour is considered, this is not expected to pose any problems here.

5.1.5 Finding Cell Face Crossings

The calculation of an approximate solution with a numerical scheme yields values for position and velocity at certain points. The tracks in between these points are unknown and generally approximated by straight lines. This is necessary for the standard tracking algorithm to find intersections with cell faces. If the particle shall be moved from $\vec{x}_n$ to $\vec{x}_{n+1}$ it might have to cross one or more cell faces. The closest face will be one delimiting the cell currently containing the particle. So only the faces of the current cell have to be checked for intersection. A cell face is internally represented as an ordered list of points from which one can calculate the face center $\vec{C}_\mathrm{f}$ and the face normal unit vector $\vec{S}_\mathrm{f}$ (see Fig. 5.3). Then the fraction λ of the time step at which the particle hits the face can be computed as

$$
\lambda = \frac{\alpha}{\beta} = \frac{(\vec{C}_\mathrm{f} - \vec{x}_n)\cdot\vec{S}_\mathrm{f}}{(\vec{x}_{n+1} - \vec{x}_n)\cdot\vec{S}_\mathrm{f}}.
\tag{5.16}
$$

This can be done for every face because in the used normal form of the plane they are infinitely extended. Only if the particle track is parallel to the face, there is no intersection and λ is set to a big number. At the end one has to choose the smallest λ that lies between zero and one. This is the fraction of the current time step after which the cell face is hit (soft particles can go out of the domain and therefore have $\lambda < 0$ or $\lambda > 1$ but this is not the case for `DsmcParcels`). Then the particle is moved to the face, boundary conditions might be applied and the tracking is continued with a smaller

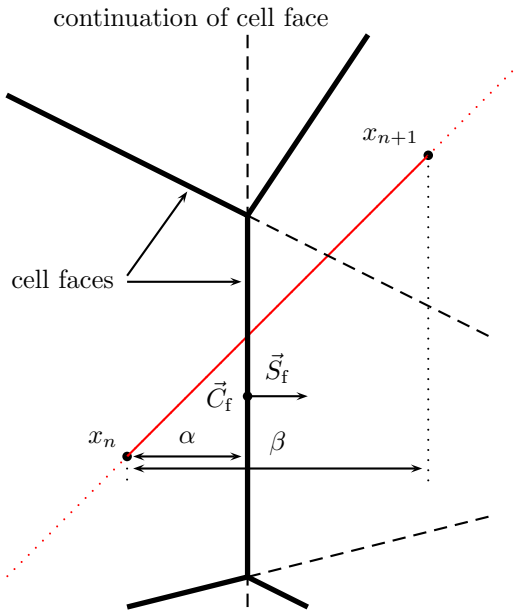

Figure 5.3 – *For tracking a particle from $\vec{x}_n$ to $\vec{x}_{n+1}$ one needs to find the intersection points with cell faces (bold lines, dashed lines indicate the infinite extension). Every face has a center $\vec{C}_\mathrm{f}$ and a unit normal vector $\vec{S}_\mathrm{f}$. With these vectors one can compute the fraction λ of the track completed until the particle reaches the face according to Eq. (5.16)*

time step until $\vec{x}_{n+1}$ is reached. If none of the λs is between 0 and 1, no face has to be crossed and the particle can be moved to the end point in one step.

For a moving mesh one needs to find the intersection of the flight path with a moving plane. The problem here is, that one only knows the points constituting the face at the beginning and end of the full time step Δt_cD. If the face is not rotating one has in the beginning $\vec{C}_{\mathrm{f},00} = \vec{C}_\mathrm{f}(t = t_0)$ and after the time step it is $\vec{C}_\mathrm{f} = \vec{C}_\mathrm{f}(t = t_0 + \Delta t_\mathrm{cD})$. If a face has been hit before in the time step, this will have happened at $\vec{C}_{\mathrm{f},0} = \vec{C}_\mathrm{f}(t = t_0 + stepFraction \cdot \Delta t_\mathrm{cD})$. So at the time of the face collision it is $\vec{C}_\mathrm{f}(\lambda) = \vec{C}_\mathrm{f} - (1 - \lambda)\left(\vec{C}_\mathrm{f} - \vec{C}_{\mathrm{f},0}\right)$ with $\lambda = (1 - stepFraction) \cdot \Delta t_\mathrm{cD}$ beeing the fraction of the current (reduced) time step. Now the face can be described as all points $\vec{r}$ for which holds $\vec{S}_\mathrm{f} \cdot (\vec{r} - \vec{C}_\mathrm{f}(\lambda)) = 0$. Using $\vec{x}(\lambda) = \vec{x}'_n + \lambda \cdot (\vec{x}_{n+1} - \vec{x}'_n)$ and inserting this for $\vec{r}$ in the last equation one obtains

$$\lambda = \frac{(\vec{C}_{\mathrm{f},0} - \vec{x}'_n) \cdot \vec{S}_\mathrm{f}}{(\vec{x}_{n+1} - \vec{x}'_n - \vec{C}_\mathrm{f} + \vec{C}_{\mathrm{f},0}) \cdot \vec{S}_\mathrm{f}} \tag{5.17}$$

where $\vec{x}'_n = \vec{x}_n + stepFraction \cdot (\vec{x}_{n+1} - \vec{x}_n)$ is the position at the beginning of the current reduced time step. If the face is rotating not only the face center is moving but also the face normal. Using the same notation for $\vec{S}_\mathrm{f}$ as for $\vec{C}_\mathrm{f}$ one has $\vec{S}_{\mathrm{f},0} = \vec{S}_{\mathrm{f},00} + stepFraction \cdot (\vec{S}_\mathrm{f} - \vec{S}_{\mathrm{f},00})$. A point on the rotating plane can be obtained by

$$\vec{r}_0 = \vec{C}_{\mathrm{f},0} + \frac{(\vec{C}_\mathrm{f} - \vec{C}_{\mathrm{f},0}) \cdot \vec{S}_\mathrm{f}}{(\hat{\omega} \times \vec{S}_{\mathrm{f},0}) \cdot \vec{S}_\mathrm{f}} \cdot (\hat{\omega} \times \vec{S}_{\mathrm{f},0}) \tag{5.18}$$

where $\hat{\omega} = \frac{\vec{S}_{\mathrm{f},0} \times \vec{S}_{\mathrm{f}}}{|\vec{S}_{\mathrm{f},0} \times \vec{S}_{\mathrm{f}}|}$ is the normalized rotation axis. In order to obtain λ one now has to solve the equation

$$(\vec{x}(\lambda) - \vec{r}_0) \cdot (\vec{S}_{\mathrm{f},0} + \lambda(\vec{S}_{\mathrm{f}} - \vec{S}_{\mathrm{f},0})) = 0 \tag{5.19}$$

which is a second order polynomial with the two solutions

$$\lambda_{1/2} = -\frac{1}{2}\left(a_{\mathrm{p}} \mp \sqrt{c_{\mathrm{p}}}\right) \tag{5.20}$$

$$a_{\mathrm{p}} = \frac{(\vec{x}'_n - \vec{r}_0) \cdot (\vec{S}_{\mathrm{f}} - \vec{S}_{\mathrm{f},0}) + (\vec{x}_{n+1} - \vec{x}'_n) \cdot \vec{S}_{\mathrm{f},0}}{(\vec{x}_{n+1} - \vec{x}'_n) \cdot (\vec{S}_{\mathrm{f}} - \vec{S}_{\mathrm{f},0})} \tag{5.21}$$

$$c_{\mathrm{p}} = a_{\mathrm{p}}^2 - 4\frac{(\vec{x}'_n - \vec{r}_0) \cdot \vec{S}_{\mathrm{f},0}}{(\vec{x}_{n+1} - \vec{x}'_n) \cdot (\vec{S}_{\mathrm{f}} - \vec{S}_{\mathrm{f},0})}. \tag{5.22}$$

With this straight line algorithm one could handle curved tracks as follows: First calculate $\vec{x}_{n+1}$ and $\vec{v}_{n+1}$ from $\vec{x}_n$ and $\vec{v}_n$. Then move the particle along the connection between $\vec{x}_n$ and $\vec{x}_{n+1}$ with the velocity $\vec{v} = \frac{\vec{x}_{n+1} - \vec{x}_n}{\Delta t_{\mathrm{cD}}}$. At the end of the time step, i.e. when *stepFraction*$= 1$, set $\vec{v} = \vec{v}_{n+1}$. This has the major disadvantage that one needs to save $\vec{v}_{n+1}$ for every particle during the time step. Another problem is that one may miss a wall or hit it although the real track would go around it as can be seen in Fig. 5.4. For wall collisions in general one has the problem to define $\vec{v}_{n+1}$ in a meaningful way: For the first calculation of the tracking speed, one uses points outside the defined computational domain. Then there is a reflection at the wall. With the straight line approximation the particle will hit the wall not only with the wrong velocity but also from the wrong direction. And if one wants to calculate better approximations by recalculating the values of $\vec{x}_{n+1}$ and $\vec{v}_{n+1}$ for the time when the wall is hit (obtained with the first approximation) one might still end up shoving the particle out of the computational domain. Simply shifting it back might mean a gain or loss in potential energy. Another point to consider is that on all internal face crossings one shouldn't recalculate the velocity according to the equation of motion in order to avoid an unphysical shift along the face which would sum up systematically for all particles with the same spin orientation leading to a separation of spin states by crossing cell boundaries (compare Fig. 5.5). Clearly an artifact one has to avoid. But using the same velocity for the whole time step would mean that one cannot account for a change in magnetic fields on cell faces. This would look like if the magnet was shifted slightly with the distance depending on the particles velocity. Another unwanted artifact. Although all described effects can be reduced to negligible size simply by reducing the time step Δt_{cD}, one may want to use a better algorithm in order to be able to use a bigger time step for faster calculation.

For this purpose, the particle tracks will be approximated by the Taylor series up to second order, i.e. by a parabola. If the force is constant, one can describe the track exactly by a parabola and as long as the force is changing only slowly, the parabola

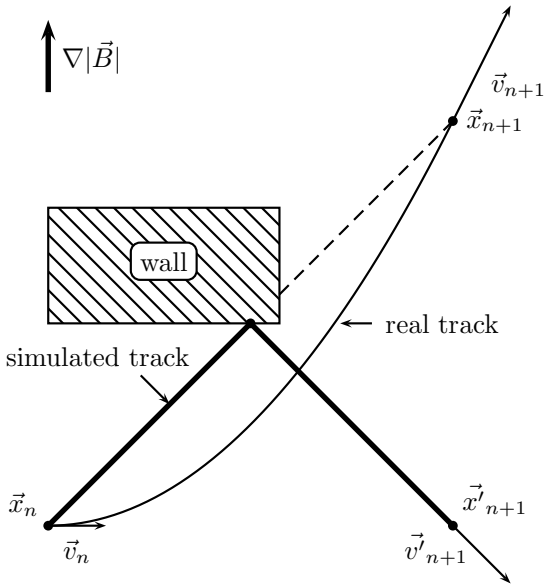

Figure 5.4 – *Illustration of a false wall collision*

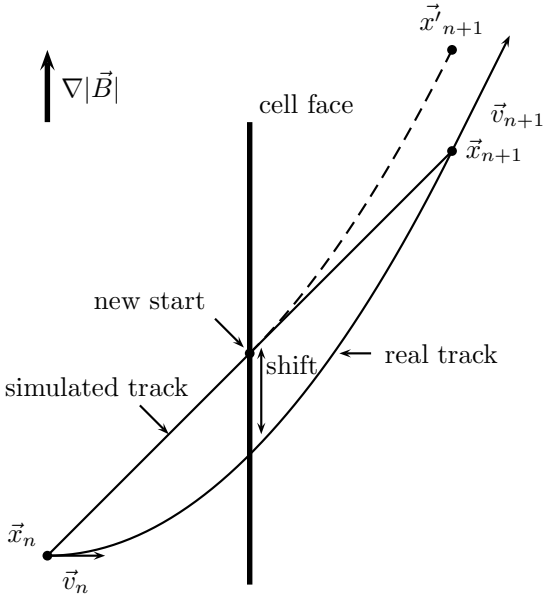

Figure 5.5 – *With the standard tracking algorithm the cell face crossing is at a wrong place. If the particle is moved there and the differential equation is solved again for the next point, one would end up at a wrong place at the end of the time step. Hence it is important to move along the same straight line during the whole time step.*

approximation will still be good but one does not have a unique set of parameters for the parabola anymore. In fact, the parameters will depend on the requirements posed to fit the parabola. If the real track is denoted by $\vec{x}(t)$ and the parabola by $\vec{p}(t)$ one could require that the time average of the deviation $\frac{1}{\Delta t}\int_0^{\Delta t} \vec{x}(t) - \vec{p}(t)\, dt$ vanishes such that on average one hits the correct point on the face. However this is not feasible since $\vec{x}(t)$ is not known. The best approximation that one has is the one from solving the equation of motion numerically (compare Eq. (5.15)), i.e.

$$\vec{f}(t) = \vec{x}_n + \vec{v}_n\, t + \frac{t^2}{6m}\left[\vec{F}(\vec{x}_n) + \vec{F}\left(\vec{x}_n + \vec{v}_n \cdot \frac{t}{2}\right) + \vec{F}\left(\vec{x}_n + \frac{t}{2}\left(\vec{v}_n + \frac{t}{2}\,\frac{\vec{F}(\vec{x}_n)}{m}\right)\right)\right]. \tag{5.23}$$

Requiring an easy parabolic fit to this function in order to reduce computational effort, two possibilities come to mind. The first one uses $\vec{x}_n$, $\vec{v}_n$ and $\vec{x}_{n+1}$ while the second one uses $\vec{x}_n$, $\vec{x}_{\frac{1}{2}} = \vec{f}\left(\frac{\Delta t}{2}\right)$ and $\vec{x}_{n+1}$. For the first case one obtains

$$\vec{p}_1(t) = \vec{x}_n + \vec{v}_n\, t + \frac{1}{2}\,\frac{\vec{F}}{m} t^2 \tag{5.24}$$

$$\vec{F} = \frac{1}{3}\cdot\left[\vec{F}(\vec{x}_n) + \vec{F}\left(\vec{x}_n + \vec{v}_n\cdot\frac{\Delta t}{2}\right) + \vec{F}\left(\vec{x}_n + \frac{\Delta t}{2}\left(\vec{v}_n + \frac{\Delta t}{2}\,\frac{\vec{F}(\vec{x}_n)}{m}\right)\right)\right] \tag{5.25}$$

where Eq. (5.25) is obtained from Eq. (5.15). The second set of conditions yields

$$\begin{aligned}\vec{p}_2(t) &= \vec{x}_n + (4\vec{x}_{\frac{1}{2}} - 3\vec{x}_n - \vec{x}_{n+1})\frac{t}{\Delta t} + 2(\vec{x}_{n+1} + \vec{x}_n - 2\vec{x}_{\frac{1}{2}})\frac{t^2}{(\Delta t)^2} \\ &= \vec{x}_n + \vec{v}_n\, t + \frac{t}{6\,m}\left[\vec{F}(\vec{x}_n)\, t + \vec{F}\left(\vec{x}_n + \frac{\Delta t}{4}\vec{v}_n\right)(\Delta t - t)\right. \\ &+ \vec{F}\left(\vec{x}_n + \frac{\Delta t}{2}\vec{v}_n\right)(2t - \Delta t) + \vec{F}\left(\vec{x}_n + \frac{\Delta t}{4}\left(\vec{v}_n + \frac{\Delta t}{4}\,\frac{\vec{F}(\vec{x}_n)}{m}\right)\right)(\Delta t - t) \\ &+ \left.\vec{F}\left(\vec{x}_n + \frac{\Delta t}{2}\left(\vec{v}_n + \frac{\Delta t}{2}\,\frac{\vec{F}(\vec{x}_n)}{m}\right)\right)(2t - \Delta t)\right].\end{aligned} \tag{5.26}$$

While for a single face crossing the maximal difference $|\vec{f}(t) - \vec{p}_i(t)|$ is important, minimizing the error for the whole simulation (i.e. for many face crossings) is equivalent to minimizing the time average $\overline{\vec{f}(t) - \vec{p}_i(t)} = \frac{1}{\Delta t}\int_0^{\Delta t} \vec{f}(t) - \vec{p}_i(t)\, dt$. Then, the question is, which of the two polynomials $\vec{p}_i(t)$ yields a smaller time averaged deviation from the real track. To calculate this, the force $\vec{F}$ is approximated by a first order Taylor series such that $\vec{F}(\vec{x}_n + \vec{\delta}) = \vec{F}(\vec{x}_n) + (\vec{\delta}\cdot\nabla)\vec{F}(\vec{x})\big|_{\vec{x}=\vec{x}_n}$. Using this, one obtains for the first parabola

$$\vec{f}(t) - \vec{p}_1(t) = \frac{t^2}{6\,m}\, J_{\mathrm{F}}(\vec{x}_n)\left[\vec{v}_n(t - \Delta t) + \frac{1}{4\,m}\left(t^2 - (\Delta t)^2\right)\vec{F}(\vec{x}_n)\right] \tag{5.27}$$

$$\overline{\vec{f}(t) - \vec{p}_1(t)} = -\frac{1}{6\,m}\, J_{\mathrm{F}}(\vec{x}_n)\left[\frac{(\Delta t)^3}{12}\vec{v}_n + \frac{(\Delta t)^4}{30}\,\frac{\vec{F}(\vec{x}_n)}{m}\right] \tag{5.28}$$

and for the second one

$$\begin{aligned}\vec{f}(t) - \vec{p}_2(t) &= \frac{t}{6\,m}\, J_{\mathrm{F}}(\vec{x}_n)\left[\vec{v}_n\left(t^2 + \frac{(\Delta t)^2}{2} - \frac{3}{2}t\,\Delta t\right)\right. \\ &+ \left.\frac{\vec{F}(\vec{x}_n)}{m}\left(\frac{1}{4}t^3 + \frac{3}{16}(\Delta t)^3 - \frac{7}{16}t(\Delta t)^2\right)\right]\end{aligned} \tag{5.29}$$

$$\overline{\vec{f}(t) - \vec{p}_2(t)} = -\frac{(\Delta t)^4}{2880\, m^2}\, J_{\mathrm{F}}(\vec{x}_n)\cdot\vec{F}(\vec{x}_n). \tag{5.30}$$

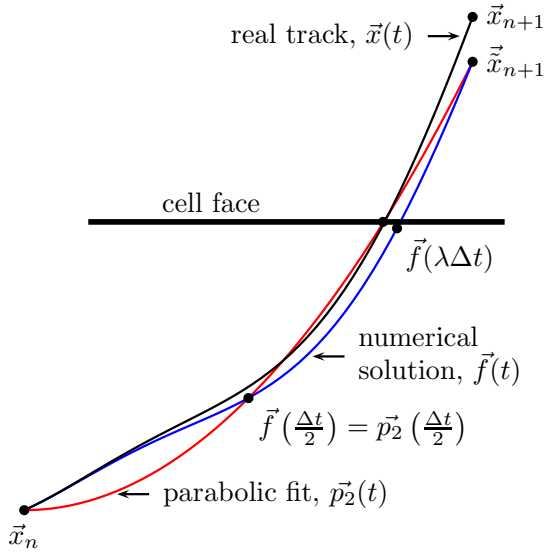

Figure 5.6 – *Scheme of the parabolic fit to find cell face crossings; In reality, the particle would move along the black track from $\vec{x}_n$ to $\vec{x}_{n+1}$. The numerical solution $\vec{f}(t)$, drawn in blue, deviates from the real track with time and leads the particle to $\tilde{\vec{x}}_{n+1}$. The parabola $\vec{p}_2(t)$ is fitted through the points $\vec{x}_n$, $\vec{f}(\Delta t/2)$ and $\vec{f}(\Delta t) = \tilde{\vec{x}}_{n+1}$ and is shown in red. The intersection of $\vec{x}(t)$ with the cell face is approximated by the intersection of $\vec{f}(t)$ with the cell face which in turn is approximated by the face crossing of $\vec{p}_2(t)$. Since $\vec{f}(t)$ and $\vec{p}_2(t)$ do not necessarily have to hit the face at the same time, the particle will not be moved exactly to the cell face. The resulting problems will be described in the next section.*

Comparing Eqs. (5.28) and (5.30) shows that on average the error is at least one order of magnitude smaller if one uses the second parabola. To check if the absolute values are good enough for the simulation, one can insert reasonable numbers. Considering a hydrogen atom with a mass of $m_\mathrm{H} = 1.67 \cdot 10^{-27}\,\mathrm{kg}$. Assuming a very strong B field gradient of $1000\,\mathrm{T/m}$ yields a force $|\vec{F}| = |\mu_\mathrm{B} \nabla |\vec{B}|| = 9.27 \cdot 10^{-21}\,\mathrm{N}$. Further assume that the force drops from this value to zero within $1\,\mathrm{mm}$ and that the force gradient is parallel to the force. It was reasoned before that the timestep shouldn't exceed $10^{-6}\,\mathrm{s}$. With these adversely chosen values one obtains

$$\overline{\vec{f}(t) - \vec{p}_2(t)} \approx 1.1 \cdot 10^{-11}\,\mathrm{m}. \tag{5.31}$$

This is much smaller than the diameter of an atom and clearly enough for the simulation. The strong Δt-dependence usually does the rest. For clarification, the procedure is shown in Fig. 5.6.

This kind of parabola fit is used in the program only to calculate the time when the face is hit. This time will then be used to recalculate the position and, more importantly, the velocity at the cell face with Eq. (5.15) in order to get the best possible results. To find the time of the cell face crossing one has to solve the same equations as for the simple straight line algorithm. Therefore write $\vec{p}_2(\lambda)$ as

$$\vec{p}_2(\lambda) = \vec{a}\lambda^2 + \vec{b}\lambda + \vec{x}_n \tag{5.32}$$

with

$$
\begin{aligned}
\vec{a} &= \frac{(\Delta t)^2}{6\,m}\left[\vec{F}(\vec{x}_n) + 2\vec{F}\left(\vec{x}_n + \frac{\Delta t}{2}\vec{v}_n\right) - \vec{F}\left(\vec{x}_n + \frac{\Delta t}{4}\vec{v}_n\right)\right. \\
&+ \left. 2\vec{F}\left(\vec{x}_n + \frac{\Delta t}{2}\left(\vec{v}_n + \frac{\Delta t}{2}\frac{\vec{F}(\vec{x}_n)}{m}\right)\right) - \vec{F}\left(\vec{x}_n + \frac{\Delta t}{4}\left(\vec{v}_n + \frac{\Delta t}{4}\frac{\vec{F}(\vec{x}_n)}{m}\right)\right)\right] \quad (5.33) \\
\vec{b} &= \vec{v}_n\,\Delta t + \frac{(\Delta t)^2}{6\,m}\left[\vec{F}\left(\vec{x}_n + \frac{\Delta t}{4}\vec{v}_n\right) - \vec{F}\left(\vec{x}_n + \frac{\Delta t}{2}\vec{v}_n\right)\right. \\
&+ \left. \vec{F}\left(\vec{x}_n + \frac{\Delta t}{4}\left(\vec{v}_n + \frac{\Delta t}{4}\frac{\vec{F}(\vec{x}_n)}{m}\right)\right) - \vec{F}\left(\vec{x}_n + \frac{\Delta t}{2}\left(\vec{v}_n + \frac{\Delta t}{2}\frac{\vec{F}(\vec{x}_n)}{m}\right)\right)\right]. \quad (5.34)
\end{aligned}
$$

Then the intersection of the parabola with a static cell face will happen at

$$
\lambda_{1/2} = -\frac{1}{2}\left(\frac{\vec{S}_\mathrm{f}\cdot\vec{b}}{\vec{S}_\mathrm{f}\cdot\vec{a}} \pm \sqrt{\left(\frac{\vec{S}_\mathrm{f}\cdot\vec{b}}{\vec{S}_\mathrm{f}\cdot\vec{a}}\right)^2 - 4\left(\frac{\vec{S}_\mathrm{f}\cdot(\vec{x}_n - \vec{C}_\mathrm{f})}{\vec{S}_\mathrm{f}\cdot\vec{a}}\right)}\right). \quad (5.35)
$$

If the cell face is moving but not rotating this will be

$$
\lambda_{1/2} = -\frac{1}{2}\left(\frac{\vec{S}_\mathrm{f}\cdot(\vec{b} - \vec{C}_\mathrm{f} + \vec{C}_{\mathrm{f},0})}{\vec{S}_\mathrm{f}\cdot\vec{a}} \pm \sqrt{\left(\frac{\vec{S}_\mathrm{f}\cdot(\vec{b} - \vec{C}_\mathrm{f} + \vec{C}_{\mathrm{f},0})}{\vec{S}_\mathrm{f}\cdot\vec{a}}\right)^2 - 4\frac{\vec{S}_\mathrm{f}\cdot(\vec{x}_n - \vec{C}_{\mathrm{f},0})}{\vec{S}_\mathrm{f}\cdot\vec{a}}}\right). \quad (5.36)
$$

The situation becomes more complex if the cell face is moving and rotating. In this case one has to find the roots of the third order polynomial

$$
p_3(\lambda) = A\lambda^3 + B\lambda^2 + C\lambda + D \quad (5.37)
$$

with

$$
\begin{aligned}
A &= \vec{a}\cdot(\vec{S}_\mathrm{f} - \vec{S}_{\mathrm{f},0}) \quad (5.38) \\
B &= \vec{a}\cdot\vec{S}_{\mathrm{f},0} + \vec{b}\cdot(\vec{S}_\mathrm{f} - \vec{S}_{\mathrm{f},0}) \\
C &= \vec{b}\cdot\vec{S}_{\mathrm{f},0} + (\vec{x}_n - \vec{r}_0)\cdot(\vec{S}_\mathrm{f} - \vec{S}_{\mathrm{f},0}) \\
D &= (\vec{x}_n - \vec{r}_0)\cdot\vec{S}_{\mathrm{f},0}
\end{aligned}
$$

where $\vec{r}_0$ is the same as in Eq. (5.18). With the definitions

$$
\begin{aligned}
q &= 2\,B^3 - 9\,ABC + 27\,A^2D && (5.39)\\
r &= B^2 - 3\,AC && (5.40)\\
Q &= \sqrt{q^2 - 4\,r^2} && (5.41)\\
R &= \sqrt[3]{\frac{1}{2}(Q+q)} && (5.42)
\end{aligned}
$$

and the definition of a phase θ with $\theta = 0$ for $q^2 \geq 4\,r^3$ and

$$
\theta = \begin{cases} \arctan\left(\frac{|Q|}{q}\right) & q > 0 \\ \arctan\left(\frac{|Q|}{q}\right) & q < 0 \\ \frac{\pi}{2} & q = 0 \end{cases} \qquad (5.43)
$$

if Q is imaginary, one can write the three solutions as

$$
\begin{aligned}
\lambda_1 &= -\frac{1}{3\,A}\left[B + \left(3\,|R| + \frac{r}{|R|}\right)\cos\left(\frac{\theta}{3}\right) - i\,\sin\left(\frac{\theta}{3}\right)\left(\frac{r}{|R|} + |R|\right)\right] && (5.44)\\
\lambda_{2/3} &= \frac{1}{3\,A}\left[-B + \left(3\,|R| + \frac{r}{|R|}\right)\cos\left(\frac{\theta \pm \pi}{3}\right) - i\,\sin\left(\frac{\theta \pm \pi}{3}\right)\left(\frac{r}{|R|} + |R|\right)\right]. && (5.45)
\end{aligned}
$$

In this notation it is easy to decide whether a solution is real or complex without using complex numbers for the calculation.

