Extending the capabilities of a plasma simulation model

a case study of a hollow cathode discharge

B.H.P.Broks

August 20, 2002

Summary

This work, which is the report of a one-year graduation project at the Eindhoven University of Technology, deals with the extension of a plasma simulation code called PLASIMO, with the purpose of modeling high-power density plasmas.

The plasma simulation code PLASIMO has been the focus of the plasma simulation efforts on the Eindhoven University of Technology for the last decade. The present code is a model factory, capable of supporting models for a wide variety of plasmas. In order to expand the capabilities of PLASIMO to the simulation of high-power density plasmas, such as pinch plasmas or the plasma used in Laser-Wakefield acceleration, various extensions to the code have been made, which will be discussed here.

First, a 0-D model, based on the Disturbed Bilateral Relations (DBR), has been made. This model is capable of simulating a fairly wide range of plasmas with a low to moderate degree of ionization. The results of these simulation can be used by PLASIMO as starting conditions for the iterative solver. A code-to-code comparison between PLASIMO and the DBR model is described.

The principle of DBR can also be applied for the explanation of numerical results generated by PLASIMO. The results of a series of simulations that go from a Non-Local Thermal Equilibrium to a Local Thermal Equilibrium situation will be discussed with the DBR in mind.

Then, a module capable of calculating the Lorentz force and azimuthal magnetic field in an axisymmetric situation has been written. This Lorentz force is the driving force behind pinching. While the module is capable of correctly calculating the Lorentz force for a given current distribution, the weak coupling between the pressure and Lorentz force in the flow solvers may present a challenge for the simulation of pinch plasmas.

After a request from the industry, a case study of the decay of plasma in a hollow cathode pseudospark discharge has been investigated. For this, it was necessary to improve the treatment of diffusion in PLASIMO. The Fick law of diffusion, which was used in PLASIMO, did not apply for this type of discharge due to the low pressure. A module which augments Fick diffusion by taking into account the limitations on the mean free path the vessel causes has been added. This makes it possible to simulate plasmas that are between the continuum and free molecular flow regime. Using this module, the effect of geometry on the decay time of the hollow cathode discharge has been investigated.

Contents

Chapter 1

Introduction

1.1 Outline of the project

This thesis is the report of a one-year graduation project at the Eindhoven University of Technology. The purpose of the project was to extend the possibilities of the PLASIMO modeling package so that it can be used to model high-power high-density plasmas[1].

1.2 High power density plasmas

High power density plasmas are the focus of much research nowadays. They are for instance used in the laser-wakefield acceleration project (An introduction, along with literature references, can be found in [2]), and there is a recent industrial interest for hollow cathode pseudospark discharges for use as Extreme-Ultra Violet (EUV) sources.

To handle this class of plasmas, the modeling toolkit should be able to handle the following:

- Time-dependence. Many high power density plasmas are not stable. This makes their behavior time-dependent. PLASIMO can handle time-dependent models, as a study of the plasma in a hollow cathode discharge will show (see chapter 7).
- Pinching. A module that can calculate the Lorentz force for a two-dimensional geometry is discussed in this report (see chapter 6).
- The long mean free path in the hollow cathode discharge makes it necessary to take into account free molecular flow and normal diffusion. A module that can do this is created and discussed here. (see chapter 7).
- The plasma has a high electron density. This means that equilibrium can be maintained fairly easily. However, the strong time dependence introduces deviations from equilibrium. This makes it nescessary to be

[1] It is assumed that the reader has a basic understanding of plasma physics; if this is not the case, [1] can be a starting point to gain this understanding.

able to handle both plasmas that are close to equilibrium and plasmas that are far from equilibrium. (see chapter 5)

1.3 PLASIMO

For the modeling, the PLASIMO (PLAsma SImulation MOdel) package has been used. PLASIMO is a general-purpose modeling toolkit that has been the center of the plasma modeling efforts on the Eindhoven University of Technology for the last decade. Some PLASIMO features are:

- A modern object-oriented structure, written in C++
- A Graphical User Interface (GUI) that makes it easy to modify and create models, even by unexperienced users
- An input file parser that can handle custom expressions and uses units.
- The code can be compiled on a variety of platforms, among which Windows, Sillicon IRIX and Linux

Furthermore, the code is so general that new modules can easily be added. This makes it possible to tackle problems that cannot be solved with PLASIMO yet. Such new modules also extend PLASIMO, making it capable of handling more problems. An addition by one user is in principle usable by all users. Extendability and reusability are the key elements of the design philosophy.

During this project, three extensions for PLASIMO have been written, with the purpose to make it possible to let PLASIMO handle high-power density plasmas. These are:

- A program that uses Disturbed Bilateral Relations to make an approximate calculation of various plasma parameters has been written. This makes it possible to generate starting conditions for PLASIMO, thus facilitating the use of PLASIMO. PLASIMO, like any iterative model, needs values to start the iterative procedure. If these values are not sufficiently close to the actual solution, poor convergence or even divergence may result.
- A plug-in that generalizes the calculation of the diffusion coefficient for both high-pressure and low-pressure regimes, making it possible to calculate diffusion coefficients in plasmas where the fact that the mean free path of the particles is larger than the typical system dimensions makes it impossible to use standard diffusion theory.
- A plug-in that calculates the Lorentz force, essential for the treatment of pinch plasmas, has been written.

These extensions will be discussed in detail in this report.

1.3.1 The hollow cathode discharge

A case study on a special case of the hollow cathode discharge, the hollow cathode discharge, has been carried out. In particular, the decay of the plasma in the hollow cathode after the pinch has been studied. Also, the first steps towards modeling the pinch are investigated.

1.4 The contents of this report

Chapter 2 gives a short introduction in the operation of the hollow cathode discharge. This forms the basis of the further studies on the hollow cathode discharge. In this chapter, the various stages in the operation of the hollow cathode discharge will be identified. Two of these stages will be the subject of further investigation.

In chapter 3, a general study of PLASIMO will be conducted. First, the general transport equation, the so-called ϕ-equation [3], which lies at the heart of PLASIMO, is discussed. Then, a case study is made from a few very simple problems, as an example and as a validation of the code.

Chapter 4 deals with the creation, implementation and validation of a simple 0-D model plasma model, based on the method of Disturbed Bilateral Relation [4]. The purpose of this model is to complement the complicated and sophisticated PLASIMO modeling toolkit with a simple, crude model. This model can also be highly useful for supplying reasonable estimates of key plasma parameters. As stated above, an iterative procedure, such as a PLASIMO model, needs a reasonable estimate of these parameters to be able to find a converged solution of the model.

Chapter 5 is a study of the consequences of going from a NLTE[2] model to an LTE[3] model. The results will be discussed with the ϕ-equation and the Disturbed Bilateral Relations in mind.

Chapter 6 deals with the implementation of a module that can calculate the Lorentz force, a key step in modeling pinch plasmas, where the pinching force is delivered by the Lorentz force. As will be shown, applying this module to the hollow cathode discharge will require other modifications to the code as well.

Chapter 7 deals with the implementation of a new way of calculating the diffusion coefficient. By taking into account both normal diffusion and Knudsen flow, the diffusion coefficient can be calculated for much lower pressures than if one would use only normal diffusion. This makes it possible to model the decay of the plasma in the hollow cathode. Calculations on this decay have been performed and are presented as well in that chapter.

Chapter 8 presents the conclusions of this work.

[2]Non Local Thermal Equilibrium: see chapter 3
[3]Local Thermal Equilibrium: see chapter 3

Some sections are marked with a*. In these sections, the program code is discussed. While this may be very interesting for someone involved in numerics, for others it may not be interesting, as no physics is discussed in these chapters. They may be skipped without significantly interrupting the argument.

Chapter 2

Introduction to Hollow Cathode Discharges

2.1 Introduction

Hollow cathode pseudospark discharges discharges have first been described in the 1950's. Then, the pseudospark discharge was an unwanted, and unexpected, side effect. Nowadays, it is a device in which a lot of research effort is invested. Applications include high-voltage switches and spectroscopy.

In this chapter, a new application of the hollow cathode will be discussed: its potential use in EUV[1] lithography, a necessary step in the production of ever faster computers. The discussion will be focused on hollow cathode discharges with properties that are typically used in the EUV research. In order to understand the requirements the hollow cathode must satisfy, the role of the hollow cathode in future-generation processor manufacturing processes will be discussed in section 2.2. After this discussion, a schematic design of the hollow cathode will be presented in section 2.3. Using this design, the operation of the hollow cathode will be discussed in section 2.4. A short summary will be presented in section 2.5.

2.2 Hollow cathodes in EUV lithography

As stated, hollow cathodes are being investigated as a source of EUV generation. A possible use of this radiation is water window microscopy. Here, the focus will be on another application of EUV generation, namely EUV lithography. Lithography is a key step in the production of computer chips. The motivations for researching EUV lithography will be discussed in this section.

Without going into the details of lithography, it can be stated that a light beam, that has gone trough a very complex optical system, is used to image the structures on the chip. This starts to cause problems when the size of the structures becomes significantly (about two to three times times) smaller than

[1]EUV stands for Extreme Ultra-Violet. This is a term for light in a band around 10 nm.

the wavelength of the used light. Nowadays[2], the size of the structures on a standard processor has decreased to 130 nm, and for other computer chips, such as RAM, system logic, and graphics processors, this figure is in the same range. Further reduction of the size of the structures, important for producing better computer chips, will require light with a shorter wavelength.