The location in the source code, where these changes have to take place are the functions *Particle::lambda()*. Since this is a class of the common infrastructure and several other particle types other than *DsmcParcel* are derived from it, the functions have to be overloaded for *DsmcParcels* in order to let other solvers undisturbed. But since some additional input parameters for the function are needed and the selection of the correct particle type is done by static cast, instead of overloading to save time, the class *Particle* has to be modified as well. In order to not disturb the other solvers, these additional input parameters are therefore declared as default arguments for the altered functions.

5.1.6 Sorting Particles Into the Correct Mesh Cell

Every simulated particle has a variable holding the index of the mesh cell currently containing the particle. This variable has to be changed at cell face crossings. This might sound trivial, but there is a difficulty: Because of the finite number of digits in the calculation, particles can be moved only to discrete places in the computational domain. Additionally, numerical errors make it impossible to tell exactly where the particle will be placed when it is moved to the cell face by the program. With these

two inaccuracies it is not possible to tell, on which side of the cell face the particle actually will end after the movement but as long as the face is an internal one the particles variable *celli* will always be changed to the index of the neighbor cell. This might be correct, but if the particle is still in the old cell, a call to the function *lambda()* in the next step calculating the fraction of the time step to the next cell face will yield the same cell face again with a tiny λ value. Then, the program will try to move the particle to the face, but numerical accuracy may hinder the particle to cross the face while the variable *celli* of the particle will be changed again. This can go forever and the program will be stuck. Another problem can occur if the particle is positioned slightly beyond a boundary face, which would result in the particle moving away from the computational domain. If the particle is tracked along straight lines, there is a clever way of avoiding these problems (see Fig. 5.7). In the first step, the λ value for all faces of the current cell will be calculated with the starting point taken as the cell center $\vec{C}$ and the end point beeing the final position $\vec{x}_{n+1}$. All faces with $0 < \lambda_i^{(1)} < 1$ will potentially be crossed and are appended to a list. In the second step there are two possibilities: Either the list of faces from the first step is empty and the particle can be moved to $\vec{x}_{n+1}$ or the list is not empty. In the latter case, for every face of the list the λ value will be calculated again but this time with the starting point beeing $\vec{x}_n$ yielding values $\lambda_k^{(2)}$. The face with the smallest $0 < \lambda_{k,\,\min}^{(2)} < 1$ will be the first to cross. But it can also happen that $\lambda_{k,\,\min}^{(2)}$ is negative meaning that the particle is not in the cell indicated by its variable *celli*. In this case, only *celli* is changed but the particle is not moved. This will correct the error and the particle will be moved in the next step. It can also happen that $\lambda_{k,\,\min}^{(2)} > 1$. In this case the particle is as well outside the correct cell. To correct the situation it is only moved to $\vec{x}_{n+1}$ and *celli* is kept constant.

The method described above does not work for tracking along curved tracks because the calculation of $\lambda_i^{(1)}$ does not yield all cell faces that might be crossed (see Fig. 5.8). Thus, the particle could move to other cells without knowing it. However, it is imperative to find the correct faces to cross since the selection of collision partners and the calculation of the fields depend crucially on the correct *celli* value. Therefore another algorithm is used if the particle is located within a magnetic field. In the first step $\lambda_i^{(1)}$ is calculated for all cell faces for a straight track between $\vec{x}_n$ and the cell center $\vec{C}$ (note that using the reversed order starting at $\vec{C}$ does not yield the necessary accuracy because of the clustering of the representable decimal numbers around zero ($1 - x = 1$ if $|x| < 10^{-15}$)). If it is $0 < \lambda_i^{(1)} < 1$ for at least one face i, then the particle is outside the cell. Now, the question is whether the face i is an internal face or a boundary face. If it is a boundary face, the particle is outside the computational domain and moved back into the direction of the cell center in small steps until it is in the cell again. If the face is an internal one, the particle will be moved across the face (for $\lambda_i^{(1)} < 10^{-12}$) or the variable *celli* of the particle will be changed otherwise. Note that for the shift into the direction of the cell center it is assumed that the cell is convex and that the correction will have to be made at approximately every second face crossing. The particle is generally moved not more than a few femtometers, so that even with a very large num-

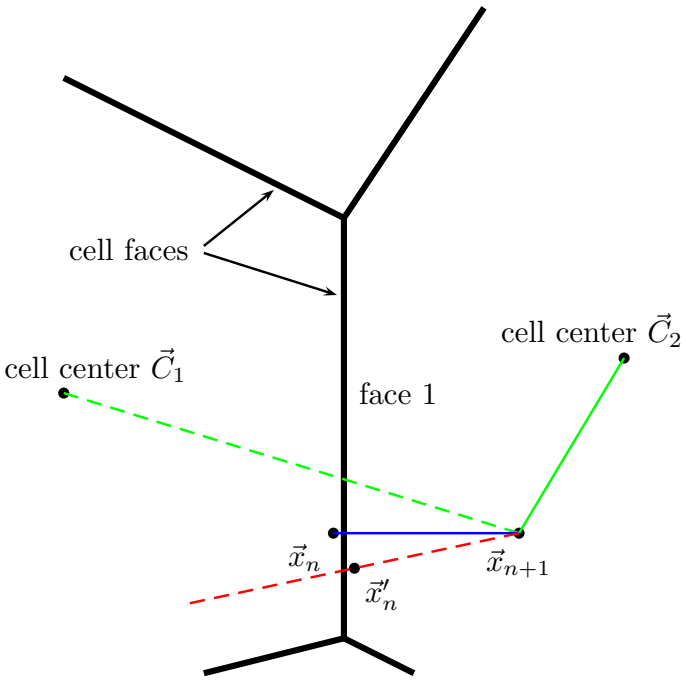

Figure 5.7 – *Finding cell face crossings works in two steps: If the particle starts at $\vec{x}_n$ and is sorted into cell 1, in the first step face 1 is found to be crossed and in the second step the distance to the cell face is calculated. If the particle is already sorted into cell 2 then there is no cell face crossing detected between the cell center $\vec{C}_2$ and $\vec{x}_{n+1}$ and the particle is simply moved to $\vec{x}_{n+1}$. The same is true if the particle starts at $\vec{x}'_n$ and is sorted into cell 2. If it starts at $\vec{x}'_n$ and is sorted into cell 1, the second step will yield a $\lambda < 0$ and the particle will be sorted into the neighbor cell.*

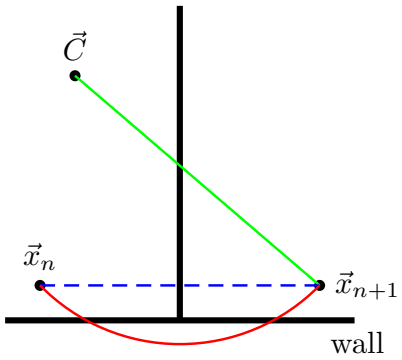

Figure 5.8 – *The straight line algorithm does not work in magnetic fields. The intersection of the red track with the wall would not be detected and the particle would not interact with the wall.*

ber of mesh cells no sizeable effect from this correction is expected. In magnetic fields another small shift of the particle position is applied to make sure that the particle is actually at the face which is not guaranteed otherwise with the parabola algorithm for multipoles of higher order than sextupoles, compare with caption of Fig. 5.6. After all these tiny shifts, the particle is in the correct cell again and the $\lambda_k^{(s)}$s can be calculated. The smallest face k for which it is $0 < \lambda_k^{(2)} < 1$ will then be crossed.

5.2 High Frequency Transitions

The high frequency transition units are idealized such that all high frequency transitions take place in one plane. If a particle crosses this plane, its hyperfine state will be changed with a certain probability into another hyperfine state. The plane can be chosen arbitrarily, independent of the mesh, i.e. it is not necessary to align the plane with the cell faces of the mesh. A high frequency transition unit is then described by the following parameters: The center of the plane and the normal vector together with the name of the shape of the plane (i.e. circle or rectangle) and efficiencies of the transitions for all particle species from the *typeIdList*. The efficiencies have to be specified in the following way (compare appendix C): The dictionary `efficiencies` contains as many subdictionaries as there are particle species and each subdictionary is a subdictionary as well with the only entry beeing the keyword `efficiency` and a list of probabilities. This list is meant as matrix where the element $M_{i,j}$ is the probability that the initial state i is changed to the final state j. Internally this matrix is stored such that in each row the element in a certain column is the sum of all previous columns of that specific row, i.e. $M_{i,j} = \sum_{k \leq j} M_{i,k}$. This is just a convenient choice which makes it easier to select the correct final hyperfine state. If a hyperfine transition unit is encountered, the final hyperfine state is initially set to 0 and a random number R_h is chosen. Afterwards the final state is increased as long as $R_\mathrm{h} < M_{i,f}$.

Finding a hyperfine transition unit is not trivial since the transition plane can lie within mesh cells but does not have to cross the cell completely. The procedure is as follows: First, find all faces of the current mesh cell where the particle is in and calculate λ_CF as described in Sec. 5.1.5 for every cell face. To this end the functions *Particle::findFaces()* are overloaded and changed such that the current particle position is used instead of the cell center. Then, for each of the hyperfine transition units λ_HF can be calculated with the same functions *DsmcParcel::lambda()*. If one λ_HF is smaller than the smallest λ_CF and additionally $0 < \lambda_\mathrm{HF} < 1$, the corresponding hyperfine transition unit is possibly hit by the particle. But it has still to be checked if the particle does not move around that interaction plane. This is done by the function *DsmcParcel::hfsTransitionProperties::insideHfRegion()* which can check this for different shapes of the plane. For example if the shape is declared as circle in `dsmcPropertiesDict`, then it is just checked if the distance between the position at the face and the face center is smaller than the declared radius. If the high frequency transition is encountered, a large negative face index (*facei*$=-1000-$*hfsId*) is returned to allow some functions to distinguish these faces from the normal cell faces and to allow them to retrieve their properties. Otherwise the encounter of the hyperfine transition unit is handled similarly to normal cell faces. The particle is moved to the face and the hyperfine state of the particle is changed with the probability described above.

All the changes made to the *dsmcParcel* class are shown in Fig. 5.9. Several new attributes were added to the various classes which made it necessary to extend the read

and write functions for particles. As described above, the function *DsmcParcel::move()* had to be extended as well to allow tracking along curved tracks which made it also necessary to overload the functions *Particle::lambda()* and *Particle::findFaces()*. The function *forceOnParticle()* calculates the force on a particle by a magnetic field at a specified position while the function *setPosAndU()* is used to place a particle at a cell face with correct velocity after λ is calculated, i.e. the particle is placed at position $\vec{f}(\lambda T)$ in Fig. 5.6. *hitFace()* is used to account for steps in the B field. *isAtHfFace()* checks if a particle is at a high frequency transition unit and if true calls *hitHfTransition()* which changes the hyperfine state of the particle. The two functions *numericHfState()* and *hfsToWhWs()* are used to convert the two used notations for the hyperfine state labeling into each other, namely the standard notation and the one using *whichHalf* and *whichState*. Finally, the remaining functions are used to calculate the sine and cosine of the hyperfine mixing angle for the particle for a specified field strength. The class *constantProperties* has some new attributes which will be needed in a new collision model which is described in Sec. 5.6. The new particle class *dsmcPatchParcel* is an extension of the standard particle used when particles have to count the number of patch hits as described in Sec. 5.4.

5.3 Calculation of the Polarization

For the calculation of the polarizations according to Eqs. (2.18) to (2.21), it is only necessary to know the relative occupation numbers of the various hyperfine states which are calculated during the run and the magnetic field strength. The polarization can therefore be determined after the run and is performed in the function *dsmcPlusFields::write()*. The extended class *dsmcPlusFields* will be compiled to the library *dsmcSpinPlusFields.so* and can be called via the configuration file `controlDict`, see appendix D. In the original version, this function was only used to calculate some fields like p, *translationalT* and U (velocity) from others and write them to disk. Now, it also calculates the mean of the hyperfine population numbers and the various polarizations. In order to get all the relevant data, a new instance of the *dsmcCloud* class is temporarily created and the *forceActiveList* is changed in such a way, that it marks only the cells where the magnetic field is purely homogeneous. If the field is inhomogeneous in a cell, it is not clear which field strength to take for the calculation and every choice might cause significant errors. Therefore the polarization is only calculated for homogeneous fields and is set to zero otherwise. For the program to know which magnets produce a homogeneous field, a *dictionary* is created within the function which holds the *typeName* of all magnet types with homogeneous field. This *dictionary* has to be extended if a new magnet with homogeneous field is programmed. When all the cells with constant magnetic field are marked, a new *volScalarField magB* is created giving the magnitude of the field strength for every cell and the ensuing calculation of the polarization is straightforward. Finally, the new polarization fields will be written to disk but only if it makes sense, i.e. the nuclear polarization will be written if $I > 0$ and the tensor polarization will be written if $I \geq 1$.

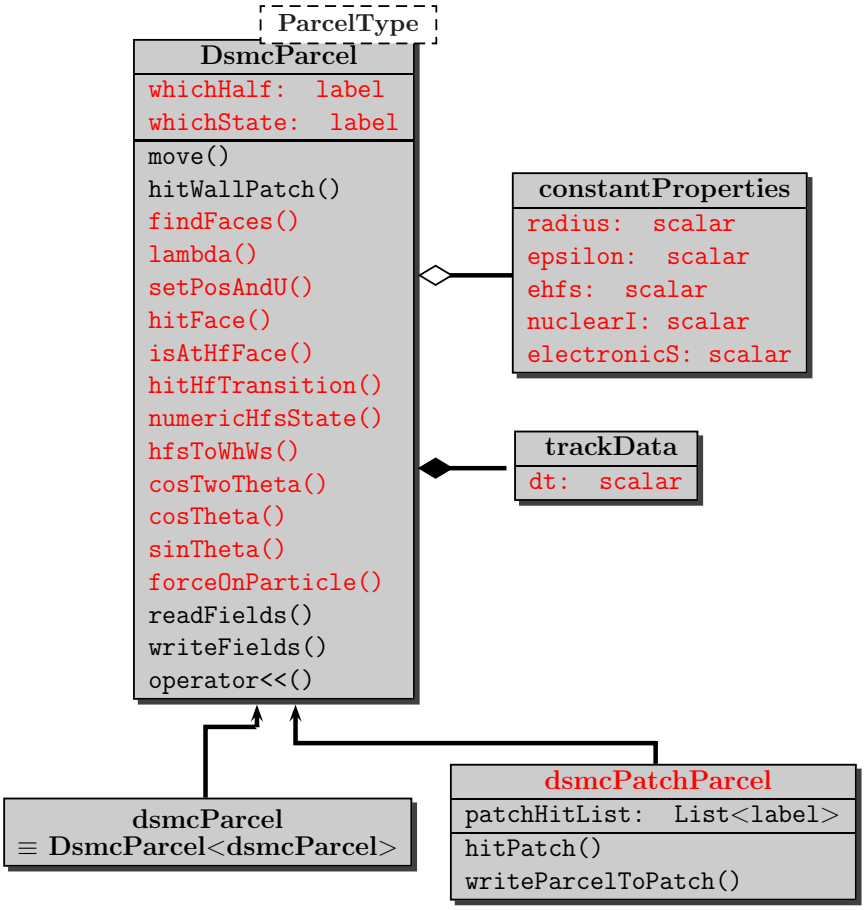

Figure 5.9 – *UML diagram of* dsmcParcel *class in solver* dsmcSpinModFoam *meant as an extension to the conventional* dsmcParcel *class of the solver* dsmcFoam *depicted in Fig. 4.4. Class attributes printed in red are new, functions printed red are also new except of findFaces() and lambda() which are overwritten versions of the ones from the class* Particle *from the common core of OpenFOAM. Functions printed in black are extended versions of the original ones. The class* dsmcPatchParcel *is completely new.*

5.4 Particles Counting Wall Hits

Studying gas properties that are influenced by wall collisions often requires some parameters of the wall interaction and the number of wall collisions. One can either predict the effect of the wall or, if some parameters are not known but the resulting property together with the number of wall collisions, one can deduce the wall parameters. That is why an option to count the number of wall hits for every particle was created. In order to achieve this, a new particle class *dsmcPatchParcel* was derived from the template particle class *DsmcParcel*, see Fig. 5.9. Each new particles has a list which shows how often the particle collided with the various patches of the mesh. The correct list entry is simply increased by one if a patch is hit during the simulation which is done by the overloaded function *hitPatch()*. In the file `dsmcProperties`

there is the option `writeParticlesAtPatch` which requires a list of patch names. For all patches specified in this list, some of the particle properties will be written to a file with name `Parcels_at_patch_patchName.dat` were patchName is the name of the patch, each time a particle hits this patch. This data can be used to extract some useful information like the collision age distribution. It should be noted though that in order to save storage space the number of wall collisions will always be set to zero at the beginning of a simulation, such that it will take some time to reach equilibrium, even when the simulation is a restart of an old simulation where the other gas properties like e.g. number density already are in equilibrium. Thus, one should only use data written to this file after a certain time, when equilibrium is already achieved. As basis for this selection one can use the time of the wall collision, which is written to this file as well.

Other changes that were made concern a new *cloud* class named *dsmcPatchCloud*, see Fig. 5.2. This class creates a vector field for every patch with name `patch_patchName`. The first component of the field shows the average number of collisions with the patch of all particles within a cell and the second component shows the standard deviation of this number. The third component gives the maximum number of collisions with the patch for all particles within a cell. These fields are written to disk during the simulation and allow the visualization of the spacial collision age distribution. For efficiency, the described functionality is completely implemented in new classes and runs using it can be started with the option `-countPatchHits`, i.e.

```
dsmcSpinModInitialise nProcs -countPatchHits
dsmcSpinModFoam nProcs -countPatchHits
```

for initialization and starting a run where `nProcs` is the number of processors to be used.

5.5 New Inflow Model

Simulating a complete ABS is a difficult task since there are regions with pressures that differ by several orders of magnitude. As a consequence, the time step Δt_{cD} must be set to the minimal value suitable for the high density region within the nozzle. However, for most parts of the computational volume, this value is orders of magnitudes too small. This isn't a problem for accuracy but only for execution time. The situation is similar with the size of the mesh cells which cannot be arbitrarily big in order to not loose spatial resolution. Hence, the ratio of real to simulated particles has to be relatively high to get at least some particles into a cell in the low density regions. Both factors lead to very high execution times of several months. This is very impractical, especially if an optimization algorithm shall be used to optimize some parts of the ABS which certainly needs hundreds of runs. The idea to circumvent this problem

is to split the computational volume into parts. This assumes that the upstream flow in the first part of the ABS does not change significantly if some magnets are placed differently in the low density part downstream. Then, one can start a calculation with the whole ABS until one has a reasonably good result at some cutting plane in the middle of the ABS. After that, one changes to a reduced mesh, were only the low density part of the ABS is represented and restarts the simulation with a much higher density of simulated particles. This requires a method to set the boundary values at the cutting plane appropriately and a suitable method is implemented in the new inflow boundary model *MixedInflow*. This inflow model can be used as a combination of the two existing inflow models *FreeStream* and *InOutflow* and additionally has the option `cellInternalValues` that can be set in the configuration file `dsmcProperties`. If this option is set to true, the boundary condition for this patch will be derived face by face from the cell values adjacent to the patch. In order to get good results, the time averaged mean fields from the first calculation should be used for this together with small cell sizes at the new inlet for a good spacial resolution. The time averaged fields will have to be present in the start time folder of the new case. They can be mapped like all other fields with the *mapFields* utility from one mesh to the other if a file with the correct name and header is already present in the start time folder of the new case.

For the implementation one should first note, that the mean fields are not read automatically by the program such that one has to do this first. For the fields *hfsPopulationNumbersMean1* and *hfsPopulationNumbersMean2* this is a bit tricky, since these fields are only present if there are enough particle species and one has to solve some issues with registering and deregistering the fields with the object registry. If this is done, one can start setting the boundary conditions. For that it is important to note, that the inflow models *FreeStream* and *InOutflow* use the following equation (from [59], Eq. 4.22) to decide how many particles to inject across a face:

$$\frac{\beta_i \dot{N}_i}{n_i} = \frac{1}{2\sqrt{\pi}} \left(e^{-s_i^2 \cos^2\theta} + \sqrt{\pi} s_i \cos\theta \left(1 + erf(s_i \cos\theta)\right) \right) \tag{5.46}$$

where $\beta_i = \sqrt{\frac{m_i}{2k_\mathrm{B}T}} = \frac{1}{v_{\mathrm{m,i}}}$ is the inverse of the most probable molecular thermal speed and $s_i = \frac{v_0}{v_{\mathrm{m,i}}}$ is the molecular speed ratio for particle type i. θ is the angle between the stream velocity $\vec{v}_0$ and the face normal of the inlet and $erf(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-y^2}\, dy$ denotes the error function. For the derivation of this equation it is assumed that the molecules have a thermal velocity distribution function, i.e. the gas is assumed to be in thermal equilibrium. This assumption is also used for the selection of the particle speed and internal energy. Hence, gases out of equilibrium cannot be simulated with this model. This is important since the gas in the ABS could well be out of equilibrium after the nozzle and the conditions should be checked. Another issue arises with setting the correct boundary fields because the used time averaged fields are averaged over all particle species. However, in the ABS, the different gases have different stream velocities and possibly different translational temperatures. Different temperatures would

stand against the equilibrium assumption but one can and should account for the different stream velocities. This could be done by calculating the gas velocity for every particle type separately but that would require a lot of additional storage space and would be unnecessary for most simulations. Therefore the different stream velocities $\vec{v}_i$ for each gas component are calculated from the mean fields. Depending on the number of particle species, this is more or less complicated. The easiest case where there is only one gas is trivial. For two gas components one can use two equations for the average momentum $\overline{\vec{p}}$ (`momentumMean`) per mesh cell and velocity $\overline{\vec{v}}$ (`UMean`) to obtain the two stream velocities where it is implicitly used that the average of the thermal velocity is zero:

$$\begin{aligned} \overline{\vec{p}} &= n_1 m_1 \overline{\vec{v}}_1 + n_2 m_2 \overline{\vec{v}}_2 && (5.47) \\ \overline{\vec{v}} &= \frac{n_1}{n_1+n_2}\overline{\vec{v}}_1 + \frac{n_2}{n_1+n_2}\overline{\vec{v}}_2. && (5.48) \end{aligned}$$

This yields

$$\begin{aligned} \overline{\vec{v}}_2 &= \frac{\overline{\vec{v}}(n_1+n_2) - \frac{\overline{\vec{p}}}{m_1}}{n_2\left(1-\frac{m_2}{m_1}\right)} && (5.49) \\ \overline{\vec{v}}_1 &= \frac{\overline{\vec{p}} - n_2 m_2 \overline{\vec{v}}_2}{n_1 m_1} && (5.50) \end{aligned}$$

where n_i are the proper components of the field `rhoNMean`. For three gas components the situation is more difficult. There are simply not enough equations to disentangle the mean fields. However, by assuming that the stream velocities of all three gases are parallel, the number of equations required reduces from 9 to 3. Then, the stream velocities of all gases can be written as $\overline{\vec{v}}_i = \overline{v}_i \frac{\overline{\vec{v}}}{|\overline{\vec{v}}|}$ and one can use the average kinetic energy per mesh cell (`linearKEMean`) as additional constriction. One has only to consider that taking the square and averaging do not commute, i.e. that $\overline{v}^2 \neq \overline{v^2}$. The needed average of the square of $\overline{v}$ can be obtained according to Eq. (3.83):

$$\begin{aligned} \overline{v^2} &= \frac{\beta^3}{\pi^{\frac{3}{2}}} \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \left(\dot{x}^2+\dot{y}^2+\dot{z}^2\right)\, e^{-\beta^2\left[(\dot{x}-\dot{x_0})^2+(\dot{y}-\dot{y_0})^2+(\dot{z}-\dot{z_0})^2\right]}\, d\dot{x}\, d\dot{y}\, d\dot{z} && (5.51) \\ &= \dot{x_0}^2 + \dot{y_0}^2 + \dot{z_0}^2 + \frac{3}{2\beta^2} \\ &= \overline{v}^2 + \frac{3}{2\beta^2}. \end{aligned}$$

Then, one has to solve the following equation system

$$|\overline{\vec{p}}| = \sum_{i=1}^{3} n_i m_i \overline{v}_i \tag{5.52}$$

$$|\overline{\vec{v}}| = \sum_{i=1}^{3} \frac{n_i}{n_1 + n_2 + n_3} \overline{v}_i \tag{5.53}$$

$$E_\mathrm{t} = \sum_{i=1}^{3} \frac{1}{2} n_i m_i \left(\overline{v}_i^2 + \frac{3}{2\beta^2(m_i)} \right) \tag{5.54}$$

where the resulting equations for $\overline{v}_i$ are very lengthy and will not be written here but they can be found in the source code in the function *MixedInflow::initialize()* in a form optimized for fast calculation with the freely available computer algebra program Maxima. The last problem with this is, that there are two solutions and one has to choose one. To this end, one should remember that the $\overline{v}_i$ are the projections onto the stream velocity $\overline{\vec{v}}$ and hence are assumed to be positive. Then it is required that the sum of all three $\overline{v}_i$ over all faces of the patch is maximal.

If the field `UMean` is not present, a mean velocity can still be calculated by dividing the fields `momentumMean` and `spaceAverageRhoMMean` or, if these fields are also not present, by dividing the fields `momentum` and `spaceAverageRhoM`. However, in these cases, the velocity will be the same for all particle species and in the latter case the fluctuation from one face to the other will likely be very high and the result of the simulation might be useless. For other boundary fields one has to do less. As mentioned above, the translational and internal temperatures are required to be the same for all gas components and now only for the case where the mean field is not available some minor calculations are needed. For the hyperfine states one simply has to read the corresponding fields. After that, the initialization is basically finished and the rest of the model is almost the same as for the already available inflow models.

5.6 New Binary Collision Models

The original *dsmcFoam* has only two binary collision models, namely *VariableHardSphere* and *LarsenBorgnakkeVariableHardSphere*, compare with Fig. 4.6. It was found in [72] that the DSMC method yields good results for the gas flow in an atomic beam source but the simulated temperatures were too high. M. Stancari [86] tried various input parameters for the VSS collision model based on the inverse power law potential of Birds DSMC94 code and could improve the results for the temperature at the cost of of a higher deviation of simulated and measured beam velocity. This might be caused by the approximation of the temperature dependence of the cross section by a simple power law. This is probably a bad approximation as the measured temperatures

are around $20\,\mathrm{K}$ where the relative velocity of the collision partners v_r is so low that an attractive part of the molecular potential can have a significant effect. In order to account for this, the VSS model is implemented on the basis of the Lennard-Jones potential, both with and without Larsen-Borgnakke energy redistribution. In both cases, the cross section is calculated according to Eq. (3.80) using Eqs. (3.79) and (3.77). A cutoff on K is set such that K is set to 0.1 if $K \leq 0.1$. This is done because Eq. (3.76) is only a fitfunction where the terms with higher i contribute for subsequently lower K such that more terms would be needed for lower K. The value $K = 0.1$ was chosen as limiting case because it is the lowest plotted value in both papers, [66] and [67] and at some lower value for K one can expect quantum effects to take over which are not considered in the calculations. For the correct use of Larsen-Borgnakke model, it would be necessary to have the correct number of translational degrees of freedom which is given by Eq. (3.109). However, this equation is not only complicated but also depends on the translational temperature which is not readily obtained during the calculation. Therefore, for simplicity the value obtained with the inverse power law model (Eq. (3.108)) is taken. For hydrogen with its huge collision relaxation numbers, no effect from this is expected in the ABS.

5.7 Spin-Exchange Collisions

The cross section for spin-exchange collisions is taken to be the same as for momentum exchange collisions and therefore a spin-exchange collision always occurs when two particles collide. Since the probability of a spin exchange does not depend on the momentum of the particle and the momentum exchange is assumed not to depend on the spins of the collision partners, it does not matter whether the spin exchange part is performed before or after the momentum exchange. But one problem is that the probability of a spin exchange depends on the magnetic field. Since the collision partners are generally not at the same position but only within the same (sub-)cell of the mesh, they experience different field strengths if the field is inhomogeneous. It looks problematic to use the fields at both particle locations and then mix the states as if they were at the same position. The problem is solved by using the field value in the middle of the two particles, i.e. $B = B\left(\frac{1}{2}(\vec{x}_1 + \vec{x}_2)\right)$ were $\vec{x}_i$ is the position of particle i. This method is a first order approximation and might cause a small error in the spin relaxation rate if the field variation is strongly nonlinear and the cell size is too big.

The B field and all the mixing angles of the states are calculated in the function *DsmcCloud::spinExchangeCollisions()*. The actual determination of the final hyperfine states is done in the base class of the binary collision model since it is the same for all collisions, i.e. in the function *BinaryCollisionModel::spinExchangeCollision()*. There, a random number $R_\mathrm{f} \in [0, 1]$ is generated and the cumulative probability $p = \sum_{c,d} p_{cd}$ of the different final states are calculated in a loop over all possible values. This calculation is aborted when $p > R_\mathrm{f}$ and the hyperfine states in place at abortion are taken as the final ones. The probability p_{cd} of ending in final state $|cd\rangle$ is according to Sec. 2.3.2 given by

$$p_{cd} = |\langle cd|U|ab\rangle|^2 = \frac{1}{2} M_{ab}^{cd} + |\langle cd|ab\rangle|^2 \tag{5.55}$$

where M_{ab}^{cd} is everything in the square brackets of Eq. (2.38) which can easily be computed.

5.8 New Wall Interaction Model

For studies of depolarization and recombination on the walls it is important to know, how often a wall was hit. Experimentally it can be shown, that the simple *WallInteractionModels* like *MaxwellianThermal* and *SpecularReflection* work only for some materials while they give a much too low number for others. There is an ongoing discussion in the community whether the reflected molecules from a wall should have a cosine or a cosine square distribution. In order to investigate the difference in a storage cell, a new *WallInteractionModel* named *MaxwellianThermalCosineExponent* was created that is basically equivalent to the standard *MaxwellianThermal* model, except that it has a distribution of

$$\frac{dn}{d\Omega} \propto cos^k(\theta) \tag{5.56}$$

around the wall normal vector. $k > 0$ is an arbitrary exponent that can be set in the configuration file `dsmcProperties` with the name `cosineExponent`. In order to obtain this cosine distribution, one has to set the velocity of the reflected molecules properly and this can be done with the correct distribution function. In an equilibrium gas it is

$$\frac{dn}{n} = f\, d\vec{v} = \frac{\beta^3}{\pi^{\frac{3}{2}}} e^{-\beta^2(v_x^2+v_y^2+v_z^2)}\, dv_x dv_y dv_z \tag{5.57}$$

or in polar coordinates

$$\frac{dn}{n} = \frac{\beta^3}{\pi^{\frac{3}{2}}} v^2 e^{-\beta^2 v^2} \sin\theta\, d\theta d\Phi dv. \tag{5.58}$$

With $d\Omega = \sin\theta\, d\theta d\Phi$ it is $\frac{dn}{d\Omega} = const.$ To get the proper θ-dependence, one has to include a factor $\cos^k\theta$ in the distribution function such that, after normalization it is

$$\begin{aligned} f\, d\vec{v} &= f_{\mathrm{v}}\, dv \cdot f_\theta\, d\theta \cdot f_\Phi\, d\Phi & (5.59)\\ &= \left[\frac{4\beta^3}{\sqrt{\pi}} v^2 e^{-\beta^2 v^2}\, dv\right] \cdot \left[(k+1)\cos^k\theta\, \sin\theta\, d\theta\right] \cdot \left[\frac{1}{2\pi}\, d\Phi\right] & (5.60) \end{aligned}$$

where the integration over θ stretches only from 0 to $\frac{\pi}{2}$. Sampling from f_Φ is simple since it is equally distributed and it is $\Phi = 2\pi R_\mathrm{f}$ with $R_\mathrm{f} \in [0,1]$ an equally distributed random number. For θ the cumulative distribution function $F_\theta = \int_0^\theta f_{\theta'}\, d\theta'$ is

$$F_\theta = 1 - (\cos(\theta))^{k+1}. \tag{5.61}$$

Setting this to an equally distributed random number $R_\mathrm{t} \in [0,1]$, one obtains

$$\cos\theta = R_\mathrm{t}^{\frac{1}{k+1}} \tag{5.62}$$

using that $1 - R_\mathrm{t}$ is also equally distributed in $[0,1]$. Now also θ is fixed and only the magnitude v of the velocity is missing. The method used for θ does not work since the cumulative distribution function F_v cannot be inverted. The method of choice here is to use the velocity vector of the old *MaxwellianThermal* method and use its magnitude, since both values must be the same. Then, the velocity after a wall collision is

$$\vec{v} = v\cos\theta \cdot \vec{n} + v\sin\theta\cos\Phi \cdot \vec{t_1} + v\sin\theta\sin\Phi \cdot \vec{t_2} + \vec{v}_\mathrm{w} \tag{5.63}$$

where $\vec{n}$ is the normal unit vector of the wall, $\vec{t_1}$ and $\vec{t_2}$ are the two tangential unit vectors and $\vec{v}_\mathrm{w}$ is the velocity at the wall specified as boundary condition.

5.9 Recombination

Recombination of hydrogen atoms to molecules or more general, chemical reactions at surfaces, are implemented in the new run time selectable submodel *ChemicalSurfaceReactionModel*. There are only two simple models available, namely *simpleLangmuirHinshelwood* and *NoReaction*, compare Fig. 5.10.

The models are implemented in a way such that different reactions can take place at a wall patch and it can be taken into account that different walls might consist of different materials with different reaction properties. Therefore, one has to provide a list of all reactions and their properties in the file `dsmcProperties`. A simple example is shown in appendix C. Each reaction needs a list of reactants and a list of products where all particles have to be listed separately, even if one particle occurs five times in the list. Additionally, some model dependent reaction properties are needed which can be specified in a dictionary for groups of patches where each patch group needs an unambiguous name that is of the form `patchGroup_1` where the last number is meant as a counter that can either start from 0 or 1.

Each model has to implement the function *gamma* which calculates the probability for a reaction which is simply a constant in the *simpleLangmuirHinshelwood* model but could be much more complex as described in Sec. 2.3.1. The second function that has to

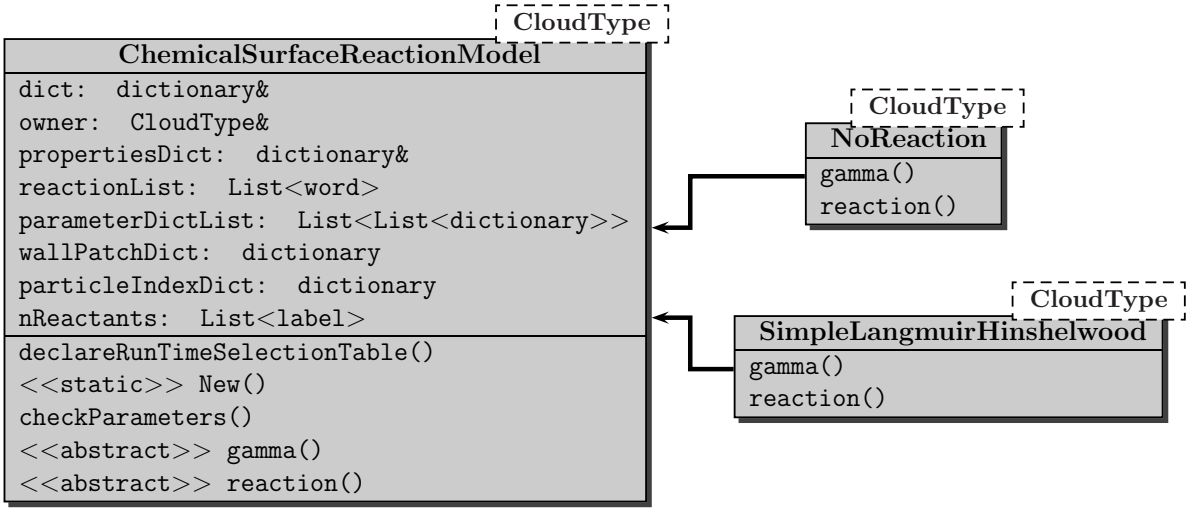

Figure 5.10 – *UML diagram of submodel* ChemicalSurfaceReactionModel *in* dsmcSpinModFoam. *As usual, all setter and getter functions as well as constructors and destructors are omitted.*

be implemented by all models is the function *reaction* that actually performs the reaction. Both functions are called when a wall is hit, i.e. by the function *DsmcParcel::hitWallPatch()*. In this function, a random number $R_1 \in [0, 1]$ is drawn and compared with the sum of the γ_i for each reaction. If $\sum_i \gamma_i < R_1$ a reaction takes place and it is decided which one. Then, the function *reaction* is called which has to handle the stoichiometry and set the properties of the reaction products. The first part has to make sure, that the particle number stays constant during the simulation. This is not quite as easy as it might seem, since the reaction is performed, when one reactant hits a wall. If there are more particles reacting together, one has to delete several particles and the question is which ones to choose. It is by no means certain that all the necessary particles are in the adjacent cell at the time of the reaction and one cannot simply delete particles from somewhere in the whole volume. The chosen solution is to change the incoming particle into one of the products and add the other products if there are more than one if

$$R_2 < \frac{1}{N_\mathrm{R}(i)} \tag{5.64}$$

where $R_2 \in [0, 1]$ is a second random number and $N_\mathrm{R}(i)$ denotes the number of reactants of particle species i (= *typeId* of the incoming particle). If $R_2 > \frac{1}{N_\mathrm{R}(i)}$, the incoming particle will be deleted. If a reaction takes place, the particle properties of the products will be set such that they are in thermal equilibrium with the wall and have a cosine theta velocity distribution around the normal vector of the wall. Further, their hyperfine state is chosen such that the products are on average unpolarized although this might not be the case in reality as described in Sec. 2.3. However, there is currently

no generic way to describe molecule polarizations which would allow to handle polarized products in the simulation. One should also note that the used selection method for the reaction assumes that the reaction speed isn't limited by the absence of one of the reactants at the wall, i.e. the reaction will always take place if condition (5.64) is fulfilled, no matter if there are actually enough particles of all reactants around. In order to drop this limitation, one would have to record the particles hitting faces of a wall patch. One possibility could be to create a *volVectorField::GeometricBoundaryField* which in its elements stores the number of particles that have already hit the face. The components of the vector would be used for the various particle species. Then, the reaction would take place only if the wall face was already hit by enough particles of all types taking part in the reaction and the function *reaction()* would have to decrease these numbers by the number of occurrence in the list of reactants.

6 Simulation Results

Every new program needs thorough testing and the best way to check a program is to test only one new feature at a time. This enables one to identify and locate most of the errors. The task was executed constantly during the course of the work after some initial simulations of the gas flow through an ABS to check whether the DSMC method and especially OpenFOAM works for this case. However, some errors arising from the interplay of various new components may not be found by this method and more complicated simulations are necessary for this. These are much more difficult to interpret and errors are therefore much more difficult to detect and locate. The fact that corrections for one error may again be erroneous, possibly such that the correction seems to work but suddendly another program component is negatively affected, is especially inconvenient. This would require to test all features again after a correction when one cannot exclude that other program components are affected by a change in the code. The major difficulty with this is the long time it takes for a single run. Typical runtimes lie in the order of several days to months. For this reason, the calculations were executed mainly on three different computers, two older ones with $3\,\mathrm{GHz}$ Intel dualcore processors and a new one with two 2.4 GHz Intel Xeon hexacore processors each with two virtual processors allowing for a total of 24 parallel processes. All computers use some distribution of the OpenSuse operating system with version numbers between 11.1 and 12.2. Still, although many tests were performed, exhaustive testing, and especially comparison with measurements, was not possible because late in the work some major errors were detected which rendered many of the previous results more or less useless. Nonetheless, all new features were tested and some results will be shown and compared with theoretical predictions or measurements.

All meshes for the simulations were created with Gmsh, only the more complicate ones for the storage cell were created with SALOME and then imported into OpenFOAM (see appendix B). Because of the long execution time, many simulations were run at low pressure such that the mean free path length of the particles was generally quite long allowing for large cell sizes. However, in order to retain some spacial resolution, the cells were usually made much smaller. This in turn increases the number of required simulated particles in the simulation, since there should ideally be about 20 particles per cell. This requirement was not always fulfilled since it is difficult to match cell size, gas density, and the need for spacial resolution with computational effort. The most extreme cases occur in the simulations of the ABS where the mean free path length is between $\lambda_0 \approx 10^{-5}\,\mathrm{m}$ in the nozzle and $\lambda_0 \approx 100\,\mathrm{m}$ in the last vacuum chamber. The first number is so small that millions of cells would be required only for the nozzle to satisfy the requirement of cell sizes much smaller than the mean free path

length. This makes the meshes very large and can lead to memory exhaustion. As a compromise, a cell size of several tenth of millimeters was chosen for the nozzle. This might lead to artifacts in the expansion region after the nozzle and should be investigated in detail with simulations of only the nozzle and the expansion with different cell sizes. The cell size for the vacuum chambers was set at several millimeters such that the central beam can still be resolved although the exact width and structure will be lost. The bigger problem with this is, that with the low density prevailing in this region, only very few particles move through the region leading to an average density of simulated particles per cell below 10^{-5}. This presents a clear violation of the DSMC requirements, but can be justified with the low gas density. Since the mean free path length is much bigger than the ABS, there are practically no collisions in reality and therefore almost no collisions will be neglected through the selection mechanism of the program looking only in a single cell for possible collision partners. The downside is that even with time averaging switched on, it is almost impossible to obtain reasonable statistics for the low density region. Increasing the ratio of simulated particles to real particles is not an option either, because it increases significantly the execution time and it takes much longer to even reach an equilibrium state from where it makes sense to switch on time averaging, meaning an overall performance loss. The last steering parameter that can be used to influence the runtime is the timestep Δt_{cD} in the file `controlDict`. This is however determined by the mean collision time in the nozzle and cannot be chosen much larger without significantly influencing the result of the gas expansion after the nozzle. The only real possibility for reducing these problems would be to have timesteps matching the local mean collision time which however would require major changes to the program as mentioned earlier in Chap. 3.

6.1 Trial Simulation of Test Stand

In order to test whether the settings described above are justifiable, one of the first tests was the flow of H_2 gas through the ABS without magnets solved with the *dsmcModFoam* solver. This could be compared with measurements at a test stand shown in figure 6.1 and described in [15].

For the simulation, an axisymmetric test case was set up with a simplified geometry (see Fig. 6.2). Since OpenFOAM is a three dimensional solver, it requires a wedge-shaped section of the geometry. Here, as for all other axisymmetric simulations, a wedge with 5° opening angle was used together with the boundary condition `cyclic` in the configuration file `boundary` for the sides of the wedge. The radius (distance from center line to pumps) for this first simulation was taken as 5 cm, and the complete outer walls were defined as pumps (see Fig. 6.3), where the number densities set as boundary conditions were calculated with $p = nk_{\mathrm{B}}T$ with $T = 300\,\mathrm{K}$ from pressures measured in the vacuum chambers of the teststand (see Tab. 6.1). Since the simulated inward flux is not known before the end of the simulation, the value for a flux $Q = 1\,\mathrm{mbarl/s}$ was taken for all simulations. All wall temperatures as well as the temperature of the

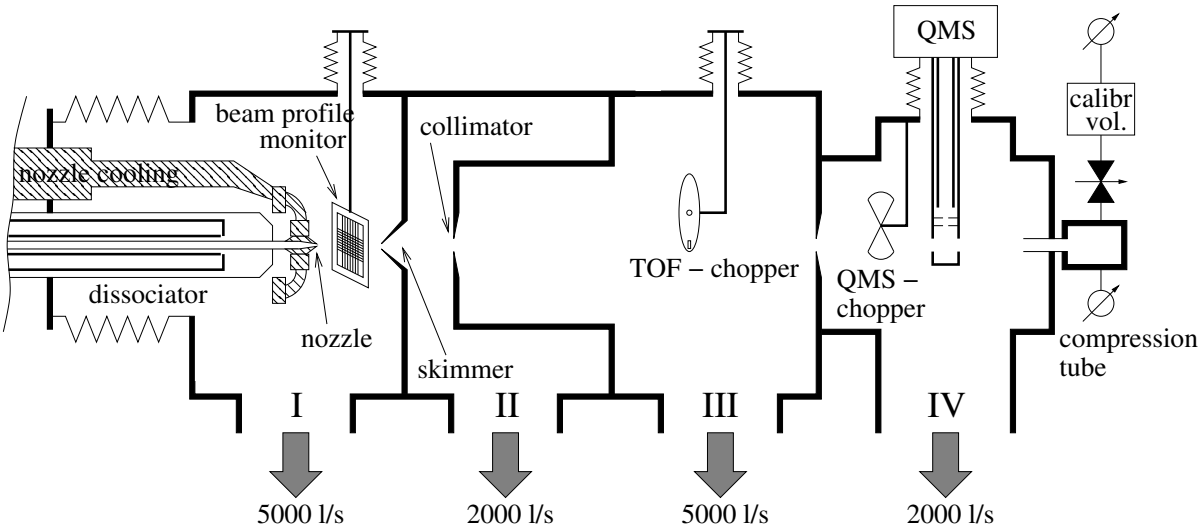

Figure 6.1 – *Drawing of the test stand ([14]); The numbers I to IV are for the four vacuum chambers and the arrows and numbers below identify turbo pumps with the indicated pumping speed.*

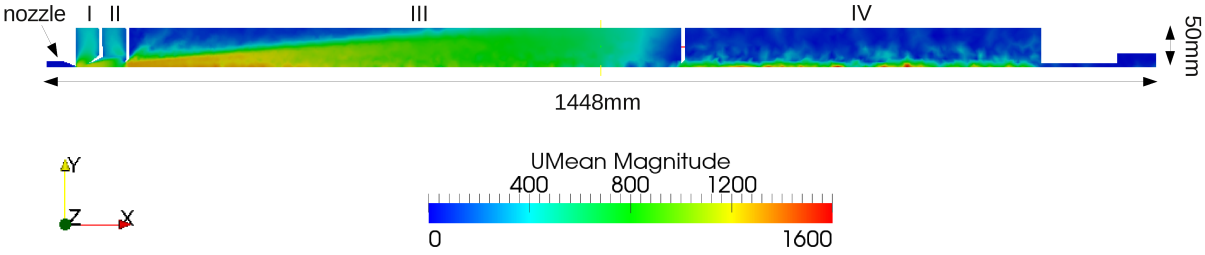

Figure 6.2 – *Simulated velocity in the upper half of the test stand; The symmetry axis is aligned with the lower edge of the plotted region and the dimensions of the test stand model are plotted to scale. Skimmer and collimators are like in Fig. 6.1 and the numbers I-IV again indicate the vacuum chambers. One can see that there are still significant statistical fluctuations in chamber IV.*

incoming gas were set to $T = 300\,\mathrm{K}$ except of the conical part of the nozzle which was cooled to $T = 100\,\mathrm{K}$.

The number density at the inflow was set to $1 \cdot 10^{22}\,\mathrm{m}^{-3}$ which leads to a flux $Q = k_\mathrm{B} T \int \vec{v} n \, d\vec{A}$ of just over $1\,\mathrm{mbar\,l/s}$. The time step was set to $\Delta t_\mathrm{cD} = 5 \cdot 10^{-10}\,\mathrm{s}$ and every simulated particle represents 10^{11} real particles. As wall interaction model `MaxwellianThermal` was used, which implements an undirected reflection from the wall with total thermalization. As binary collision model the newly implemented VSS model based on the Lennard-Jones potential described in Sec. 5.6 was used. The simulation was executed for about three months on one of the dualcore PCs and during

Table 6.1 – *Measured pressures in the vacuum chambers of the test stand for different inward fluxes through the nozzle with a nozzle-skimmer distance of* 30 *mm (from [14])*

Q mbarl/s	p_1/mbar	p_2/mbar	p_3/mbar	p_4/mbar
0	$2.0 \cdot 10^{-7}$	$6.0 \cdot 10^{-8}$	$5.0 \cdot 10^{-8}$	$1.3 \cdot 10^{-7}$
1	$2.1 \cdot 10^{-4}$	$1.8 \cdot 10^{-5}$	$7.0 \cdot 10^{-7}$	$1.7 \cdot 10^{-7}$
2	$5.2 \cdot 10^{-4}$	$2.8 \cdot 10^{-5}$	$1.4 \cdot 10^{-6}$	$2.1 \cdot 10^{-7}$
3	$8.6 \cdot 10^{-4}$	$4.3 \cdot 10^{-5}$	$2.2 \cdot 10^{-6}$	$2.6 \cdot 10^{-7}$
4	$1.3 \cdot 10^{-3}$	$6.0 \cdot 10^{-5}$	$3.0 \cdot 10^{-6}$	$3.1 \cdot 10^{-7}$
5	$1.6 \cdot 10^{-3}$	$7.5 \cdot 10^{-5}$	$2.2 \cdot 10^{-6}$	$1.8 \cdot 10^{-7}$

that time 0.1161 s of real time were simulated. In order to obtain a result with not too much statistical fluctuation, time averaging was switched on very early, certainly before a steady state was reached, which leads to poor results in the third and fourth vacuum chamber. Generally, there are three ways to judge whether steady state is reached, i.e. when it makes sense to switch on time averaging. The first is simply to look at the fields during the simulation and check if they vary from one time step to another. This is a very crude method, since unaveraged fields show a lot of statistical fluctuations from one mesh cell to another, and one cannot really tell whether differences originate from statistical fluctuations or coherent changes over time. The second method is proposed in [81] and uses the total number of simulated particles at a time as criterion. With a small change to the database file `$WM_PROJECT_DIR/bin/foamLog.db` where `$WM_PROJECT_DIR` is an environment variable set by OpenFOAM, the *foamLog* script delivered with OpenFOAM can extract the particle number for every time step from the logfile and one can plot it with Gnuplot. During the simulation, the particle number will approach a constant value. This is much easier to see and usually yields a later start time for averaging. The third method is a rule of thumb after which the steady state may be reached after the particles had time to cross the whole simulated volume about three times, where the average velocity can be taken for the calculation. This may be the safest method and usually leads to the largest time, for some cases about an order of magnitude larger than the other methods. Usually the second method was used during the work but in some cases like here, the first one was used because the *foamLog* script can have problems with very large logfiles of several Gigabyte if there is not enough main memory available.

The result of this first major simulation was used to plot velocity (Fig. 6.2), temperature (Fig. 6.3) and pressure (Fig. 6.4) and to extract some macroscopic values which were compared with measurements. The results are listed in Tab. 6.2. One can see that the simulated results are in good agreement with the measurements, but to really understand the comparison, it is important to mention how these values are obtained. First, the pressures in the various vacuum chambers are measured close to the chamber walls with an uncertainty of about $50\,\%$. The simulated chambers have a different shape and

Table 6.2 – *Comparison of measured and simulated values for the test stand in Fig. 6.1 operated with H_2; The index of the pressure indicates in which vacuum chamber it was measured, v_x is the velocity component parallel to the symmetry axis and T denotes the translational temperature. The errors for the simulated values are not known exactly but probably in the single per cent range.*

	measured	simulated
p_1/mbar	$2.1 \cdot 10^{-4}$	$3.4 \cdot 10^{-4}$
p_2/mbar	$1.8 \cdot 10^{-5}$	$2.7 \cdot 10^{-5}$
p_3/mbar	$7.0 \cdot 10^{-7}$	$8.0 \cdot 10^{-7}$
p_4/mbar	$1.7 \cdot 10^{-7}$	$1.7 \cdot 10^{-7}$
v_x/(m/s)	1274 ± 8.4	1290
T/K	19.03 ± 1.11	16.5

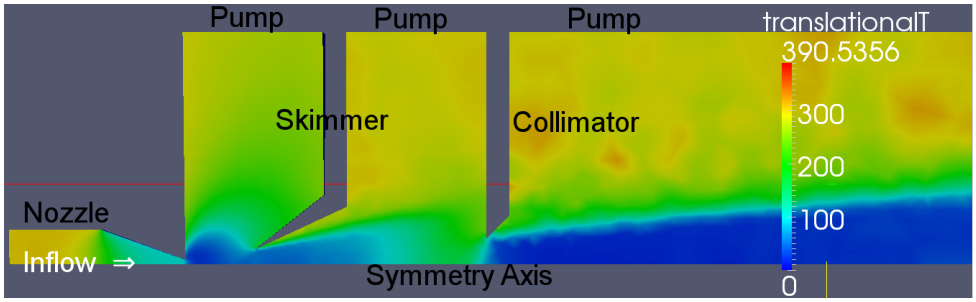

Figure 6.3 – *Close up view of the simulated translational temperature in the upper half of the test stand close to the nozzle.*

a smaller radius and the values were extracted only 4 cm off axis and a small gradient parallel to the axis was averaged out. The velocity and temperature measurements are performed with a chopper that allows a short pulse of gas to pass and this pulse lengthens over time until it reaches the quadrupole mass spectrometer (QMS) which measures the time-dependent intensity. The velocity can then be obtained by the mean flight time of the particles, while the pulse shape determines the temperature. The details can be found in [15]. It is important to note that with this measurement method the rest gas influence is suppressed. This cannot be achieved with the standard DSMC output, where all field values are averages over all gas molecules within a mesh cell, and the beam cannot be separated from the rest gas. Therefore, the temperature and velocity values are extracted on axis over a few centimeters behind the collimator where the density ratio of the beam to restgas is maximal. At this position, the temperature is lowest and increases slightly with increasing distance from the collimator. One can also expect that the steady state was reached during the simulation for these positions such that the above mentioned error from starting time averaging too early can be neglected here.

Figure 6.4 – *Close up view of the simulated pressure in the upper half of the test stand close to the nozzle.*

This first simulation looks very promising and it seems that the new binary collision model can solve the problem of too high simulated temperatures found in [15]. However, it should be stressed that about two years after this simulation two errors were found in the collision model (and were then corrected). First, the formulas for viscosity and diffusion cross section (3.78 and 3.79) were interchanged and second, the factor $\frac{2}{3}$ in Eq. (3.79) was omitted. The effect is that the simulated total cross section was somewhat higher than it should have been and the scattering angle was also inaccurate. The effect of the cross section would look like if the pressure was a bit too high and the effect of the scattering angle is not known. Both affect only the expansion behind the nozzle since there are almost no collisions far from the nozzle. Still, the effect on the simulated values for $\vec{v}$, T and p is not clear and the simulation was not repeated with the correct collision model because of time constraints. However, a simulation of a complete ABS with more effects and correct collision model was performed and the results can be found in Sec. 6.7.

6.2 Test of Motion in Magnetic Fields

The next step was testing the motion of particles in magnetic fields. A simple test case was set up for this, where atomic hydrogen in a single hyperfine state is expanded through a nozzle and an unrealistically strong sextupole magnet with a pole tip field of $1500\,\mathrm{T}$ and a length of $4\,\mathrm{cm}$ is placed $1\,\mathrm{cm}$ behind the nozzle. The magnet does not have any walls, only its field is active within this region of $3\,\mathrm{cm}$ radius. Further, the complete boundary is open and acts as pump respectively as source for atoms. Only the nozzle has a real wall, where particles can be reflected from. These settings are meant to simplify the analysis. The test case is again axisymmetric and the pressure in the upper half of the vessel for injected hyperfine state $|1\rangle$ is shown in Fig. 6.5 while the same is shown for injected state $|3\rangle$ in Fig. 6.6. One can clearly see that atoms in state $|1\rangle$ are focused by the magnet while atoms in state $|3\rangle$ are defocused. The magnet seems to be so strong that the atoms in state $|1\rangle$ cannot even reach the outer edge of the B field but are completely reflected at a smaller radius and increase the pressure inside

the magnet. Atoms in state $|3\rangle$ are rapidly pushed out of the magnet, thus lowering the pressure inside.

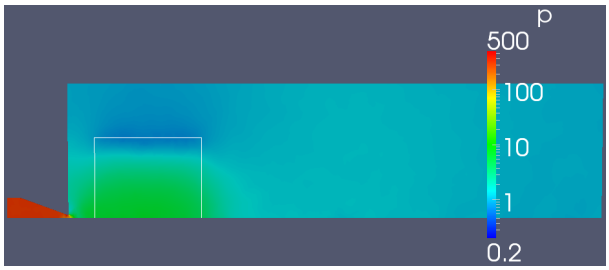

Figure 6.5 – *Pressure of H atoms in state $|1\rangle$ moving through a strong B field which is active within the region demarcated by the thin white line.*

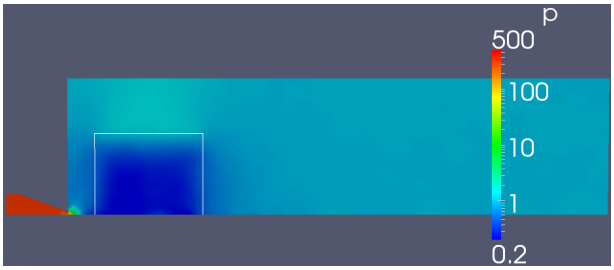

Figure 6.6 – *Pressure of H atoms in state $|3\rangle$ moving through a strong B field which is active within the region demarcated by the thin white line.*

6.3 Test of Collision Age Determination and New Wall Collision Model

Testing the new collision age determination from Sec. 5.4 was executed with a model of the storage cell shown in Fig. 2.7. At the ABS feeding tube a pressure of H_2 molecules of $2.7 \cdot 10^{-3}$ Pa at 300 K was set and at the cell and BRP outlet the pressures are about three orders of magnitude smaller leading to an effusive flow through the cell with the `InOutflow` inflow model. In one case the `FreeStream` model was used with a gas velocity at the inlet of $v = 1290\,\mathrm{m/s}$ obtained from the simulation described in Sec. 6.1. For this run with the `MaxwellianThermal` wall interaction model, Fig. 6.7 shows the average number of wall collisions per mesh cell in the ABS feeding tube and the cell. Figure 6.8 shows the same in a cut through the feeding tube and the BRP tube. In both pictures the total number of wall collisions is shown but it is also possible to plot only the number of wall collisions with a single wall patch to investigate where the collisions take place.

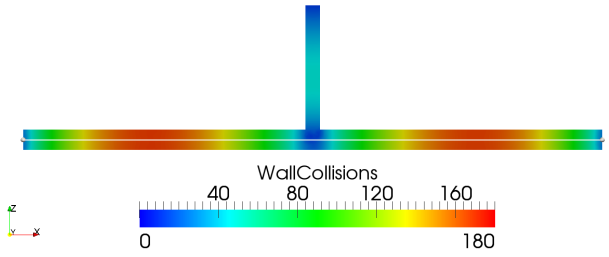

Figure 6.7 – *Total number of wall collisions in the ABS feeding tube and the cell; The average number of 116.5 seen by the COSY beam is calculated along the white line.*

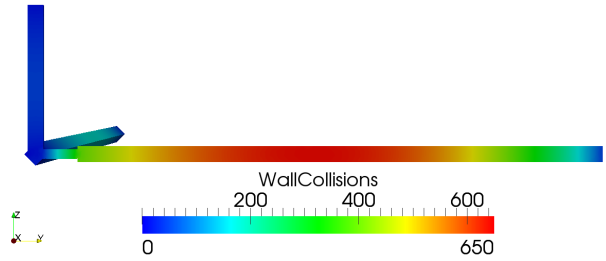

Figure 6.8 – *Total number of wall collisions in the ABS feeding tube and the BRP tube*

Table 6.3 – *Number of wall collisions in a storage cell for different wall interaction models.*

wall interaction model	$\overline{N}_{\mathrm{w,cell}}$	$\overline{N}_{\mathrm{w,BRP}}$
`SpecularReflection`	92.4	66.2
`MaxwellianThermal`	81.8	330.3
`MaxwellianThermal/FreeStream`	116.5	109.9
`MaxwellianThermalCosineExponent` $(k = 0)$	58.6	435.7
`MaxwellianThermalCosineExponent` $(k = 1)$	118.8	968.2
`MaxwellianThermalCosineExponent` $(k = 2)$	185.4	1468.6

An important number for experiments is the average number of wall collisions of particles, both in the cell, which is seen by the COSY beam, and at the exit of the BRP tube, which influences the polarization determined by the BRP and which has to be related to the polarization seen by the COSY beam. The average number of wall collisions along the cell $\overline{N}_{\mathrm{w,cell}}$ has to be weighted with the number density n, since the density decreases from the center outwards. Then it is

$$\overline{N}_{\mathrm{w,cell}} = \frac{\int_{-0.2}^{0.2} n N_{\mathrm{w,cell}}(x)\, dx}{\int_{-0.2}^{0.2} n\, dx}. \tag{6.1}$$

This was calculated for different wall interaction models, both with the new `MaxwellianThermalCosineExponent` model (Sec. 5.8) with different exponents k as well as with the models of OpenFOAM (`MaxwellianThermal` and `SpecularReflection`). All but one run were performed with effusive flow and the last one was with `FreeStream` as inflow model as described above. The results of these calculations are listed in Tab. 6.3.

For the polarization measurement in the BRP, it is not the number of wall collisions of the particles within the BRP tube that matters, but the number of wall collisions of particles actually moving out of the BRP tube into the BRP. All particles leaving through a patch are dumped to file together with their number of wall collisions, and the average as well as the collision age distribution can be calculated from this information. The average number of wall collisions of all particles leaving the end of the BRP tube $\overline{N}_{\mathrm{w,BRP}}$ is also listed in Tab. 6.3 and the collision age distribution for the `MaxwellianThermal/FreeStream` case is shown in Fig. 6.9.

Interestingly, the average number of wall collisions in the cell $\overline{N}_{\mathrm{w,cell}}$ is higher than the corresponding number $\overline{N}_{\mathrm{w,BRP}}$ at the BRP outlet for the cases with the `SpecularReflection` and `MaxwellianThermal/Freestream` settings, although the BRP tube is much longer. This may best be understood with the `SpecularReflection` model where the angle of incidence at the wall is equal to the angle of reflection. This allows mainly particles with a flight direction essentially parallel to the BRP tube to enter the

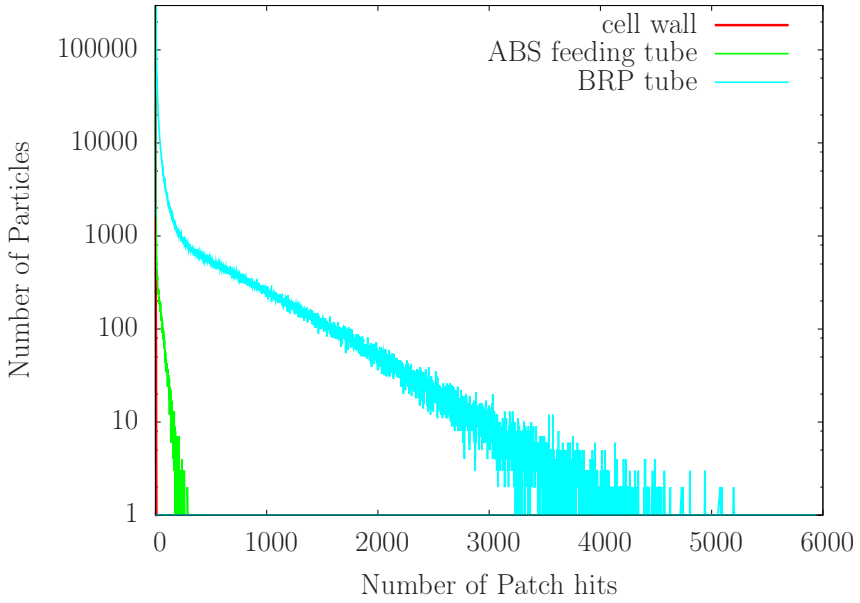

Figure 6.9 – *Collision age distribution, obtained with the* `FreeStream` *inflow model, for particles leaving the BRP tube divided into contributions of individual parts of the wall. The total distribution, which is not shown here, follows the one of the BRP tube very closely. Particles leaving through the BRP tube apparently have almost no collisions with the cell wall. Shown is the distribution of* $\approx 4.8 \cdot 10^6$ *particles.*

BRP tube and they can therefore collide only seldom with the wall, before they leave it through the BRP outlet. On the other hand, particles in the cell may enter from the ABS feeding tube with almost arbitrary angle (only particles with small angles to the z-axis may be reflected upwards and leave through the ABS feeding tube again) and will hit the cell wall frequently before they leave the tube. Something similar might happen with the `MaxwellianThermal/Freestream` settings where the distribution of the total number of wall hits at the BRP outlet seems to be composed of two exponentials. Most particles collide only a few times with the walls but another fraction undergoes thousands of wall collisions leading to a long tail of the distribution. The first part may come from particles which fly pretty straight down the ABS feeding tube and are then reflected into the BRP tube with a small angle to the y-axis, while the rest may be particles that had multiple wall collisions before they entered the BRP tube and therefore have all possible angles to the y-axis. When the particles have no initial downward stream velocity as in the case with the `MaxwellianThermal` model, the average angle to the z-axis is much higher and therefore the particles collide more often with the walls before entering the BRP tube, leading to a relative increase of the second kind of particles.

6.4 Test of Hyperfine Transition Units and Calculation of Polarization

High frequency transition units were tested in a flow of hydrogen atoms through a tube and spin exchange collisions were temporarily switched off. Several transition units were placed along the flow direction and the transition efficiencies were set such that all particles that cross the transition plane were put into a single hyperfine state although some of the transitions are physically impossible. In Fig. 6.10 it is shown which transitions have been used and the resulting nuclear polarization is plotted for a homogeneous background field of $0.1\,\mathrm{T}$. The electron polarization is not shown but is correct as well. The result in Fig. 6.10 was obtained with `SpecularReflection` as wall interaction model but a simulation with `MaxwellianThermal` was also done. However, this model has plenty of particles which are deflected in the backward direction after a wall collision and with the unrealistically set transition efficiencies, particles with the wrong hyperfine state move into a region where they would not be expected by considering only the flow direction. For example, at the second transition, where all particles are pushed into hyperfine state $|3\rangle$, particles crossing the transition plane from right to left (i.e. against the flow direction) will be in state $|3\rangle$ although state $|4\rangle$ is prevailing in this section. The result looks very strange with apparent distortions in the corner between the wall and the transition plane, although the result is correct. The effect is reduced to negligible levels with the `SpecularReflection` model because the number density at the exit of the tube and within the tube is low, leading to almost no particles moving into the wrong direction.

The simulation described above shows that the high frequency transitions and the polarization determination work perfectly when the transition plane crosses cells. However, it was also found in other test runs, that there are problems when the transition unit is aligned with cell faces. In this case, the transition probability of the transition unit is reduced significantly and the transition probability depends apparently on the position. This occurs at some transitions while it works perfectly at others. Another problem occurs when the simulation is executed on several processors. From time to time the program simply gets stuck in the calculation but no error message is produced. It simply runs forever without making progress. The problem may be resolved by starting the simulation again from a slightly earlier timestep, but this is not a real solution. The problem here is most likely caused by the shifts of the particles at cell faces as described in Sec. 5.1.6 and occurs when the cell face and the transition plane are not perfectly aligned within the numerical accuracy. If a particle is shifted at a cell face, it may be shifted across the high frequency transition unit without noting its presence and therefore pass unchanged. This explains the first part of the problem. That the second part of the problem occurs only in parallel runs indicates that some particles may be shifted forth and back between processors. Both problems would probably be solved if the high frequency transition units and the cell faces were perfectly aligned. This may be achieved with Gmsh as meshing tool since it allows to

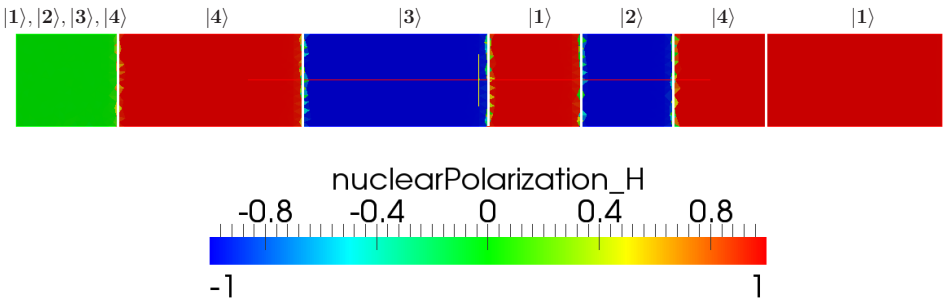

Figure 6.10 – *Simulation of a flow (left to right) of H atoms through a tube with several high frequency transition units marked by the vertical white lines. The units are idealized such that they put all particles into one hyperfine state of which the nuclear polarization is shown in the picture. The labels above the picture show which hyperfine state state is present in the region below. Some cells are transected by the transition units such that the cell average of the nuclear polarization is in between the ones of neighboring cells.*

specify the geometric tolerance. For the simulations here a tolerance of 10^{-12} m was used but OpenFOAM has a relative accuracy of about 10^{-15}. Therefore, a smaller tolerance in Gmsh might already solve the issue but this was not tested and the method might fail, since the points of the mesh may be shifted during mesh manipulations in OpenFOAM or because the cell faces and the transition units are calculated differently. Another possibility to solve the problem would be to restrict the high frequency transitions to normal cell face crossings, reducing flexibility. For now, the issue is not further pursued and the transition units are simply misaligned with the cell faces.

6.5 Test of Recombination at the Walls

The recombination at the walls was tested in a small cube with a side length of 2 cm that is initially filled with hydrogen atoms. Over time the particles collide with the wall, where they recombine according to the `SimpleLangmuirHinshelwood` model with a fixed recombination probability of $\gamma = 5 \cdot 10^{-3}$. The initial gas and wall temperatures were set to 300 K and as wall interaction model `MaxwellianThermal` was chosen. Therefore, the particle velocity is randomized after each wall collision, and on average all particles have the same velocity and travel the same distance between successive wall collisions. Then, one would expect that the number of recombinations per time depends only on the number of atoms, the rate ν_{wt} of wall collisions and the recombination probability, i.e.

$$\frac{dN_{\mathrm{H}}}{dt} = -N\nu_{\mathrm{wt}}\gamma \tag{6.2}$$

$$\rightarrow N(t) = N_0 e^{-\nu_{\mathrm{wt}}\gamma t} \tag{6.3}$$

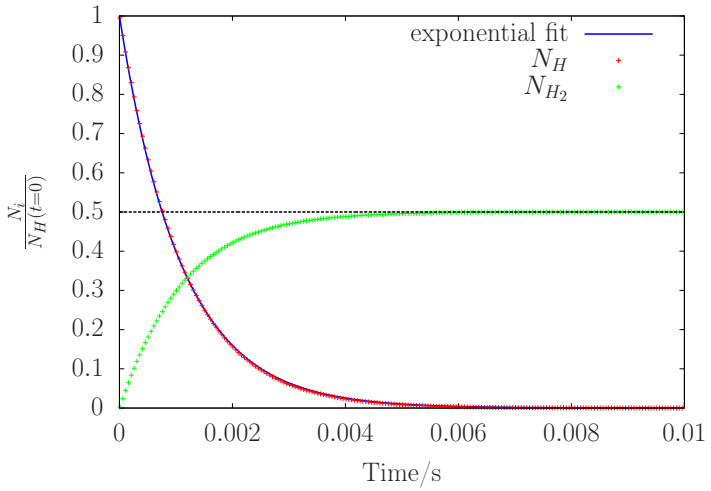

Figure 6.11 – *Normalized particle numbers during recombination in a cube of* 2 *cm side length. At the start of the simulation, the volume is filled with H atoms only which recombine during wall collisions with a probability of* $5 \cdot 10^{-3}$.

Fig. 6.11 shows the particle numbers normalized to the initial number of H atoms and a fit of Eq. (6.3) to the simulated number of atoms. The fit yields a wall collision rate $\nu_{\mathrm{wt}} = 183\,462.4\,\mathrm{s}^{-1}$. With an average thermal speed of the atoms of

$$\overline{v} = \frac{2}{\sqrt{\pi}}\frac{1}{\beta} = \frac{2}{\sqrt{\pi}}\sqrt{\frac{2k_{\mathrm{B}}T}{m}} = 2512.5\,\frac{\mathrm{m}}{\mathrm{s}} \tag{6.4}$$

one obtains an average flight distance between two wall collisions of $1.37\,\mathrm{cm}$ which is reasonable. This shows that the program correctly simulates the recombination.

It is also important to compare the particle numbers at the beginning and the end of the simulation because the proton number will not be perfectly conserved as described in Sec. 5.9. The relevant data can be obtained from the logfiles `log.dsmcSpinModInitialise` and `log.dsmcSpinModFoam`. The simulation starts with $159\,954$ simulated H atoms and ends with $80\,049$ H_2 molecules. One can make sure that no atoms are included in the final number by searching the last time of a wall collision of an H atom in the file `Parcels_at_patch_wall.dat`. This occurs after about $0.012\,\mathrm{s}$ and until the end of the simulation at $0.08\,\mathrm{s}$ no more atoms hit the wall. Therefore, one can conclude that the proton number increases by $0.18\,\%$ during the simulation. This error is small and is expected to scale with $1/\sqrt{N}$. This is negligible for every real simulation, where the number of simulated particles will usually be higher.

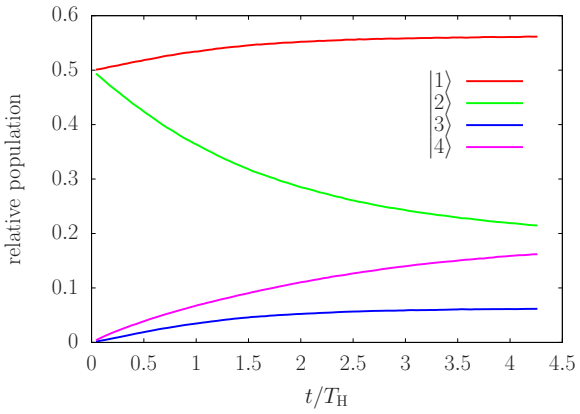

Figure 6.12 – *Evolution of hyperfine population numbers with spin exchange collisions for H in a homogeneous B field with* $B = 0.05\ T$.

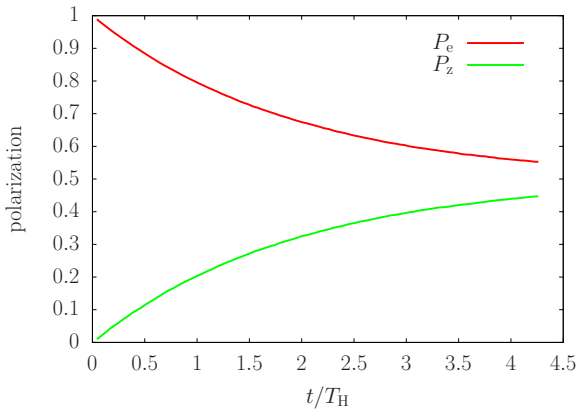

Figure 6.13 – *Evolution of polarization during spin exchange collisions for H in a homogeneous B field with* $B = 0.05\ T$.

6.6 Test of Spin Exchange Collisions

The tests of spin exchange collisions were performed in a small cube with edge length of 2 mm and a number density of $n = 10^{18}\,\mathrm{m}^{-3}$. Every simulated particle represents only 10^4 real particles, yielding roughly $8 \cdot 10^5$ simulated particles. Two simulations were run, one with hydrogen and one with deuterium atoms. In both cases the initial hyperfine population numbers were set such that the gas is fully electron spin polarized while it exhibits no nuclear or tensor polarization. Further, a small homogeneous magnetic field comparable to the critical field was applied. The computational volume was split into 11 288 cells containing ≈ 71 particles on average. In order to obtain a good time resolution, the result of the calculation was averaged for only 10^{-4} s before it was written to disk. Despite the relatively high particle number per cell, one still observes large fluctuations of the polarization between individual mesh cells for each time step. In order to smooth the data, the results shown in Figs. 6.12 to 6.15 are calculated as averages over the whole volume of the cube. In each case, the time is scaled to the mean collision time T_H respectively T_D between to atoms. This time is extracted from the file `log.dsmcSpinModFoam` by first extracting the numbers of collisions for every time step with the *foamLog* script and then averaging all values and finally dividing by the time step Δt_cD and the number of simulated particles.

One can check whether the spin temperature equilibrium values introduced in Sec. 2.3.2 are attained. For both cases it is $\langle m_\mathrm{F} \rangle = 1/2$ for the chosen initial values of the hyperfine populations. With this, one obtains $\eta = 1/2$ for hydrogen and $\eta \approx 0.278$ for deuterium. The expected values are obtained with the formulas given in Tab. 2.4 and one obtains the values in Tab. 6.4. The simulations clearly match the expectations. Another point is the relaxation rate that can be approximated by Eq. (2.40). For hydrogen one expects that $n_2 - n_4 \approx 0.18$ at $t = 2T_\mathrm{H}$ which is fulfilled. The shape of the individual curves can also be calculated numerically from Eq. (2.39) or, more conve-

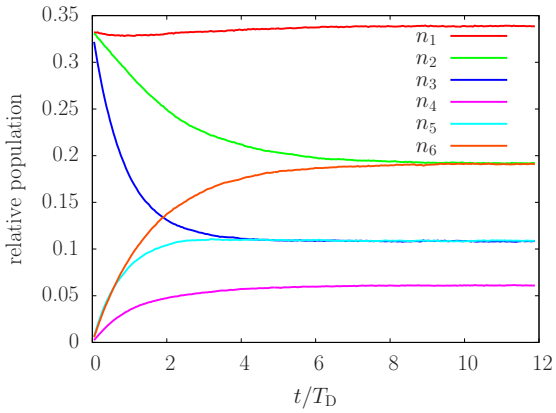

Figure 6.14 – *Evolution of hyperfine population numbers with spin exchange collisions for D in a homogeneous B field with* $B = 0.01\ T$.

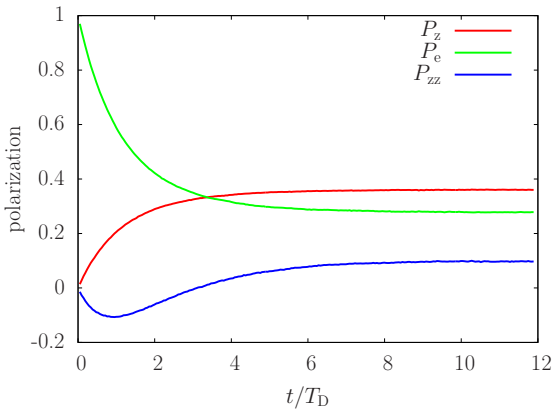

Figure 6.15 – *Evolution of polarization during spin exchange collisions for D in a homogeneous B field with* $B = 0.01\ T$.

Table 6.4 – *Expected population numbers and polarization of hydrogen and deuterium in spin temperature equilibrium for the simulated conditions where in both cases* $\langle m_F \rangle = 1/2$.

Value	Hydrogen	Deuterium
n_1	0.56	0.34
n_2	0.19	$n_2 = n_6 = 0.19$
n_3	0.06	$n_3 = n_5 = 0.11$
n_4	0.19	0.06
P_e	0.5	0.28
P_z	0.5	0.36
P_{zz}		0.10

niently, from the explicit equations given in [44]. Here, they were just compared with the corresponding plots for different B field in [38] which shows perfect agreement. All this shows that the spin exchange collisions work as expected.

6.7 Simulations of a Complete ABS (in Two Steps) and Test of a New Inflow Model

Several runs of a complete ABS were executed at the end of the thesis. The ABS was split into two parts for the calculation and both parts were calculated individually. The first part comprises the nozzle, the first two vacuum chambers and the bore of magnet 1, compare Fig. 6.16. The second part also comprises the bore of magnet one which is meant as an overlap region for defining the inflow parameters for the second part, which further comprises vacuum chambers 3 to 5. The first part of the ABS is the same

for all simulations and calculated only once. This provides a huge speedup, because the high number densities in and around the nozzle require short simulated time steps, leading to an execution time of roughly two months. The second part can be handled with much bigger time steps such that it can be calculated within one or two days.

The setup for the first part is as follows. At the inflow a number density of $n_{\mathrm{H}} = 3.5 \cdot 10^{22}\,\mathrm{m}^{-3}$ is specified for hydrogen atoms and $n_{\mathrm{H}_2} = 0$ for molecules. The gas temperature at the inflow is set to $T = 1200\,\mathrm{K}$ and the nozzle temperature is $T = 100\,\mathrm{K}$. The probability for recombination during a wall collision is set to $\gamma = 4.5 \cdot 10^{-4}$ in the nozzle and to $\gamma = 8 \cdot 10^{-2}$ [87] at all other walls, which are kept at a constant temperature of $T = 300\,\mathrm{K}$. The resulting degree of dissociation behind the nozzle is $\alpha \approx 0.7$, which is in good agreement with measurements. The number densities at the outlets of chamber 1 and 2 are set to $n_{\mathrm{H},1} = 1.52 \cdot 10^{19}\,\mathrm{m}^{-3}$ and $n_{\mathrm{H},2} = 2.4 \cdot 10^{18}\,\mathrm{m}^{-3}$, respectively. The corresponding densities for the molecules are set to zero. The beam that is formed in this first part is flowing into a vacuum at the exit of magnet 1. The vacuum ($n_{\mathrm{H}} = 10\,\mathrm{m}^{-3}$, $n_{\mathrm{H}_2} = 0$) is chosen in order to not get reflected particles which would spoil the results for the second part of the calculation. Magnet 1 is represented only physically but does not produce a magnetic field in the first part of the simulation.

The second part of the simulation is always executed with $n_{\mathrm{H}} = 0$ at all three outlets while the number densities for molecules are set to $n_{\mathrm{H}_2,3} = n_{\mathrm{H}_2,5} = 7.25 \cdot 10^{16}\,\mathrm{m}^{-3}$ and $n_{\mathrm{H}_2,4} = 4.83 \cdot 10^{16}\,\mathrm{m}^{-3}$ respectively. This is to reflect that the background is mainly composed of molecules. The number densities at the outlets are calculated from measured pressures in the various chambers and the pressure in the fifth chamber is usually higher than the pressure in the fourth chamber, because the beam is dumped into it. The conditions at the entrance of magnet 1 are calculated with the new inflow model *MixedInflow* described in Sec. 5.5 with the option `cellInternalValues` set to true. The fields for the hyperfine population numbers were set manually to the values for unpolarized gas because time averaging for these fields was accidentally switched off during the first part of the simulation. The magnets are present in all simulations, either producing a magnetic field or not. The MFT is also always present and interchanges the population numbers of hyperfine states $|2\rangle$ and $|3\rangle$ with $100\,\%$ efficiency if switched on.

The results of both parts of the simulation are mapped onto a mesh of the complete ABS. The result of the first part is always mapped first such that the result of the second part overwrites the field values within the bore of magnet 1. An example can be seen in Fig. 6.16 which shows the number densities $n_{|1\rangle}$ for particles in hyperfine state $|1\rangle$ and $n_{|4\rangle}$ for particles in hyperfine state $|4\rangle$ for a simulation with all magnets switched on while the MFT is switched off. One can clearly see that particles in state $|1\rangle$ are focused while particles in state $|4\rangle$ are defocused. One can also see that the number density for state $|1\rangle$ in the compression tube is slightly lower than in chamber 5 which does not look correct. This cannot be only due to recombination as can be seen in Fig. 6.17 which shows the pressure in the ABS along an axis parallel to the x-axis (with $1\,\mathrm{mm}$

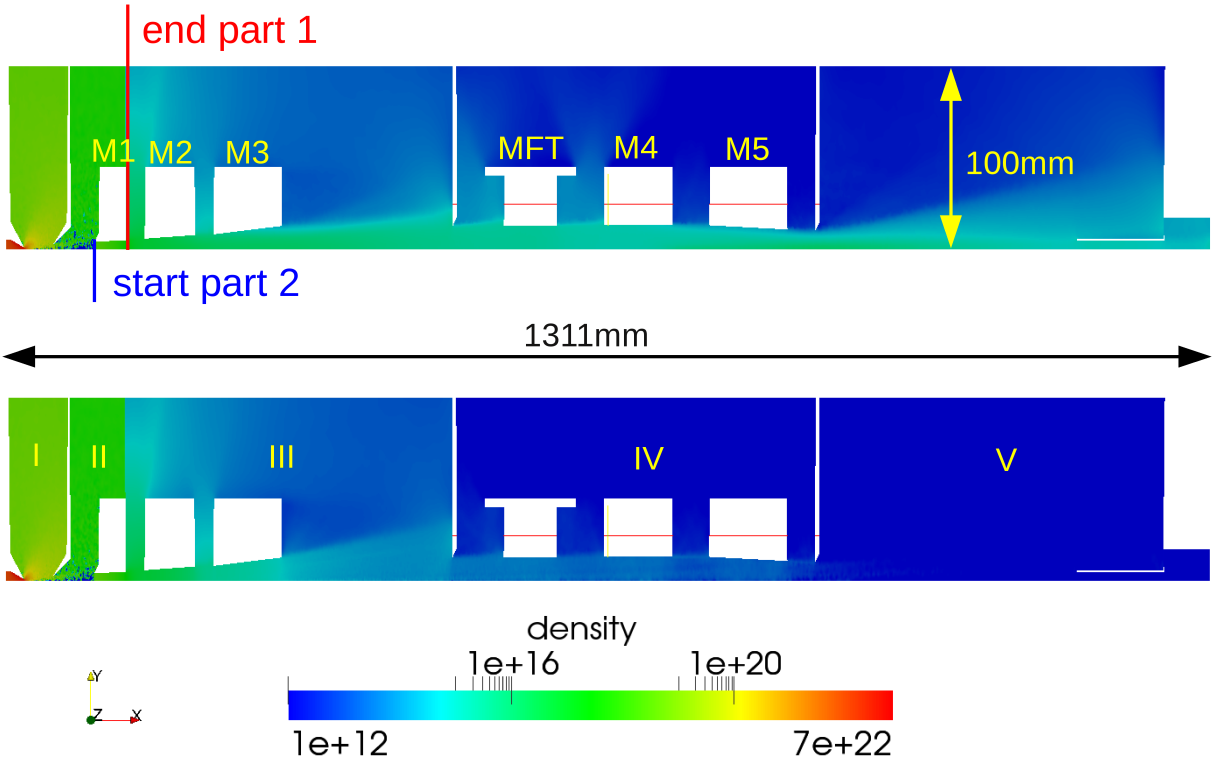

Figure 6.16 – *Number density in m^{-3} for particles in hyperfine state $|1\rangle$ (top) and $|4\rangle$ (bottom) in the ABS with all sextupole magnets (M1-M5) producing a B field while the MFT is switched off. One can clearly see that particles in state $|1\rangle$ are focused while particles in state $|4\rangle$ are defocused. The speckles between the skimmer and magnet 1 result from time averaging of the fields for the hyperfine population numbers switched off. The gas was manually made unpolarized for the second part. The overlap region for the two different parts of the simulation is shown at the top. Note that there is a thin wall between vacuum chambers 2 and 3 stretching outwards from magnet 1 along the red line which cannot be seen in the picture. The roman numbers in the bottom figure denote the vacuum chamber. The y-axis is stretched by a factor of 2.*

offset) for a run with and one without B field of the magnets. The pressure decreases along the compression tube although the fraction of atoms in the gas is very low, as shown in Fig. 6.18. Measurements also show that a higher pressure than in chamber 5 can be expected. The problem is probably the boundary condition for the wall. It was set to *calculated* for the number densities while *zeroGradient* would have been more appropriate. The *zeroGradient* condition was used for the simulation in Sec. 6.1 which showed a significant pressure increase in the compression tube. The false boundary condition is applied at all walls and the simulations have to be rerun to get correct results.

The hyperfine population numbers and the polarizations are calculated for a run with (Figs. 6.19 and 6.20) and one without (Figs. 6.21 and 6.22) MFT. The magnets are on

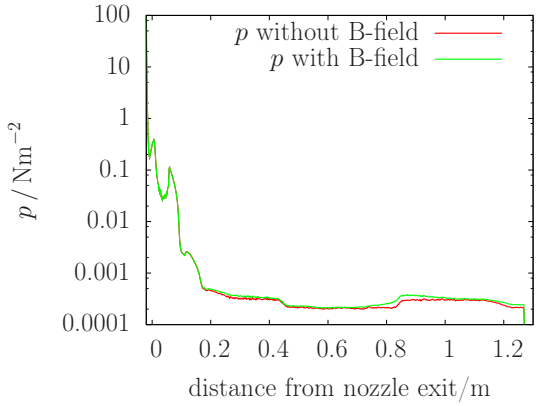

Figure 6.17 – *Pressure along an axis parallel to the beam axis with* 1 *mm offset. The green curve shows the case when the magnets produce a B field and the red curve shows a case without B field. The difference of both curves is small because the H_2 background dominates as can be seen in Fig. 6.18.*

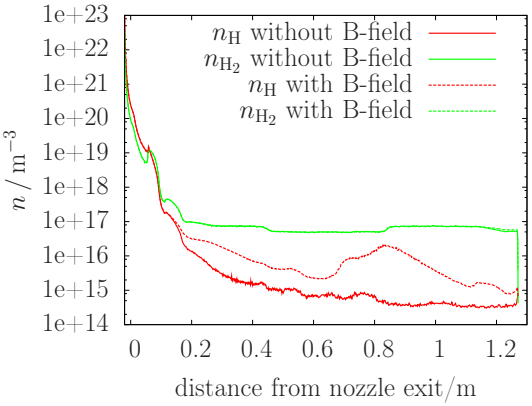

Figure 6.18 – *Number densities of H and H_2 along an axis parallel to the beam axis with* 1 *mm offset with and without B field. The density of H atoms increases significantly when the magnets produce a B field while the H_2 density does not change.*

in both cases. The result of the simulation without MFT matches the expectation. The result of the simulation with MFT, however, reveals some problems of the program. The first thing which cannot be seen in the figures is that the MFT switches particles from state $|2\rangle$ to $|3\rangle$ not only within the bore of the yoke but also beyond the yoke. Apparently there is a minor error in the program, probably in the function `DsmcCloud::hfsTransitionProperties::insideHfRegion()` which was not immediately found. The more serious error is that the MFT switches less than half of the particles from state $|2\rangle$ to state $|3\rangle$. As a first guess it was assumed that the transition was accidentally aligned with cell faces which could have led to particles leaking through the MFT, as described in Sec. 6.4. In order to check this, two more simulations were performed, each with the transition plane of the MFT placed an additional millimeter behind its previous position. The results look much the same than the ones shown in Fig. 6.21. This indicates that there is another problem, which did not appear during the simulations for Sec. 6.4. Currently the cause is not clear, but it will be investigated. Another curiosity is the relative increase of particles in state $|2\rangle$ along the beam axis behind the MFT when the MFT is on.

The simulation was also meant as a test of the new inflow model. This seems to work fine, but more detailed studies are necessary to definitely decide whether it really works correctly. There are several assumptions in the model described in Sec. 5.5 which are not yet shown to be correct. In order to get more confidence in the simulation results it will be crucial to study this further.

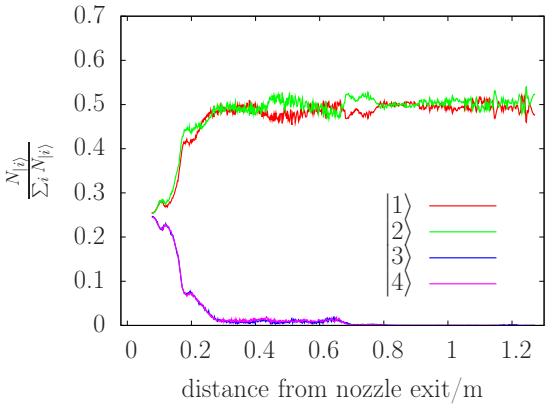

Figure 6.19 – *Hyperfine population numbers at* 1 *mm offset parallel to the beam axis in the ABS with MFT switched off. Hyperfine states* $|1\rangle$ *and* $|2\rangle$ *become focused, while states* $|3\rangle$ *and* $|4\rangle$ *become defocused. The results of the first part of the simulation are not shown.*

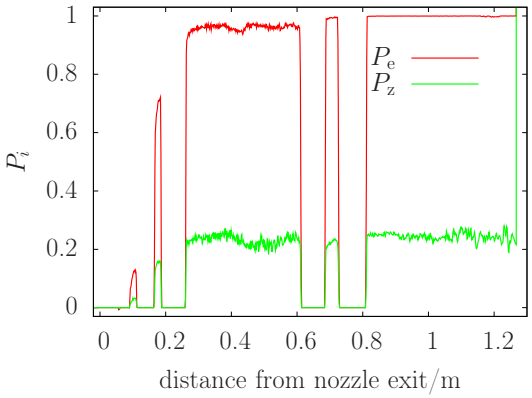

Figure 6.20 – *Electron and nuclear polarization at* 1 *mm offset parallel to the beam axis in the ABS with MFT switched off. The polarization is not calculated in inhomogeneous B fields, i.e. within the magnets, where it is set to zero. The homogeneous background field is* 1 mT.

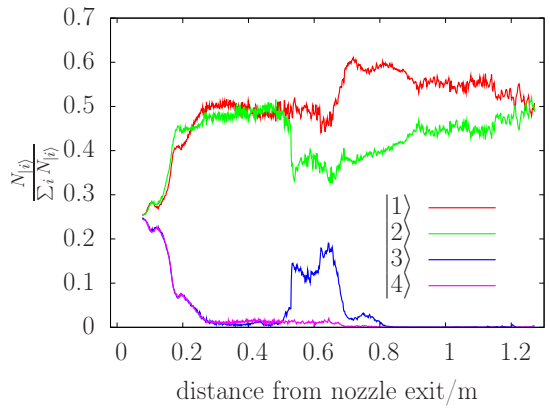

Figure 6.21 – *Hyperfine population numbers at* 1 *mm offset parallel to the beam axis in the ABS with MFT switched on. Hyperfine states* $|1\rangle$ *and* $|2\rangle$ *become focused, while states* $|3\rangle$ *and* $|4\rangle$ *become defocused. The MFT flips less than half of the electron spins of particles in state* $|2\rangle$*. The results of the first part of the simulation are not shown.*

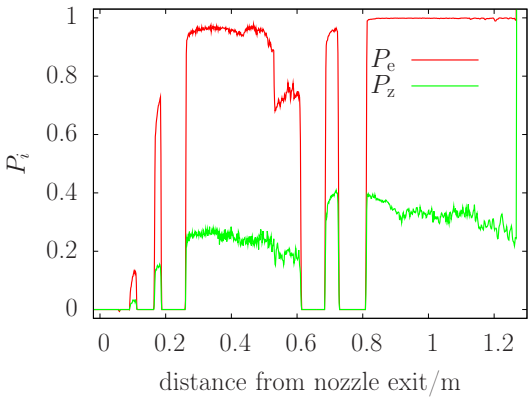

Figure 6.22 – *Electron and nuclear polarization at* 1 *mm offset parallel to the beam axis in the ABS with MFT switched on. The polarization is not calculated in inhomogeneous B fields, i.e. within the magnets, where it is set to zero. The homogeneous background field is* 1 mT.

7 Optimization

Optimization is the process of finding the best element in a set of possibilities with regard to a set of criteria. These criteria can generally be combined to a cost function

$$f : P \rightarrow \mathbb{R} \tag{7.1}$$
$$f : \vec{x} \mapsto f(\vec{x}) \tag{7.2}$$

that maps the element $\vec{x}$ from the parameter space $P \subseteq \mathbb{R}^n$ to a real number. Then, optimization means finding a local or global extremum of f. Without loss of generality, one can restrict oneself to the search for a minimum, since a maximization of a function f is equivalent to the minimization of $-f$. Optimization is a very wide field of mathematics and there exist many different algorithms to find an optimum. Which one to choose depends on the known properties of the cost function like information about first or higher order derivatives or whether the function is continuous, linear, and/or convex. The more knowledge exists, the more specialized an algorithm can be selected and a better convergence behaviour is expected. However, if one does not use any information about the cost function, the No Free Lunch (NFL) theorems [88] state, that, averaged over all possible cost functions, all algorithms perform equally well. That means, if one algorithm performs better than another algorithm on a certain problem, it will necessarily perform worse on some other, even if the second algorithm is a simple random search. So there is no a priori knowledge of the performance of a special algorithm. The situation is even worse: The NFL theorems also state that testing two algorithms with a certain cost function for some time and comparing the results, one cannot find out which algorithm performs better in a longer optimization run with the same cost function. Therefore, effective simulation relies on fortuitous matching between the cost function and the algorithm. Another important consequence of the NFL theorems is, that comparing the performance of two algorithms on one problem does not tell anything about the performance of the same algorithms on other problems.

For the optimization of the atomic beam source the situation is pretty dire. There is almost no information about the cost function. One can probe it only at a certain point of the parameter space with a DSMC calculation. A gradient can only be calculated numerically, which is computationally expensive in this case, especially for higher dimensional parameter spaces. One may suspect that f is continuous, but it may exhibit large gradients and f is certainly nonlinear. One does not know if f is convex and

one cannot tell how many local and global extrema it has. The situation is further complicated by the fact that f is calculated with a stochastic method and hence the calculated function values scatter around the real value. Nonetheless, one wants to have a method that robustly finds the global optimum in a short time. These are very stringent requirements on the algorithm and not many algorithms can deal with such general cases. One possibility are evolutionary algorithms [89] and a genetic algorithm was already used to optimize the sextupole system of the HERMES-ABS (located at DESY in Hamburg, Germany) with a ray-tracing particle tracking algorithm [90]. However, genetic algorithms have major drawbacks. For one, it is difficult to set suitable parameters like the mutation rate and on the other side, the algorithm can get stuck in local extrema or even converge to arbitrary points.

In [25] a gradient scheme was used to optimize the magnet system of the RHIC-ABS (Brookhaven, USA). It was found that the algorithm fails because of two reasons. First the statistical fluctuations of the ray-tracing algorithm sometimes lead to reversed gradients when the gradients were small, and this made the algorithm move into the wrong direction. The second problem was the somewhat unexpected occurrence of local extrema, where the algorithm got stuck and the final result of the optimization therefore depended on the initial parameters. Finally, the group used a brute force method and calculated the cost for several ten thousands of parameter combinations and manually selected the best one.

For the DSMC calculation, a brute force method is out of question because of its high computation time. Anyway, it would scale bad with the dimensionality of the parameter space. A gradient scheme is also prohibitively expensive and would probably fail because of the same reasons as above. The genetic algorithm does not seem to be suitable, since it is not guaranteed to converge to an optimum. Actually, the only black box algorithm that can come up with a convergence proof is Adaptive Simulated Annealing (ASA), which will be used in this work and is described in the next section.

7.1 Adaptive Simulated Annealing (ASA)

Like many other optimization schemes, Adaptive Simulated Annealing [91] (formerly called Very Fast Simulated Re-Annealing [92]) is inspired by a natural process, which is the annealing of a metal after hardening in this case.

The optimization method exploits some analogy with a typical physical problem, namely the minimization of the energy of a system. The cost function of a problem is scalar valued and it can therefore be viewed as a potential landscape. In this picture, every evaluation of the cost function returns the value of the potential energy at this point in parameter space. If the k th evaluation of f for parameter set $\vec{x}_k$ yields

$f(\vec{x_k}) = E_k$ and in the next step one obtains $f(\vec{x}_{k+1}) = E_{k+1}$, the new position in parameter space is accepted with a probability that depends only on the two most recent function evaluations:

$$h(\Delta E) = \frac{1}{1 + e^{\frac{\Delta E}{T_\mathrm{a}}}} \approx e^{-\frac{\Delta E}{T_\mathrm{a}}} \tag{7.3}$$

where $\Delta E = E_{k+1} - E_k$ is the energy difference and T_a is a control parameter usually called acceptance temperature. This temperature is time-dependent and lowered according to

$$T_\mathrm{a}(k) = T_{\mathrm{a},0} e^{-c_\mathrm{a} k^{\frac{1}{D}}} \tag{7.4}$$

where D is the number of dimensions of the parameter space and $c_\mathrm{a} > 0$ is a control parameter that can be chosen freely to tune the algorithm for a specific problem. The coordinate i of a new point $\vec{x}_{k+1} \in [A_i, B_i]$ is calculated according to

$$x^i_{k+1} = x^i_k + y^i (B_i - A_i), \tag{7.5}$$

using only the current point as starting point, where $y_i \in [-1, 1]$ is a random number that is selected according to the distribution function

$$g^i_\mathrm{T}\left(y^i\right) = \frac{1}{2\left(|y^i| + T_{\mathrm{g},i}\right) \ln\left(1 + \frac{1}{T_{\mathrm{g},i}}\right)}, \tag{7.6}$$

which is shown in Fig. 7.1 for different values of the generation temperature T_g and which has a cumulative distribution function

$$G^i_\mathrm{T}\left(y^i\right) = \frac{1}{2} + \frac{\mathrm{sgn}\left(y^i\right)}{2} \frac{\ln\left(1 + \frac{|y^i|}{T_{\mathrm{g},i}}\right)}{\ln\left(1 + \frac{1}{T_{\mathrm{g},i}}\right)}. \tag{7.7}$$

Hence, a suitable y^i can be generated from a uniformly distributed random number $R^i_\mathrm{f} \in [0, 1]$ by

$$y^i = \mathrm{sgn}\left(R^i_\mathrm{f} - \frac{1}{2}\right) T_{\mathrm{g},i} \left(\left(1 + \frac{1}{T_{\mathrm{g},i}}\right)^{\left|2R^i_\mathrm{f} - 1\right|} - 1\right). \tag{7.8}$$

With this y^i, a new coordinate x^i_{k+1} of a new point is calculated according to Eq. (7.5) until it fulfills the boundary condition, i.e., until $x^i_{k+1} \in [A_i, B_i]$. To obtain a new point $\vec{x}_{k+1}$, one can move parallel to an axis of the parameter space by changing only one coordinate of $\vec{x}_k$ or one can change all coordinates at once. Using different generation temperatures $T_{\mathrm{g},i}$ for every coordinate allows to handle different sensitivities of the cost

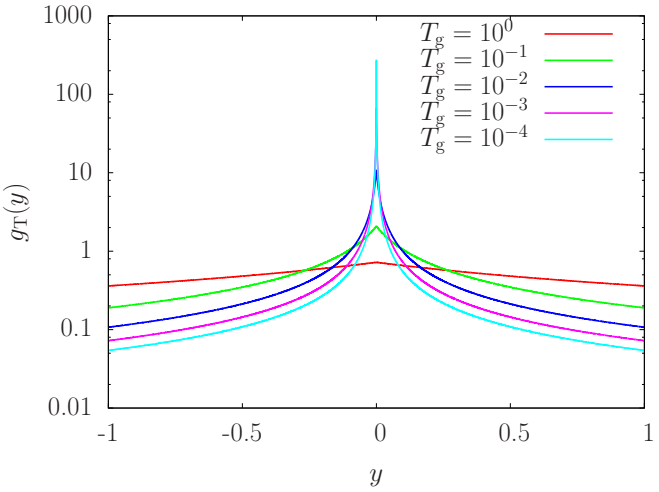

Figure 7.1 – *Probability density function $g_T(y)$ for generating new points in ASA for different values of T_g; The distribution is increasingly centered around 0 for decreasing T_g with non-vanishing probability for larger $|y|$.*

function to the various parameters. As for the acceptance temperature, the generation temperature is decreased over time, and using the distribution function in Eq. (7.6) for the algorithm allows one to decrease $T_{g,i}$ exponentially fast:

$$T_{g,i}(k) = T_{g,0,i}\, e^{-c_{g,i}\, k^{\frac{1}{D}}} \tag{7.9}$$

where again the $c_{g,i} > 0$ are free parameters to tune the algorithm. It is possible to use the number of generated states k for both, the cooling of the generation temperature and the acceptance temperature, but it has proven fruitful to use the number of accepted states for k for the acceptance temperature instead. This cooling schedule then makes sure that the distribution gets centered more and more around zero and therefore, the average stepsize in the parameter space decreases significantly with time. While at the beginning of the optimization process the whole space is sampled, the search narrows to a neighborhood of the last generated point after some time, while there is still a small probability to make a big step, which may allow to escape a local minimum.

The convergence of the method is easy to show. It is enough to proof that every point of the parameter space can be sampled infinitely often in an infinitely long run, i.e. that the probability of not generating a state $\vec{x}$ for all times goes to zero, which is

$$\lim_{k\to\infty} \prod_k (1 - g_k) = 0, \tag{7.10}$$

or which is equivalent to

$$\lim_{k\to\infty} \sum_k g_k = \infty \tag{7.11}$$

after taking the logarithm, where

$$g_k = \prod_{i=1}^{D} g_{\mathrm{T}}^i \left(y^i\right) = \prod_{i=1}^{D} \frac{1}{2\left(|y^i| + T_{\mathrm{g},i}\right) \ln\left(1 + \frac{1}{T_{\mathrm{g},i}}\right)}. \tag{7.12}$$

For large enough k it is $\ln\left(1 + \frac{1}{T_{\mathrm{g},i}}\right) \approx c_{\mathrm{g},i}\, k^{\frac{1}{D}}$ and $T_{\mathrm{g},i} \approx 0$. So, starting with a sufficiently large k_0, one obtains

$$\sum_{k=k_0}^{\infty} g_k \approx \sum_{k=k_0}^{\infty} \left(\prod_{i=1}^{D} \frac{1}{2\,|y^i|\, c_{\mathrm{g},i}} \right) \frac{1}{k} = \infty, \tag{7.13}$$

which proofs that the best solution will be found, although it might take infinitely long. One reason for a low convergence rate may be that the cost function depends only weakly on a certain parameter and the generation temperature is too low, such that the algorithm moves only slowly towards a better parameter set. If the temperature would be higher, bigger steps would be more common, and one would expect faster convergence. The opposite is the case where the cost function is very sensitive to a certain parameter, such that large steps would mean overshooting the minimum. In order to handle different sensitivities of the cost function, there is the technique of reannealing, which can be performed in certain intervals and means increasing the temperature for certain parameters according to the sensitivity $s_i = \frac{\partial f}{\partial x_i}$ of the cost function to them at the most current minimum of the cost function. If the largest s_i is called $s_{\max}$ and the generation temperature is calculated according to Eq. (7.9), one can either rescale k or alternatively $T_{\mathrm{g},0,i}$ according to

$$T'_{\mathrm{g},0,i} = \frac{s_{\max}}{s_i}\, T_{\mathrm{g},k,i}\, e^{c_{\mathrm{g},i}\, k^{\frac{1}{D}}}. \tag{7.14}$$

Reannealing may increase convergence speed, but from Eq. (7.9) one can see, that with increasing dimensionality of the problem, the temperature can be lowered slower and for a high dimensional problem it may become too slow to ever get a reasonable result. A way to increase the rate of cooling is to introduce a quenching factor Q_i of the order of D to press the temperature lower and sooner end up in a minimum such that Eq. (7.9) becomes

$$T_{\mathrm{g},i}(k) = T_{\mathrm{g},0,i}\, e^{-c_{\mathrm{g},i}\, k^{\frac{Q_i}{D}}}. \tag{7.15}$$

However, for $Q_i > 1$ the above convergence proof fails since $\sum_{k=k_0}^{\infty} \frac{1}{k^{Q_i}}$ is finite.

7.2 Solver *optimizationFoam*

The optimization of an atomic beam source requires many DSMC calculations for various parameter settings and geometries. An optimization algorithm should set these parameters and geometries automatically, and the solver *dsmcSpinModFoam* should only be used to calculate the cost function, which is used by the optimization algorithm. This makes it necessary that the optimization algorithm runs on a higher hierarchy level than *dsmcSpinModFoam*, and the folder structure for an optimization run on disk should reflect this. Further, one may want a versatile optimization tool, that can be used with different optimization algorithms and for different optimization problems, where the cost function may be calculated with different solvers. The new solver *optimizationFoam* is a framework that can fulfill all these requirements. It consists of two main classes, which both make use of OpenFOAMs runtime selection mechanism, *OptimizationAlgorithm* and *OptimizationProblem*. This makes it possible to specify in a new configuration file `optimizationOptions` (compare appendix E) which problem should be solved by which algorithm, and makes it particularly easy to include new algorithms and problems.

For the implementation, this means that a layered approach with an inheritance structure and use of templates is required. The UML diagram of the base class for the optimization algorithm is depicted in Fig. 7.2. The class serves as base class for the use with the runtime selection mechanism and as parent for all optimization algorithms, where currently only Adaptive Simulated Annealing is implemented. Further, the base class declares some general functions, which every optimization algorithm has to implement. The function *parameterIdList()* returns a list with the names of the parameters to optimize, *parameters()* returns their value, and *generatedStates()* returns the number of generated states. The other three functions are self-explanatory.

An UML diagram of ASA, the only implemented optimization algorithm, is shown in Fig. 7.3. The class implements the algorithm described in Sec. 7.1 and from there, the meaning of most class attributes should become clear. A lot of them can be set in the configuration file `optimizationOptions`. Two so far unexplained options are *minimize* and *restart*. The first is set to true by default, but if changed to false, the algorithm will try to find the maximum of the cost function by minimizing the negative cost function. If *restart* is set to true, ASA will restart the optimization from the last completed state with appropriate settings, which requires that the algorithm keeps track of those and saves them in a file `optimization/AsaState` in order to be able to reload them after a shutdown of the program. The reload or new initialization is done by the function *initialize()*. The function *optimize()* generates new states, tests if they are within the parameter space by calling the function *OptimizationProblem::viableState()*, and calculates the cost function by calling the appropriate function from the optimization problem and negating the result if necessary. Afterwards, the acceptance test according to Eq. (7.3) is performed and the acceptance and generation temperatures are lowered. Finally, the

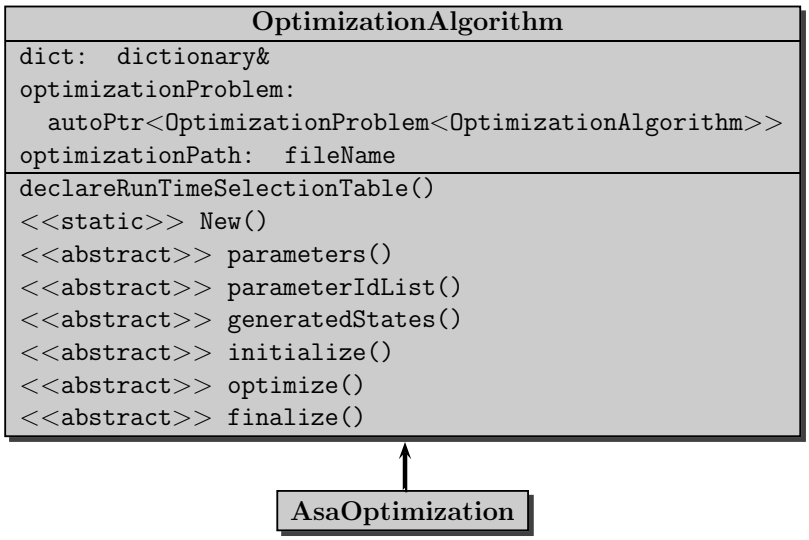

Figure 7.2 – *UML diagram of the* OptimizationAlgorithm *base class in solver* optimizationFoam

number *reannealStep* indicates after how many accepted states the reannealing is performed. This is done according to Eq. (7.14) where the partial derivatives are calculated by forward or backward difference quotient according to

$$\frac{\partial f}{\partial x_i} = \frac{f(x_i \pm \Delta x_i) - f(x_i)}{\pm \Delta x_i}, \tag{7.16}$$

where the Δx_i correspond to the parameters *deltaParameters* in the configuration file. The value of the Δx_i is limited to $10^{-3}(x_{i,\max} - x_{i,\min})$ in the program and the appropriate difference quotient is chosen when x_i is close to a boundary of the parameter space. The last function *finalize()* simply prints some final values containing some information about the best parameters found together with the minimal cost.

The base class of the optimization problem, *OptimizationProblem*, has only two abstract functions. First, *viableState()* indicates whether a generated state lies within the parameter space of the problem and second, the function *costfunction()* calculates the function value at a certain position in parameter space. This is enough for simple optimization problems like minimizing arbitrary functions as indicated by the two problem classes *testFunction0* and *testFunction2* in Fig. 7.4. The problem is much more complex for the case of optimizing an atomic beam source, since a new test case folder has to be set up for every function evaluation and the entries in the configuration files have to be adapted. Further, the mesh may have to be changed and post-processing of the DSMC result may be necessary to extract the function value. Many tasks are similar for every situation and therefore a class *DsmcOptimizationProblem* was created. The actual problem classes are then derived from this parent class as can be seen in Fig. 7.4.

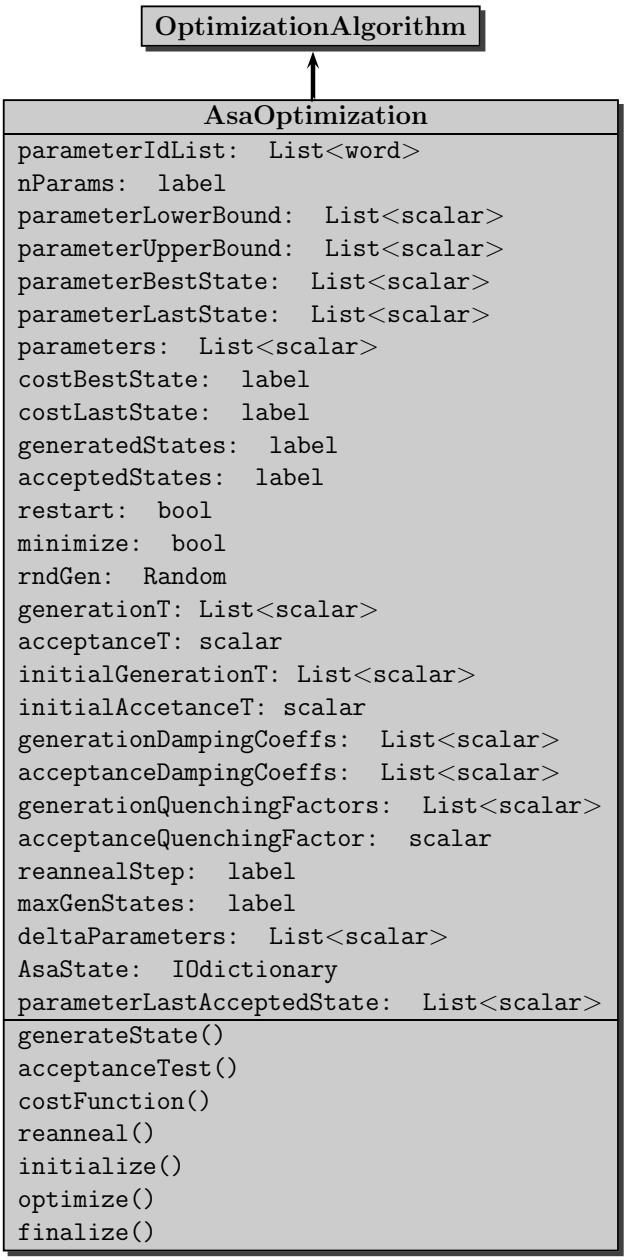

Figure 7.3 – *UML diagram of the optimization algorithm class for ASA*

The design here is such that the parent class does the standard work and the child class performs the special actions or handles exceptions from the standard. This leads to a somewhat interwoven structure and may best be explained with an example. For this, a test case was created, which is called *AsaDsmcTest_1*. The task here is to minimize the flow through a small tube of circular cross section and of fixed length, that is bent by an angle $\varphi \in [10^{-6}, 3]$ to form a ring segment, see Fig. 7.5. Further, the pressure or number

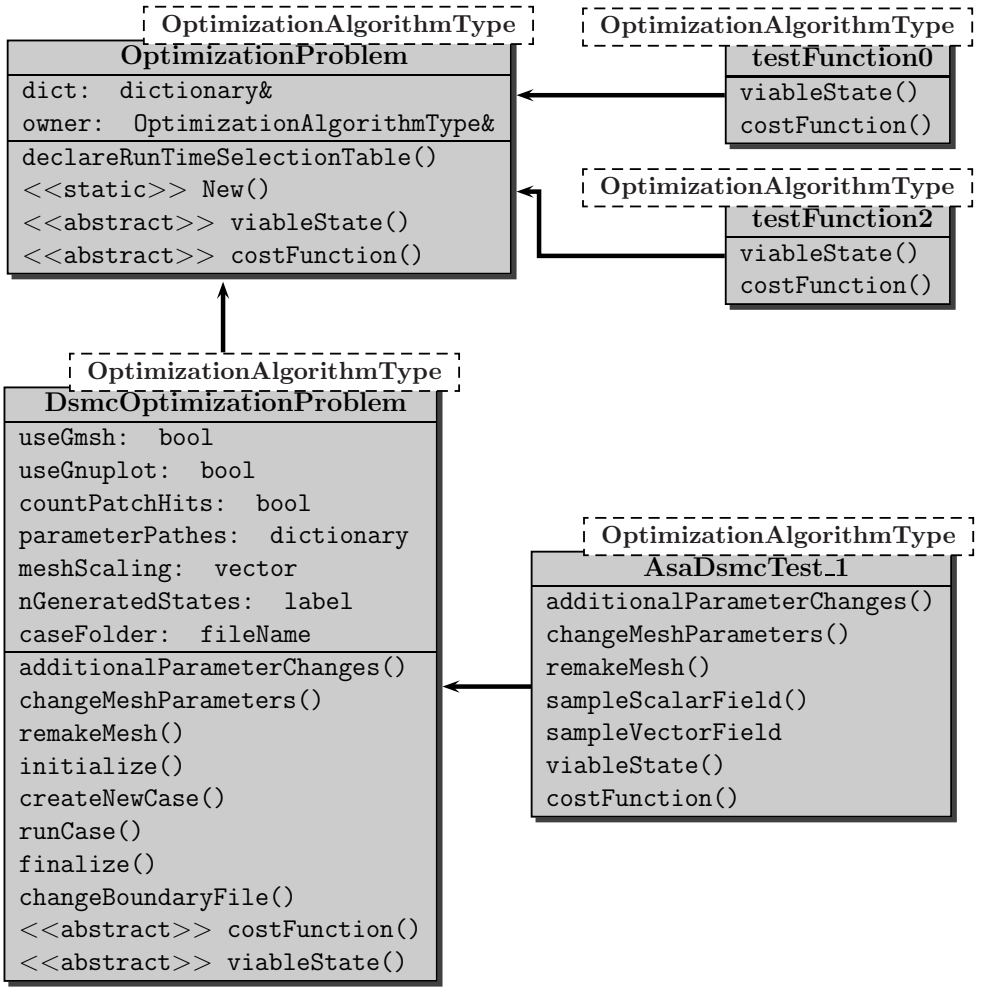

Figure 7.4 – *UML diagram of the optimization problem classes*

density n at the entrance of the tube can vary by a factor 5 in the range $n \in [10^{23}, 5 \cdot 10^{23}]$. One may expect to see the minimal flow through the tube at the lowest number density and the biggest curvature of the tube for a molecular flow. The curvature of the tube is expected to be negligible for a laminar flow. This simple test case is ideally suited to test the overall functionality of the class *DsmcOptimizationProblem*, since changing φ means changing the geometry of the problem and changing n needs several changes in configuration files.

Before the start of an optimization, the function *initialize()* will read some parameters from the configuration file. For example, if *useGmsh* is set to true, the freely available third party software `Gmsh` [93] will be used for meshing and if *useGnuplot* is set to true, the commonly used plotting tool `Gnuplot` [94] will be used for plotting the information contained in the logfile of a run. Additionally, the option *countPatchHits* will use the cloud class *dsmcPatchCloud* as described in Sec. 5.4 to let the particles count the number

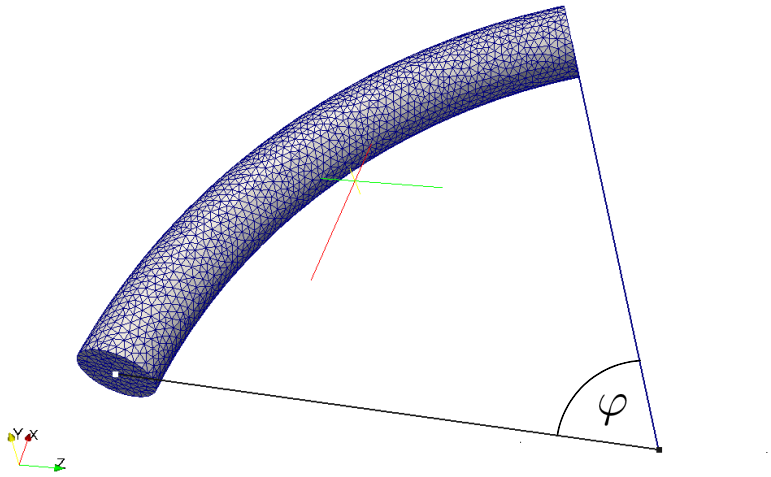

Figure 7.5 – *Geometry for a test case during the optimization of the problem* AsaDsmcTest_1

of wall collisions and the option *meshScaling* will allow to scale the mesh during import into OpenFOAM from other mesh formats.

The work for calculating one function value can be divided into three major steps: First, setting up the test case, then doing the DSMC calculation and finally extracting the flow through the tube from the DSMC result. The first task is executed by the function *createNewCase()*. This function will start with copying the relevant files from the folder of the last test case named `TestState_i` to a new folder with the new number of generated states in its name, i.e. `TestState_i+1`. In order for some of the file handling operations to succeed, one has to remove a barrier in OpenFOAM that limits the allowed file structure to that one known from a single DSMC case and prohibits a further hierarchy level. This limitation may be useful in many cases, but here it has to be removed by a change to the file `regIOobjectWrite.C` from the common infrastructure of OpenFOAM. It is simply extended in such a way that if the option *allowWritingInArbitraryPlace* is set to true in the top level *controlDict* (compare Fig. B.1), writing is allowed in arbitrary places and otherwise the barrier is kept. Then, after copying the folders, the configuration files have to be changed. In order to find the correct places in the correct files where the changes have to be made, the full path to the parameters has to be provided in the dictionary *DsmcProblemOptions/parameterPathes* in the file `optimizationOptions`, because some parameter names may occur multiple times in a file. The first word in the list is either *mesh*, if the geometry has to be changed, or the file name of the configuration file followed by the names of the various *dictionary* hierarchy levels, compare appendix E. If the parameter does not need a remake of the geometry, the parameter value in the appropriate configuration file is changed to the one

required by the optimization algorithm and the new file version is saved. If the geometry has to be changed during the optimization, one needs tools to automatically change the geometry and remake the mesh. In principle, this could be done with OpenFOAMs *blockMesh* utility, but it is quite difficult to use for complex geometries. Therefore Gmsh is used for this task. Then, the function *changeMeshParameters()* loads and changes the `*.geo` file that specifies the geometry and the function *remakeMesh()* uses the *"system"* call of C++ to call Gmsh and remesh the geometry, import it into OpenFOAM, and scale it by the scaling factors provided in the `optimizationOptions`. In order to make necessary changes to the `boundary` file, the function *AsaDsmcTest_1::remakeMesh()* overloads the function *DsmcOptimizationProblem::remakeMesh()* but calls the function of the parent class first and then calls the function *DsmcOptimizationProblem::changeBoundaryFile()*, which is a generic tool to change the boundary conditions of *boundary patches*. After that, the function *changeAdditionalParameters()* is called to make changes to other *dictionaries* that may have to be made when other parameters depend on one of the variables specified in the `optimizationOptions`. In the current example this is an additional change to the `dsmcInitialiseDict` to make the initial condition of the number density fit with the inflow properties.

After the new case is set up, the function *runCase()* is called, which reads the number of processors from the *decomposeParDict* and again uses the *"system"* command to decompose, initialise, run, and reconstruct the case and then calls the function *finalize()*, which executes the *foamLog* script to extract all data from the logfile and, if selected, Gnuplot is used to plot this data by a call to the script `$FOAM_SRC/optimization/Problems/DsmcOptimizationProblem/DsmcDataPlot.sh`. This may not seem to be a nice way of doing it, but it offers an easy way to change the plots without recompiling everything. Finally, the function *finalize()* deletes the logfiles and the `processor` folders to free disk space.

The functions *createNewCase()* and *runCase()* are called from within the function *costFunction()*, which now has the task of extracting the function value from the obtained data. In the current example, this means calculating the flux Φ through the tube from the fields which can be done according to the formula

$$\Phi = \frac{d(pV)}{dt} = k_\mathrm{B} T \frac{dN}{dt} = k_\mathrm{B} \int \vec{v} n \, d\vec{A} \cdot \frac{\int T \, d\vec{A}}{\int 1 \, d\vec{A}}, \tag{7.17}$$

where the last term is the average temperature on the surface with surface element $d\vec{A}$. For a better statistical result, the flow will be calculated through ten different planes along the tube and the average will be returned to the optimization algorithm. In order to sample the relevant data from the fields, OpenFOAM provides the *sample* utility which requires a dictionary `system/sampleDict` that specifies which fields are sampled at which planes, the interpolation scheme, and the output format. Because the tube is bent by different angles, the parameters of the planes to sample have to be

changed for every case, which is also done in the function *additionalParameterChanges()*. Then, the sampled data can be integrated by OpenFOAMs *sampledSurfaces* class and the average flow can be calculated. This completes the evaluation of the costfunction and the optimization algorithm can continue its work.

7.3 Test of the Solver *optimizationFoam*

7.3.1 Test of Optimization Algorithm ASA

In a first step, the optimization algorithm was tested with various test functions from [95]. In order to test different aspects of the optimization algorithm, the three functions $f_0(\vec{x})$, $f_2(\vec{x})$ and $f_4(\vec{x})$ were chosen. The first function is defined by

$$\begin{aligned} f_0(x_1,\ldots,x_4) &= \sum_{i=1}^{4} \begin{cases} (t_i \operatorname{sgn}(z_i) + z_i)^2 c d_i & \text{if } |x_i - z_i| < t_i \text{ and } |x_i| < t \\ d_i x_i^2 & \text{otherwise} \end{cases} \\ z_i &= \left\lfloor \left| \frac{x_i}{s_i} \right| + 0.49999 \right\rfloor \operatorname{sgn}(x_i) s_i, \\ s_i &= 0.2, \\ t_i &= 0.05, \\ c &= 0.15 \\ d_i &= \{1.0, 1000.0, 10.0, 100.0\}, \\ -10\,000 &\leq x_i \leq 10\,000. \end{aligned} \tag{7.18}$$

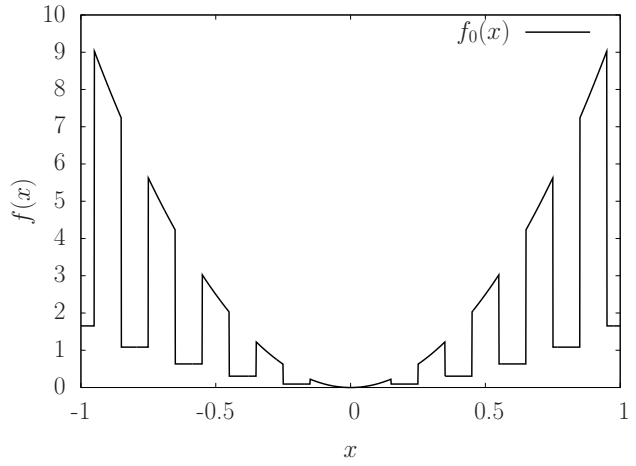

Figure 7.6 – *Test function function $f_0(\vec{x})$ along an axis with $d_i = 10$ around the minimum at $x = 0$.*

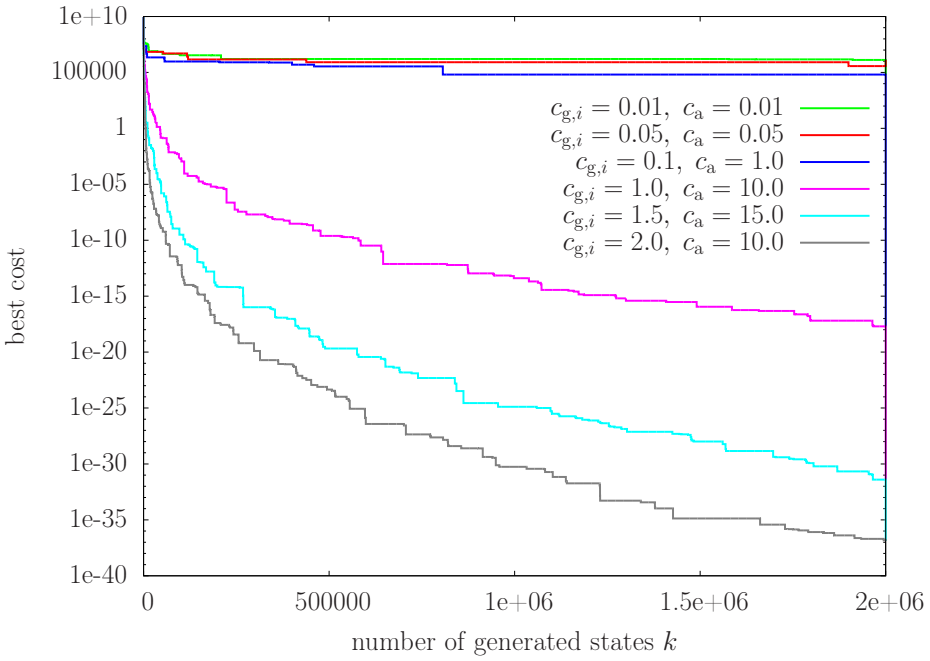

Figure 7.7 – *Best cost for the optimization of $f_0(\vec{x})$ with different cooling parameters $c_{g,i}$ for generation and c_a for acceptance temperature. The convergence behavior depends strongly on the chosen parameters and the algorithm does not converge for the first 3 parameter sets after $2 \cdot 10^6$ generated states while it rapidly converges for the last three.*

This is a paraboloid in four dimensions with steepness d_i along the axis x_i that has numerous holes of width $2t_i$ centered at $x_{i,j} = j \cdot s_i$ with increasing floor level for increasing $|x_{i,j}|$ (see Fig. 7.6). The function has one global minimum at $\vec{x} = 0$ where $f(0) = 0$ and $10^{20} - 1$ local minima [96]. This function is very difficult to optimize because of the large number of local minima and its discontinuity. Figure 7.7 shows that the present implementation of ASA can handle this function and find its global minimum within reasonable time for suitably chosen cooling parameters. This clearly shows the importance of optimizing the cooling parameters as well. It was not done here, so one can expect to find a set of cooling parameters that leads to much faster convergence than shown in Fig. 7.7. The initial values of the parameters x_i are only of minor importance for the convergence rate and were all set to $x_i(k = 0) = 10\,000$. The initial generation (acceptance) temperature was set to $T_{g,i}(0) = 1.0$ ($T_a(0) = 10^7$). Reannealing and quenching were not used.

The second function is

$$f_2(x_1, x_2) = 100(x_2 - x_1^2)^2 - (1 - x_1)^2 \qquad -2.048 \leq x_{1/2} \leq 2.048 \tag{7.19}$$

which has a global minimum at $x_1 = x_2 = 1$ with $f_2(1, 1) = 0$ and a parabolic valley stretches along $x_2 = x_1^2$. This valley makes the function hard to minimize [96]. Five optimization runs with different cooling parameters were performed with initial values

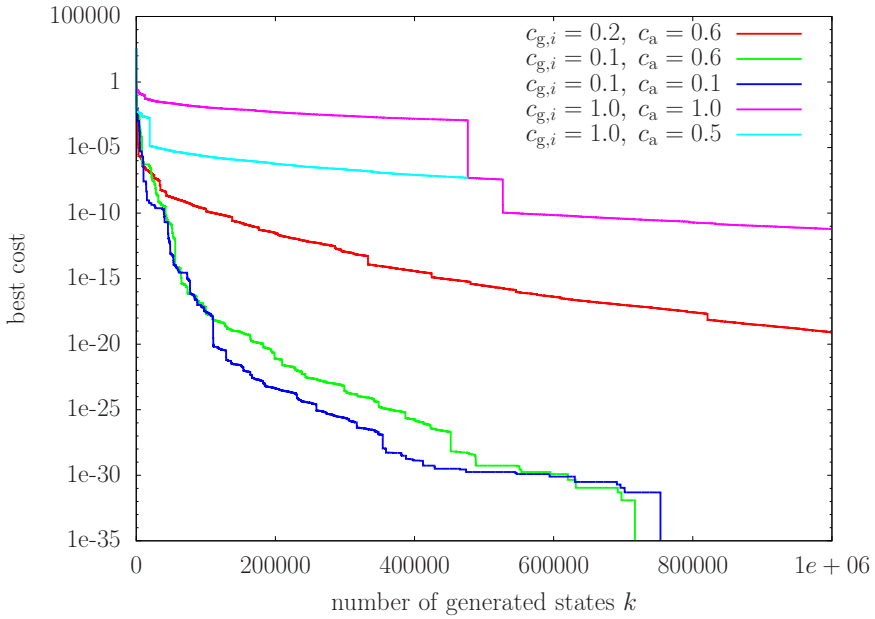

Figure 7.8 – *Best cost for the optimization of $f_2(\vec{x})$ with different cooling parameters $c_{g,i}$ for generation and c_a for acceptance temperature.*

$x_1(k=0) = 1.2$ and $x_2(k=0) = 1.2$. The results are shown in Fig. 7.8. Reannealing was performed every 200 accepted states during all runs with the stepsize $\Delta x_i = 0.004$ for the calculation of the partial derivatives. The behavior of the generation temperature with reannealing is shown in Fig. 7.9.

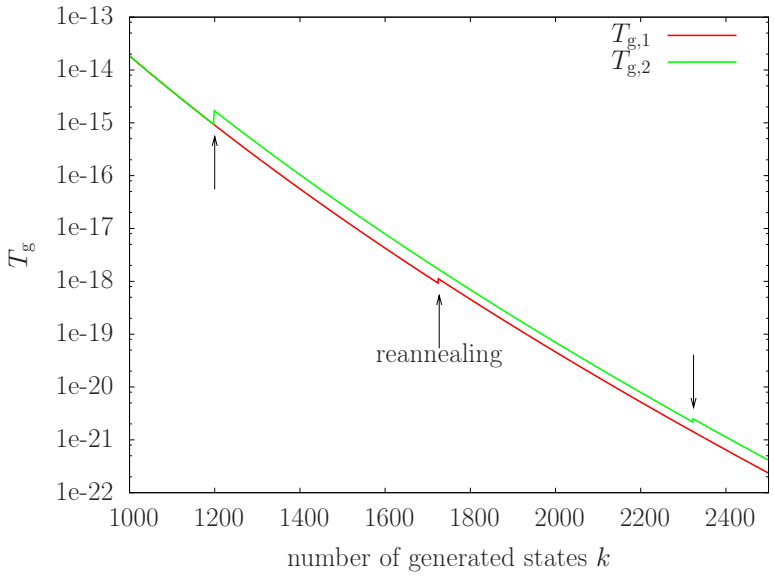

Figure 7.9 – *Generation temperatures $T_{g,1/2}$ during the optimization run of function $f_2(x_1, x_2)$ with $c_{g,i} = 1.0$ and $c_a = 1.0$. Steps where reannealing takes place are marked by arrows.*

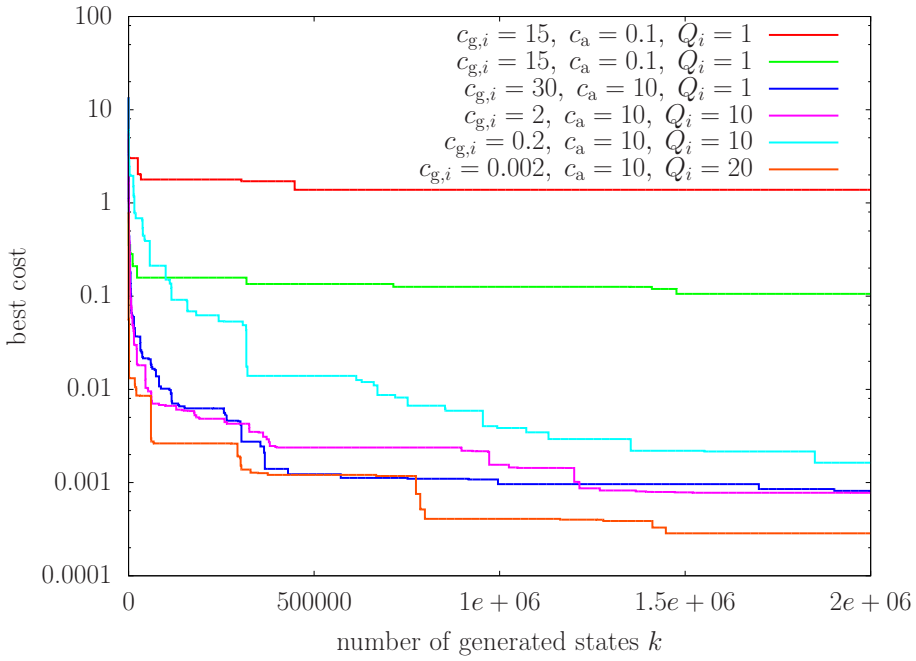

Figure 7.10 – *Best cost for the optimization of $f_4(\vec{x})$ with different cooling parameters. The only difference between the first (red) and second (green) curve is a difference in the initial acceptance temperature $T_{a,0}$ which is left unspecified in the first case and set to $T_{a,0} = 0.1$ in the second case. The initial values for all simulations were $x_{i,0} = 1$ except in the last case where they were set to $x_{i,0} = 0$ as a check.*

The third function to minimize is

$$f_4(x_1, \ldots, x_{30}) = \sum_{i=1}^{30} x_i^4 + \eta \qquad -1.28 \leq x_i \leq 1.28 \tag{7.20}$$

where $\eta \in [0, 1]$ is an equally distributed random number. The function has a global minimum at $x_i = 0$ and tests whether the algorithm can handle noisy functions. Figure 7.10 shows that this is not a problem and that the optimization works with quenching too. The optimization is quite fast for very different cooling parameters although it would be highly unlikely to find a minimum with a simple random walk. The parameter space is so big that the probability p of hitting the volume around the optimum with $|x_i| \leq 0.5$ only once within $2 \cdot 10^6$ steps is only $p = 2 \cdot 10^6 \cdot 1/(2.56)^{30} \approx 1.1 \cdot 10^{-6}$.

7.3.2 Test of *DsmcOptimizationProblem*

The test of the class *DsmcOptimizationProblem* was performed with the test case *AsaDsmcTest_1* described in Sec. 7.2. Because of the long computation time, no complete optimization could be performed but 30 different values for the costfunction were

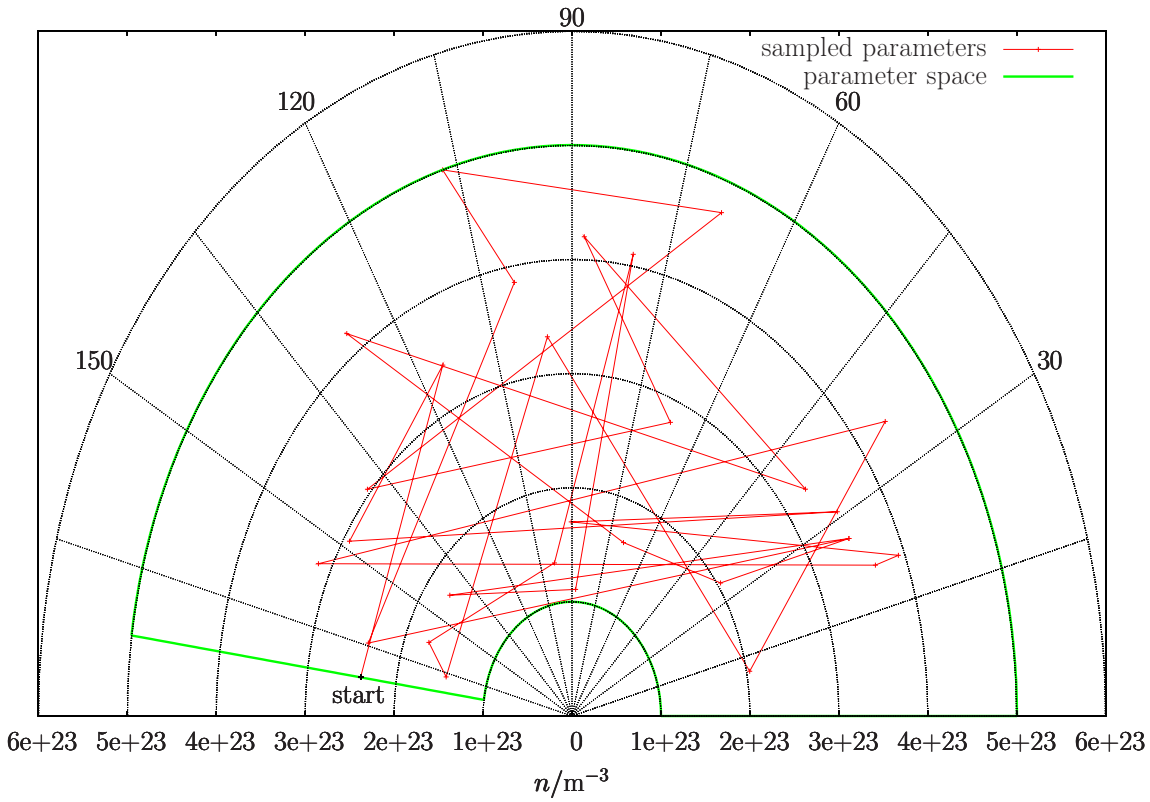

Figure 7.11 – *Sampled parameters during optimization of problem* AsaDsmcTest_1 *described in Sec. 7.2; The radial coordinate gives the number density n at the entrance of the tube and the azimuthal angle gives the bending angle φ of the tube. The starting point is marked in black and the green line delineates the allowed parameter space.*

calculated during two different runs, which also proves that the `restart` option is working. However, the main point here is, that the program is able to edit all configuration files, automatically change the mesh, start a new run and extract the relevant data from the output of the solver *dsmcSpinModFoam*. Figure 7.11 shows the generated points in parameter space in a polar plot, where the radial direction shows the number density n at the entrance of the tube and the polar angle shows the bending angle φ of the tube. The number density at the exit of the tube is kept constant at $n = 2.4 \cdot 10^{20}\,\mathrm{m}^{-3}$ throughout all runs. The flux Φ through the tube is plotted against the number density at the tube entrance in Fig. 7.12 and is fitted with a straight line $\Phi(n) = a \cdot n + b$ with $a = (2.80525 \pm 0.03231) \cdot 10^{-23}\,\mathrm{mbar \cdot l/(m^3 \cdot s)}$ and $b = -1.12635 \pm 0.1028\,\mathrm{mbar \cdot l/s}$. The data can be described perfectly by this fit, motivated by the Hagen-Poiseuille equation, but if one calculates $\Phi(n = 2.4 \cdot 10^{20}\,\mathrm{m}^{-3})$, the flux does not vanish. Figure 7.13 shows that the bending angle φ of the tube has no measurable influence, as is expected for a laminar flow. The reason for the non-vanishing flux at zero pressure gradient is the boundary condition `zeroGradient` used for the velocity at the inlet, which leads to a non-vanishing inflow velocity.

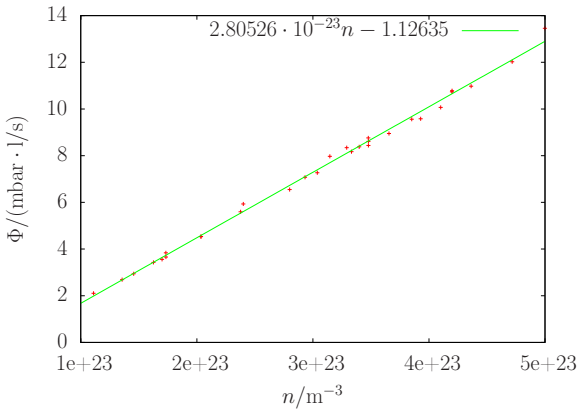

Figure 7.12 – *Plot of the flux Φ through the tube versus the number density n at the tube entrance together with a linear fit curve. Clearly, the lowest density (pressure) gradient between tube entrance and exit leads to the lowest flux through the tube.*

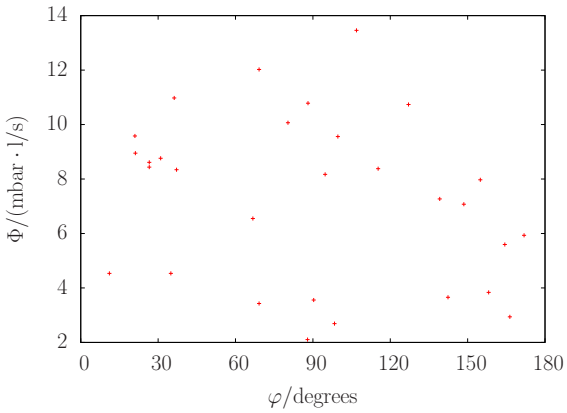

Figure 7.13 – *Plot of the flux Φ through a curved tube versus the bending angle φ of the tube; No relation is visible.*

8 Summary and Outlook

This thesis describes the development of a program to simulate the gas flow through an atomic beam source with the Direct Simulation Monte Carlo Method implemented by the open source program OpenFOAM 1.7.1. This program was extended to include magnetic fields which can act on particles with spin as a new property. High frequency transition units were added to the program as a second hardware part. Recombination and spin-exchange collisions were implemented. Further, the new program is able to determine the collision age distribution of particles if requested. For low densities, like the ones occuring in the storage cell, this can also be done with faster ray-tracing codes, but if collisions play an important role, like in the ABS, the new program will be more accurate. Last but not least, Adaptive Simulated Annealing was implemented as an optimization algorithm which can set up all the necessary configuration files for a DSMC calculation and which can be used to optimize the ABS. All parts of the program were shown to work properly in simple test cases. Some remaining problems with the high frequency transition units were detected and should become solved in the near term. The implementation of many parts of the program is very generic and interfaces using OpenFOAMs *runTimeSelection* mechanism are provided for several new program components. This allows one to easily implement new magnet types, surface recombination models and optimization algorithms without the need to care about much of the rest of the source code.

A first thing that should be done with the new program is to simulate the ABS for different running conditions and compare the result with measurements. This should help to identify and correct possible inaccuracies. It may also help to get a better understanding of the gas flow in the device. Further simulations of other atomic beam sources could help to clarify why these sources have different intensities.

The new program can further help to calibrate the polarization measurement of the BRP. This polarimeter is supposed to measure the polarization seen by the accelerator beam but the extracted gas for the polarimeter flows through a long tube before the measurement. Recombination and depolarization occur along the way. The program can help to estimate the polarization loss through spin-exchange collisions. Further, the effects of wall depolarization and the polarization of molecules can be estimated together with simulated data when the number of wall hits is counted. Turning this around, one can derive material properties like recombination probability and probability of wall depolarization from the combination of measurements and simulations. Further, the angular distribution of particles reflected from a wall can be determined

by comparing density profiles along the storage cell (from detector count rates during experiments) with simulations for different wall interaction models. This would lead to improvements in other values like the recombination probability since the number of wall hits could be simulated more accurately. Depending on the required accuracy, it may make sense to include wall depolarization and more elaborate wall interaction models as well as better recombination models in the program. For some applications it may also be useful to lift the artificial restriction to only three particle species of the program. In order to describe polarized molecules, it would be necessary to find and implement a more general description of the various spin orientations. For faster execution of the program it may make sense to implement the DSMC07 algorithm in OpenFOAM. The simulation of a complete ABS with its wide range of different pressures would certainly benefit a lot from such an improvement and it would allow to omit the approximations made with the *MixedInflow* inflow model.

The most important application of the new program however is the optimization of the ABS. This is not feasible on a current desktop PC because of the long execution time. Therefore, the program has to run on a supercomputer and it should be shown that it can run there. It would also be necessary to study the scaling behavior of the execution time with the number of processes. If this is bad, bottlenecks have to be found and should be eliminated. If this is not possible, one could also try to use another optimization algorithm that can be run in parallel, such that several calculations of the cost function run at the same time, each on a limited number of processors. This would however require a more elaborate *restart* routine for the optimization algorithm since users get usually only a certain time window for calculations on a supercomputer and they have to use several such time windows for the complete execution of the program.

Appendix

A Sextupole Magnets

In the atomic beam source two groups of sextupole magnets are used to separate the two different electron spin states via the Stern-Gerlach force. They consist of 24 segments each assembled in a Halbach configuration [29] made of three different NdFeB alloys.

According to [97], the radial and tangential components of the B-Field of a cylindrical 2N-multipole made from M permanently magnetized segments such that the magnetization axis of each segment advances by an angle $2\pi\,(N+1)/M$ with respect to a fixed coordinate system (compare Fig. A.1) can be written as

$$B_{rad}(r,\phi) = J\sum_{\nu=0}^{\infty}\left(\frac{r}{r_1}\right)^{n-1} H_n\,\cos(n\phi)\,K_n \tag{A.1}$$

$$B_{tan}(r,\phi) = J\sum_{\nu=0}^{\infty}\left(\frac{r}{r_1}\right)^{n-1} H_n\,\sin(n\phi)\,K_n \tag{A.2}$$

with

$$n = N+\nu M \tag{A.3}$$

$$H_n = \frac{n}{n-1}\left(1-\left(\frac{r_1}{r_2}\right)^{n-1}\right) \tag{A.4}$$

$$K_n = \frac{\sin((n+1)\epsilon\pi/M))}{(n+1)\pi/M} \tag{A.5}$$

where ϵ is a correction factor close to one for slits from glueing the segments together. For an ideal sextupole (no higher multipole terms) this reduces to

$$B_{rad}(r,\phi) = B_0\,\cos(3\phi)\left(\frac{r}{r_1}\right)^2 \tag{A.6}$$

$$B_{tan}(r,\phi) = B_0\,\sin(3\phi)\left(\frac{r}{r_1}\right)^2 \tag{A.7}$$

where $B_0 = J\,H_3\,K_3$ is the pole tip field. The B-Field in cartesian coordinates can therefore be written as

$$\vec{B} = B_0 \left(\cos(3\phi) \begin{pmatrix} 0 \\ \cos(\phi) \\ \sin(\phi) \end{pmatrix} + \sin(3\phi) \begin{pmatrix} 0 \\ -\sin(\phi) \\ \cos(\phi) \end{pmatrix} \right) \cdot \left(\frac{r}{r_1} \right)^2 \cdot B(x) \tag{A.8}$$

when the symmetry axis coincides with the x-axis. The angle ϕ is measured clockwise around the y-axis such that $\cos(\phi) = \frac{y}{\sqrt{y^2+z^2}}$ and $\sin(\phi) = \frac{z}{\sqrt{y^2+z^2}}$.

The factor $B(x)$ is 1 for normal cylindrical magnets, but is a function of x for conical magnets. From eq. A.4 one can see that the dependence of the pole tip field on the material thickness is given by the factor $1 - (r_1/r_2)^2$. For conical magnets the radius r_1 has to be replaced by the radius R at the correct position along the magnet, i.e. $R = r_{1i} + (r_{1f} - r_{1i})\frac{x}{L}$ where r_{1i} is the inner radius of the magnet at $x = 0$ and r_{1f} is the corresponding value at $x = L$ with L beeing the magnet length. With the correct normalization factor such that B_0 is the pole tip field in the center plane of the magnet one can write

$$B(x) = \frac{1 - \left(\frac{r_{1i} + \frac{x}{L}(r_{1f} - r_{1i})}{r_2} \right)^2}{1 - \left(\frac{r_{1i} + r_{1f}}{2r_2} \right)^2} \tag{A.9}$$

A more detailed fully analytical description of the field of cylindrical Halbach structures inside and outside of the magnet bore can be found in [98].

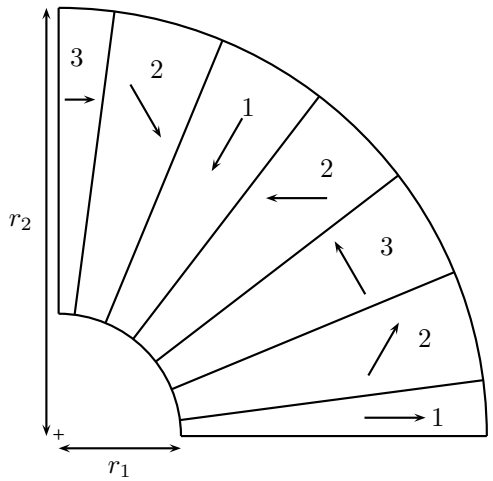

Figure A.1 – *One quadrant of a cylindrically symmetric sextupole magnet consisting of 24 segments. The arrows denote the direction of magnetization, while the number code indicates the material employed (as in [97]).*

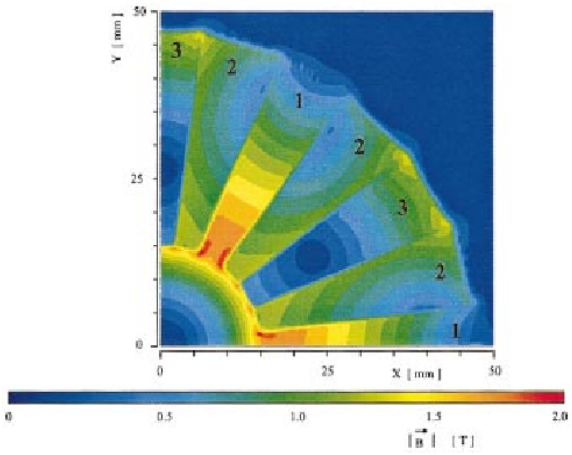

Figure A.2 – *Calculation of the magnitude of the magnetic flux density $|\vec{B}|$ in the central xy-plane of sextupole magnet 5 (from [97]).*

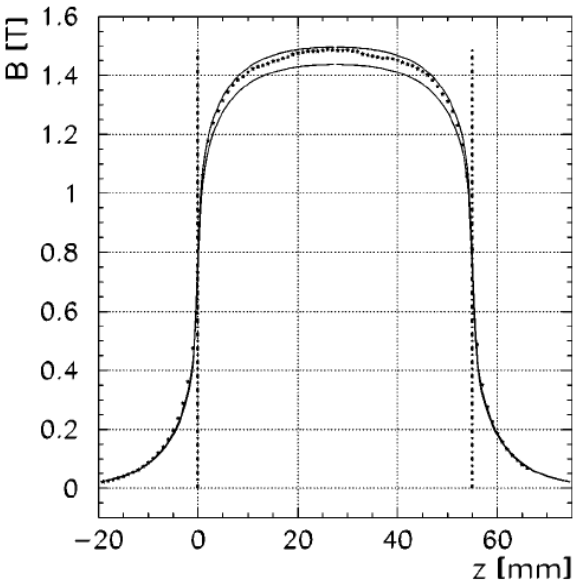

Figure A.3 – *Magnetic flux density $B(*)$ along the axis of magnet 6 at a radius of 14.5mm. The upper solid line indicates a MAFIA calculation using typical values for the remanence J^{typ} of the individual segments. The lower solid line for the configuration with minimal values of the magnetization is obtained by scaling the results obtained with J^{typ} by a factor J^{min}/J^{typ}. The dashed lines correspond to the physical boundaries of the magnet (from [97]).*

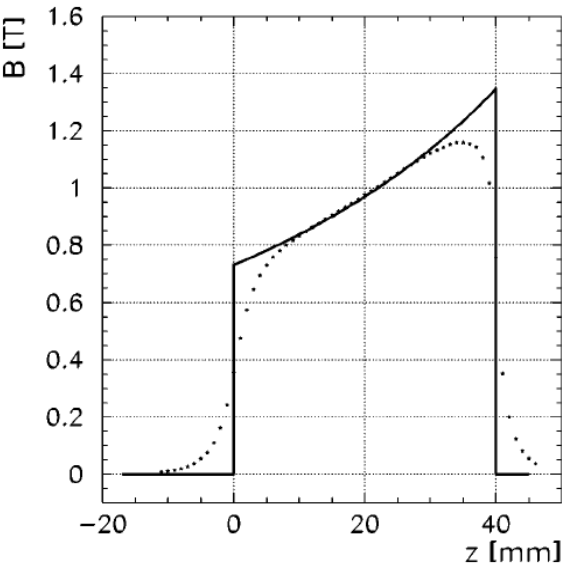

Figure A.4 – *Measured magnetic flux density B along the axis of magnet 1 at a radius of 4.5mm. The solid line corresponds to the result of a calculation based on the measured pole tip field $B(*)$, using a factor $(r/r_1)^2 \cdot [1 - (r_1/r_2)^2]$ to account for the radius r at which the distribution was measured and for the change in material thickness (from [97]).*

B Setup and File Structure of a Case

The setup of a DSMC case is mostly described in the OpenFOAM user guide [76] and some examples are delivered with OpenFOAM as tutorials. Every case folder has at least three subfolders, a start time folder (often `0`), `constant` and `system`. In the start time folder, a file for every field specifying the initial and boundary conditions has to be provided. The only new fields are the *volTensorFields hfsPopulationNumbers0* to *hfsPopulationNumbers2*, each for one particle species. The mesh information is contained in the folder `constant/polyMesh` which can be generated either with OpenFOAMs *blockMesh* utility from a *dictionary* or by importing a mesh generated with a dedicated program. The file `dsmcProperties` holds all DSMC specific data and is read only once at the beginning of the simulation. A complete example showing all new entries (but not all options) for the solver *dsmcSpinModFoam* can be found in appendix C. The `system` folder contains *dictionaries* controlling the behavior of various tools. The most important file is `controlDict` which controls the behavior of the *dsmcSpinModFoam* solver. An example is shown in appendix D. Several changes have to be made compared to the original solver *dsmcFoam*. First, the name of the application has changed as well as the name of the additional libraries that have to be loaded for postprocessing specified under the keyword *libs*. Then, there is the new option *hfsResetOnOutput* which specifies if the average field values of the hyperfine populations are kept after writing the average field to file or if the average for the next time step should start from the beginning again without considering the previous time steps. When the simulation reaches the steady state, one wants to average over many time steps to reduce the statistical error but the time steps before reaching the steady state shouldn't be included in the average. This is why the new option *startHfsLongTimeAverageAt* is created that specifies at which time the averaging should start. Otherwise one would have to change this manually during the simulation. Basically the same option called *startLongTimeAverageAt* specified in the *fieldAverage* section was created for the average of all other fields. The difference in these two is that the average of the hyperfine population numbers has to be calculated differently than that for the other fields as described in Sec. 5.1.2 and should therefore not be handled by the *fieldAverage* class. Another file that contains additional options is `dsmcInitialiseDict`. In this file the value of the hyperfine population numbers for the tool *dsmcSpinModInitialise* can be set by the command

```
hfsPopulationNumbers0 (1 0 0 0 0 0 0 0 0);
```

and analog for the other fields. When all configuration files are written, the simulation can be initialized with the command *dsmcSpinModInitialise* and run with *dsmcSpinModFoam* or alternatively with the corresponding commands for parallel execution.

As mentioned above, the mesh can be generated with third party software and can be imported into OpenFOAM. For most of the work, Gmsh was used for creating the mesh and for some difficult meshes like for the storage cell, the open-source CAD tool SALOME [99] was used. Gmsh can be used as command line tool and the geometry can be specified in a simple way in a `*.geo` file. Meshing can be done from the command line as well and the result will be saved in a `*.msh` file. This can be imported into OpenFOAM with the *gmshToFoam* tool. This will create the `polyMesh` folder. The only action that has to be done afterwards is to edit the `boundary` file in order to set the correct boundary properties like *wall* or *cyclic*. This can be done manually or with the *createPatch* utility which can also combine front and back side of a wedge to one patch which is required for rotationally symmetric cases using the *cyclic* boundary condition. If SALOME is used for mesh generation, the mesh is saved in `*.med` format, then imported into Gmsh and converted into a `*.msh` file. The following steps are like before.

Setting up an optimization case requires one more hierarchy level in the folder structure as can be seen in Fig. B.1. The top level `system` folder needs to contain only two files with pretty arbitrary content. They can just be copies of the corresponding files from a lower level `system` folder and the only necessary additional input in the top level `controlDict` file is the entry

```
allowWritingInArbitraryPlace true;
```

which loosens restrictions of OpenFOAM on the file structure as described in Sec. 7.2. The only other required file is `optimization/optimizationOtions` which specifies which optimization problem shall be solved by which algorithm and which parameters shall be considered. Additionally, the parameter ranges, initial values and specific data for the problem and algorithm can be specified. A complete example of the file `optimizationOptions` for the case described in Sec. 7.2 is shown in appendix E. If the optimization is done with DSMC, a folder `TestState_0` which contains the complete set up of a DSMC case has to be present as well. If the mesh is part of the optimization and Gmsh is used for meshing, a `*.geo` file has to be present as well in the `TestState_0` folder which will be edited by the program and Gmsh will be called for remeshing and creating the corresponding `*.msh` file that will be imported automatically into OpenFOAM. If only a simple function shall be minimized, the folder `TestState_0` is not necessary and the section *DsmcProblemOptions* can be omitted in the file *optimizationOptions*. If a completely new problem shall be tackled, one first has to program it and Fig. 7.4 shows which functions the new class must implement. Then, the new problem class has to be listed together with all algorithms to solve it in the file `optimization/Problems/BaseProblem/makeOptimizationProblems.C` and the problem has to be compiled which can be done using the *buildopt.sh* script delivered with the source code of the solver. Finally, starting a new problem can simply be done with the command *optimizationFoam* from within the optimization folder, i.e. `AsaDsmcTest_1` in the example of Fig. B.1.

Figure B.1 – *Folder structure of an optimization case were the cost function is calculated with the* dsmcSpinModFoam *solver and the geometry is part of the optimization. All folders and files are shown that are needed for the setup of an optimization case. The top level* `system` *folder needs to contain only the two shown files were the content of* `dsmcInitialiseDict` *can be arbitrary. The file* `optimization/AsaState` *needs only to be present if Adaptive Simulated Annealing (ASA) is used as optimization algorithm and the option* restart *is set to true in the* `optimizationOptions`*.* `TestState_0` *contains everything a single DSMC case contains as described in the OpenFOAM user guide [76], except for the additional files* `hfsPopulationNumbers0` *to* `hfsPopulationNumbers2`*. If the geometry is part of the optimization and the mesh is generated with Gmsh, a file* `*.geo` *is required from which the corresponding* `*.msh` *file is generated. During the calculation an additional file* `TestState_xxparameterInfo` *will be generated containing the values of the parameters used for the case xx. The folder* `TestState_1` *will be generated automatically by the solver and is shown only for clarification.*

C Example of File `dsmcProperties`

```
 /*--------------------------------*- C++ -*----------------------------*\
| =========                 |                                             |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox       |
|  \\    /   O peration     | Version:  1.7.1                             |
|   \\  /    A nd           | Web:      www.OpenFOAM.com                  |
|    \\/     M anipulation  |                                             |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      dsmcProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //


// General Properties
// ~~~~~~~~~~~~~~~~~~

nEquivalentParticles            1e10;


// Wall Interaction Model
// ~~~~~~~~~~~~~~~~~~~~~~

WallInteractionModel            MaxwellianThermalCosineExponent;

MaxwellianThermalCosineExponentCoeffs
{
  cosineExponent 2;
}


// Binary Collision Model
// ~~~~~~~~~~~~~~~~~~~~~~
```

```
BinaryCollisionModel   LarsenBorgnakkeVariableSoftSphereLennardJones;

LarsenBorgnakkeVariableSoftSphereLennardJonesCoeffs
{
    relaxationCollisionNumber   300;
}


// Inflow Boundary Model
// ~~~~~~~~~~~~~~~~~~~~~

InflowBoundaryModel             MixedInflow;

MixedInflowCoeffs
{
  inflow
  {
    inflowModelType InOutflow;
    cellInternalValues  true;

    numberDensities
    {
      H         8e21;
      H2        4e21;
    };
  }
  outflow
  {
    inflowModelType InOutflow;
    cellInternalValues  false;

    numberDensities
    {
      H         1e16;
      H2        5e15;
    };
  }
}


// Molecular species
// ~~~~~~~~~~~~~~~~~
```

```
typeIdList (H H2);

moleculeProperties
{
    H
    {
      mass                              1.67e-27;    // kg
      internalDegreesOfFreedom          0;
      radius                            2.81e-10;    // m
      epsilon                           1.187e-22;   // J
      diameter                          0;
      viscosityCoefficient              0;
      omega                             0.67;
      electronicS                       0.5;
      nuclearI                          0.5;
      ehfs                              9.412e-25;   // J
    }

    H2
    {
        mass                            3.34e-27;
        internalDegreesOfFreedom        2;
        radius                          2.928e-10;
        epsilon                         51.084e-23;
        diameter                        0;
        viscosityCoefficient            0;
        omega                           0.67;
        ehfs                            0;
        electronicS                     0;
        nuclearI                        0;
    }
}


// Forces
// ~~~~~~~~~~~~~~~~~~~~

forceIdList (S1);

forceProperties
{
    S1
    {
      type                MagneticCylindricalSextupole;
```

```
      origin              (0.03 0 0);
      P2                  (0.05 0 0);
      r1Origin            0.01;
      r1End               0.01;
      r2                  0.03;
      B0                  1.5;
      vectorB0            (1 0 0);
      phi                 0.0;
    }
}


// High frequency transition units
// ~~~~~~~~~~~~~~~~~~~~~

hfTransitionIdList (HF1);

hfTransitionProperties
{
  HF1
  {
    shape               circle;
    radius              0.03;
    faceCenter          (0.01 0 0);
    faceNormal          (1 0 0);
    efficiencies
    {
      H
      {
        efficiency (
          1 0 0 0
          0 0 0 1
          0 0 1 0
          0 1 0 0
        );
      }

      H2 //program assumes always spin-1/2 but does not use it
      {
        efficiency (
          1 0
          0 1
        );
      }
```

```
    }
  }
}


//ChemicalSurfaceReactionType
// ~~~~~~~~~~~~~~~~~~~~~

ChemicalSurfaceReactionModel simpleLangmuirHinshelwood;

ChemicalSurfaceReactionList (hydrogenRecombination);

ChemicalSurfaceReactions
{
  hydrogenRecombination
  {
    Reactants (H H);
    Products (H2);

    reactionParameters
    {
      patchGroup_1
      {
        patchNames (wall);

        parameters
        {
          gamma 5e-3;
        }
      }
    }
  }
}

// write particles hitting these patches to file
// ~~~~~~~~~~~~~~~~~~~~~

writeParticlesAtPatch (inflow outflow);
```

D Example of file system/controlDict

```
 /*--------------------------------*- C++ -*----------------------------*\
| =========                 |                                            |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox      |
|  \\    /   O peration     | Version:  1.7.1                            |
|   \\  /    A nd           | Web:      www.OpenFOAM.com                 |
|    \\/     M anipulation  |                                            |
\*---------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      controlDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

application     dsmcSpinModFoam;
startFrom       startTime;
startTime       0.0;
stopAt          endTime;
endTime         0.2;
deltaT          1e-7;
writeControl    runTime;
writeInterval   1e-4;
purgeWrite      3;
writeFormat     ascii;
writePrecision  10;
writeCompression  uncompressed;
timeFormat      general;
timePrecision   10;
runTimeModifiable yes;
adjustTimeStep  no;

hfsResetOnOutput true;
startHfsLongTimeAverageAt 0.01;
```

```
libs
(
    "libdsmcSpinMod.so"
    "libdsmcSpinPlusFields.so"
);

functions
(
    dsmcPlusFields
    {
        type dsmcPlusFields;
        enabled   on;
        functionObjectLibs ( "libutilityFunctionObjects.so" );
        outputControl     outputTime;
        resetOnOutput     off;
    }
    fieldAverage
    {
        type fieldAverage;
        functionObjectLibs ( "libfieldFunctionObjects.so" );
        outputControl     outputTime;
        resetOnOutput     on;
        startLongTimeAverageAt 0.01;
        fields
        (
            rhoN
            {
                 mean on;
                 prime2Mean off;
                 base time;
            }

            possibly other fields
        );
    }
);
```

E Configuration File `optimizationOptions`

The configuration file for optimization runs `optimizationOptions` is separared into four parts. The first part contains some general information about the optimization algorithm and optimization problem while the second part contains the parameters, their initial values and maximal range which may be refined later in the optimization problem class to allow for more complex boundaries. The third part is a *dictionary* holding configurations for the optimization algorithm used and the last part is a *dictionary* holding settings for the optimization problem. Below is an example for the case *AsaDsmcTest_1* described in Sec. 7.2.

```
 /*--------------------------------*- C++ -*----------------------*\
| =========                 |                                     |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox   |
|  \\    /   O peration     | Version:  1.7.1                     |
|   \\  /    A nd           | Web:      www.OpenFOAM.com          |
|    \\/     M anipulation  |                                     |
\*---------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "optimization";
    object      optimizationOptions;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

OptimizationAlgorithm AsaOptimization;

OptimizationProblem AsaDsmcTest_1;

minimize        true;

restart         true;

ParameterIdList ( phi inflow );

Parameters
```

```
{
    phi
    {
        upperLimit      3;
        lowerLimit      1e-06;
        initialValue    3;
    }
    inflow
    {
        upperLimit      5e+23;
        lowerLimit      1e+23;
        initialValue    2.4e+23;
    }
}

ASAOptions
{
    initialGenerationTemperature ( 1 1 );
    generationDampingCoeffs ( 0.1 0.1 );
    acceptanceDampingCoeff 0.2;
    generationQuenchingFactors ( 1 1 );
    acceptanceQuenchingFactor 1;
    reannealStep    200;
    maxGenStates    1500;
    deltaParameters ( 0.01 1e+18 );
}

DsmcProblemOptions
{
    parameterPathes
    {
        phi             ( mesh );
        inflow          (
                          dsmcProperties InOutflowCoeffs
                          inflow numberDensities H
                        );
    }
    meshScaling     ( 0.001 0.001 0.001 );
    startTime       0;
    countPatchHits  false;
    useGnuplot      true;
    useGmsh         true;
}
```

List of Figures

List of Tables

Bibliography

[1] P. Lenisa *et al.*, arXiv preprint hep-ex/0505054 (2005).

[2] W. Haeberli, Annual Review of Nuclear Science **17**, 373 (1967).

[3] D. Grosnick *et al.*, Nucl. Instr. Meth. A **290**, 269 (1990).

[4] P. Csonka, Nucl. Instr. Meth. **63**, 247 (1968).

[5] F. Rathmann *et al.*, Phys. Rev. Lett. **71**, 1379 (1993).

[6] P. Baumann *et al.*, Nucl. Instr. Meth. A **268**, 531 (1988).

[7] C. Weidemann, Ph.D. Thesis, Universität zu Köln, 2011, available under: http://kups.ub.uni-koeln.de/id/eprint/4513.

[8] W. Augustyniak *et al.*, Physics Letters B **718**, 64 (2012).

[9] R. Maier, Nucl. Instr. Meth. A **390**, 1 (1997).

[10] A. Milstein and V. Strakhovenko, Physical Review E **72**, 066503 (2005).

[11] N. Nikolaev and F. Pavlov, arXiv preprint hep-ph/0601184 (2006).

[12] C. Barschel *et al.*, ArXiv e-prints 0904.2325 (2009).

[13] M. Düren *et al.*, Nucl. Instr. Meth. A **322**, 13 (1992).

[14] N. Koch, Ph.D. Thesis, Universität Erlangen, Erlangen, 1999, available under: http://www-hermes.desy.de/notes/pub/99-LIB/koch.99.011.ps.gz.

[15] A. Nass, Ph.D. Thesis, Universität Erlangen, Erlangen, 2002, available under: http://www-library.desy.de/preparch/desy/thesis/desy-thesis-02-012.ps.gz.

[16] L. Isaeva *et al.*, Nucl. Instr. Meth. **411**, 201 (1998).

[17] T. Wise, A. Roberts, and W. Haeberli, Nucl. Instr. Meth. A **336**, 410 (1993).

[18] F. Stock *et al.*, Nucl. Instr. Meth. A **343**, 334 (1994).

[19] R. Hertenberger *et al.*, Review of Scientific Instruments **69**, 750 (1998).

[20] M. Dyug *et al.*, Nucl. Instr. Meth. A **495**, 8 (2002).

[21] F. Rathmann *et al.*, AIP Conference Proceedings **675**, 924 (2003).

[22] M. Mikirtychyants *et al.*, Nucl. Instr. Meth. A **721**, 83 (2013).

[23] A. Nass *et al.*, Nucl. Instr. Meth. A **505**, 633 (2003).

[24] A. Zelenski *et al.*, Nucl. Instr. Meth. A **536**, 248 (2005).

[25] T. Wise *et al.*, Nucl. Instr. Meth. A **556**, 1 (2006).

[26] L. W. Anderson, F. M. Pipkin, and J. C. Baird, Phys. Rev. **120**, 1279 (1960).

[27] K. Shuler and K. Laidler, The Journal of Chemical Physics **17**, 1212 (1949).

[28] H. Wise, C. M. Ablow, and K. M. Sancier, The Journal of Chemical Physics **41**, 3569 (1964).

[29] K. Halbach, Nucl. Instr. Meth. **169**, 1 (1980).

[30] R. Philpott, Nucl. Instr. Meth. A **259**, 317 (1987).

[31] C. Baumgarten, Ph.D. Thesis, LMU, München, 2000, available under: http://www-hermes.desy.de/notes/pub/00-LIB/baumgarten.00.038.thesis.ps.gz.

[32] D. W. Trainor, D. O. Ham, and F. Kaufman, The Journal of Chemical Physics **58**, 4599 (1973).

[33] C. Rettner and D. Auerbach, Surface Science **357â358**, 602 (1996).

[34] D. Shalashilin *et al.*, Faraday Discussions **110**, 287 (1998).

[35] V. Guerra, Plasma Science, IEEE Transactions on **35**, 1397 (2007).

[36] T. Wise *et al.*, Phys. Rev. Lett. **87**, 042701 (2001).

[37] E. M. Purcell and G. B. Field, apj **124**, 542 (1956).

[38] T. Walker and L. Anderson, Nucl. Instr. Meth. A **334**, 313 (1993).

[39] M.-A. Bouchiat, Journal de Physique **24**, 379 (1963).

[40] M.-A. Bouchiat, Journal de Physique **24**, 611 (1963).

[41] M. A. Bouchiat and J. Brossel, Phys. Rev. **147**, 41 (1966).

[42] D. Swenson and L. Anderson, Nucl. Instr. Meth. B **29**, 627 (1988).

[43] B. Braun, Ph.D. Thesis, LMU, München, 1995, available under: http://www-hermes.desy.de/notes/pub/95-LIB/braun.95.047e.thesis.ps.gz.

[44] J. Stenger and K. Rith, Nucl. Instr. Meth. A **361**, 60 (1995).

[45] J. Stenger and K. Rith, Nucl. Instr. Meth. A **378**, 360 (1996).

[46] J. Price and W. Haeberli, Nucl. Instr. Meth. A **349**, 321 (1994).

[47] HERMES Collaboration, European Physical Journal D **29**, 21 (2004).

[48] R. Gorski, Diploma thesis, RWTH, Aachen, 2013.

[49] Ralf Engels, private communication.

[50] B. J. Alder and T. E. Wainwright, The Journal of Chemical Physics **31**, 459 (1959).

[51] G. A. Bird, Physics of Fluids **6**, 1518 (1963).

[52] W. Wagner, Journal of Statistical Physics **66**, 1011 (1992).

[53] T. Wainwright, The Journal of Chemical Physics **40**, 2932 (1964).

[54] F. J. Alexander, A. L. Garcia, and B. J. Alder, Physics of Fluids **10**, 1540 (1998).

[55] F. J. Alexander, A. L. Garcia, and B. J. Alder, Physics of Fluids **12**, 731 (2000).

[56] N. G. Hadjiconstantinou, Physics of Fluids **12**, 2634 (2000).

[57] G. Chen and I. D. Boyd, Journal of Computational Physics **126**, 434 (1996).

[58] D. J. Rader, M. A. Gallis, J. R. Torczynski, and W. Wagner, Physics of Fluids **18**, 077102 (2006).

[59] G. A. Bird, *Molecular Gas Dynamics and the Direct Simulation of Gas Flows*, 3rd ed. (Oxford Science Publications, Oxford, 1998).

[60] G. Bird, Sophisticated DSMC, Notes prepared for a short course at the DSMC07 meeting, Santa Fe, USA, 2007.

[61] M. Gallis, J. Torczynski, D. Rader, and G. Bird, Journal of Computational Physics **228**, 4532 (2009).

[62] S. E. Olson and A. J. Christlieb, Journal of Computational Physics **227**, 8035 (2008).

[63] H. Matsumoto, Physics of Fluids **20**, 097103 (2008).

[64] W. G. Vincenti and C. H. Kruger, *Introduction to physical gas dynamics* (Wiley, New York, 1965).

[65] G. A. Bird, Progress in Astronautics and Aeronautics **74**, 239 (1981).

[66] K. Koura and H. Matsumoto, Physics of Fluids A **3**, 2459 (1991).

[67] J. O. Hirschfelder, R. B. Bird, and E. L. Spotz, The Journal of Chemical Physics **16**, 968 (1948).

[68] H. Rabitz and S.-H. Lam, The Journal of Chemical Physics **63**, 3532 (1975).

[69] R. J. Gallagher and J. B. Fenn, The Journal of Chemical Physics **60**, 3492 (1974).

[70] K. Kern, R. David, and G. Comsa, The Journal of Chemical Physics **82**, 5673 (1985).

[71] C. A. Boitnott and J. R. C. Warder, Physics of Fluids **14**, 2312 (1971).

[72] A. Naß and E. Steffens, Nucl. Instr. Meth. A **598**, 653 (2009).

[73] P. Valentini, C. Zhang, and T. E. Schwartzentruber, Physics of Fluids **24**, 106101 (2012).

[74] G. Bird, Progress in Astronautics and Aeronautics **117**, 211 (1989).

[75] OpenFOAM wiki page, http://www.openfoamwiki.net.

[76] OpenFOAM User Guide, Version 1.7.1, August 2010.

[77] OpenFOAM Programmer's Guide, Version 1.7.1, August 2010.

[78] OpenFOAM download page, http://www.openfoam.org/download/archive.php.

[79] H. G. Weller, G. Tabor, H. Jasak, and C. Fureby, Computers in physics **12**, 620 (1998).

[80] H. Jasak, A. Jemcov, and Z. Tukovic, in *International Workshop on Coupled Methods in Numerical Dynamics* (IUC, Dubrovnik, Croatia, 2007), pp. 1–20.

[81] T. Scanlon *et al.*, Computers & Fluids **39**, 2078 (2010).

[82] F. Stollmeier, AIP conference proceedings **1333**, 331 (2011).

[83] M. Culpo, PRACE white paper to appear on http://www. praceri. eu (2011).

[84] ParaView homepage, http://www.paraview.org.

[85] Physimmasters homepage, http://www.physimmasters.com/wordpress/?page_id=187.

[86] M. Stancari, Internal Note INFN Ferrara INFN/TC-06/11, 2006 (http://www.lnf.infn.it/sis/preprint).

[87] D. Singy, P. Schmelzbach, W. Grüebler, and W. Zhang, Nucl. Instr. Meth. B **47**, 167 (1990).

[88] D. H. Wolpert and W. G. Macready, Evolutionary Computation, IEEE Transactions on **1**, 67 (1997).

[89] D. Fogel, Neural Networks, IEEE Transactions on **5**, 3 (1994).

[90] F. Stock, Ph.D. Thesis, Max-Planck Institut für Kernphysik, Heidelberg, 1994.

[91] L. Ingber, CoRR **cs.MS/0001018**, (2000).

[92] L. Ingber, Mathematical and Computer Modelling **12**, 967 (1989).

[93] C. Geuzaine and J.-F. Remacle, International Journal for Numerical Methods in Engineering **79**, 1309 (2009).

[94] Gnuplot homepage, http://www.gnuplot.info.

[95] L. Ingber and B. Rosen, Mathematical and Computer Modelling **16**, 87 (1992).

[96] A. Corana, M. Marchesi, C. Martini, and S. Ridella, ACM Transactions on Mathematical Software (TOMS) **13**, 262 (1987).

[97] A. Vassiliev *et al.*, Review of Scientific Instruments **71**, 3331 (2000).

[98] R. Ravaud, G. Lemarquand, V. Lemarquand, and C. Depollier, Progress In Electromagnetics Research B **11**, 281 (2009).

[99] Salome homepage, http://www.salome-platform.org.

Acknowledgments

Several people were important for this work and I would like to thank them very much. First of all, I want to thank Prof. Dr. H. Ströher for making this work possible in this institute. Secondly, I want to thank Prof. Dr. J. Jolie for acting as second referee.

Many thanks go to Dr. Alexander Nass who was my supervisor during the work. He could answer all my questions about the ABS and let me plenty of freedom during my work.

Further, I want to thank Dr. Hellmut Seyfarth and Dr. Michael Hartmann for proof-reading my thesis.

A good working environment is not only pleasant but also very important for good work. I was lucky to work in a very conductive environment. Dr. Ralf Engels always kept me up to date with the news in our topic from all around the world and there were many interesting and entertaining coffe room discussions with e.g. Prof. Dr. Detlev Gotta, Prof. Dr. Rudolf Maier, Dr. Hans-Joachim Stein and Philipp Weiß. Further participants were Dr. Christian Weidemann and Ilhan Engin who became very good friends. The Italian Group from Ferrara also contributed to the pleasant atmosphere in the institute. Thank you all!

During my work I was able to visit some very interesting places but I am especially grateful to Dr. Andro Kacharava for organizing a very interesting trip to Georgia and to all who made this journey possible.

Last but not least I want to thank my family for supporting me throughout my whole life.

Erklärung

Ich versichere, dass ich die von mir vorgelegte Dissertation selbständig angefertigt, die benutzten Quellen und Hilfsmittel vollständig angegeben und die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken im Wortlaut oder dem Sinn nach entnommen sind, in jedem Einzelfall als Entlehnung kenntlich gemacht habe; dass diese Dissertation noch keiner anderen Fakultät oder Universität zur Prüfung vorgelegen hat; dass sie - abgesehen von unten angegebenen Teilpublikationen - noch nicht veröffentlicht worden ist sowie, dass ich eine solche Veröffentlichung vor Abschluss des Promotionsverfahrens nicht vornehmen werde. Die Bestimmungen der Promotionsordnung sind mir bekannt. Die von mir vorgelegte Dissertation ist von Prof. Dr. H. Ströher betreut worden.

Martin Gaißer

Teilpublikationen

- M. Gaisser, A. Nass, H. Ströher, Conference Proceedings for the 20$^{\text{th}}$ International Symposium on Spin Physics (SPIN2012), JINR, Dubna, Russia, to be published
- M. Gaisser, A. Nass, H. Ströher, Conference Proceedings for the International Workshop on Polarized Sources, Targets & Polarimetry (PSTP2013), Charlottesville, USA, to be published