When the wavelength drops to less than 150 nm, almost all materials have a very high absorption. It is thus necessary to use mirrors. However, even that becomes troublesome. A frequency window in which “good” (about 70% reflectivity) mirrors are available is a frequency window at 13.4 nm. If this window is to be used for lithography, a powerful light source, that can produce up to 100 W in a narrow band near 13.4 nm, is required. The hollow cathode discharge is a candidate for this light source.

The hollow cathode discharge works in a pulsed mode. The decay time of the plasma after the pinch (see section 2.4) may be limiting the repetition frequency. Minimizing the decay time is a way in which the repetition frequency, and thus the power output, can be boosted. By creating a model that can predict the decay time for given parameters, such as geometry, it will become possible to determine the effect of a change in the geometry on the decay time, and thus the repetition frequency and the output power.

2.3 Design

As stated in the introduction, there are many possible designs for a hollow cathode discharge. For a more thorough explanation, one might look at [5, 6, 7], An example is given in figure 2.1. Here, a schematic drawing of a hollow cathode discharge is given. The cathode is generally a few centimeters wide and high.

2.3.1 Gas considerations

Various gasses can be used to operate the discharge, however, for EUV generation, three gasses are considered: xenon, lithium and tin. Each has advantages and disadvantages, which will be discussed.

- Xenon: Strongly ionized xenon (Xe^{8+} to Xe^{14+}) can emit light in a band around 13 nm. However, the fraction of the light emitted in this band is rather low. It has, however, technological advantages, in particular the fact that it is a noble gas. This means, that it does not react easily with surfaces, and will not damage the mirrors as much. Sputtering and ion implantation can still damage the mirrors, though.

- Lithium: Li^{2+} can also emit light in the 13 nm region. It is more efficient than xenon, but has the unfortunate disadvantage of being extremely reactive. Therefore, it is a technological challenge to prevent the discharge from emitting large quantities of lithium. Lithium would cause damage to mirrors if it comes into contact with them, because of deposition and surface reactions.

[2]August 2002

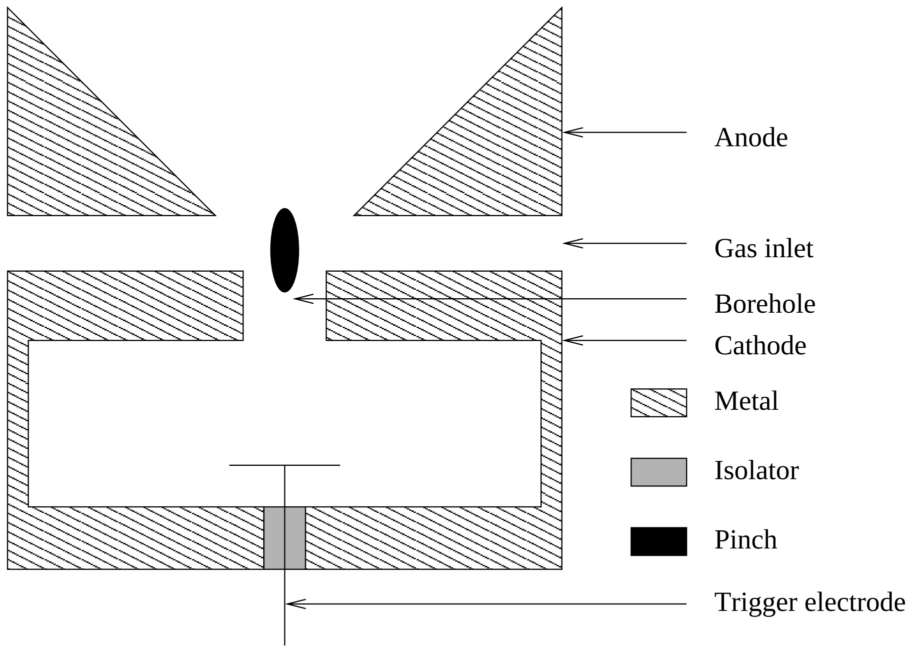

Figure 2.1: A schematic drawing of a cross section of the hollow cathode design. it has a cylindrical symmetry. The device is a few centimers in diameter. The actual geometry is varied to obtain an optimal design.

- Tin: Tin also is more efficient than xenon, but is also not chemically inert.

2.4 Operation

The hollow cathode does not operate in a steady-state mode, but rather in a continuing cycle. At the beginning of the cycle, the anode plate is at a voltage that is much higher than the cathode plate (a few kV). The trigger electrode is at a potential that is a few hundreds of volts higher than the cathode. This means, that electrons are rapidly drawn to the cathode, and this means that there are too few electrons to initiate the discharge. The trigger electrode functions as an inhibitor.

When the trigger potential is removed, the free electrons will not be removed anymore. Then, a Townsend avalanche will start. The electrons produced in this way get absorbed by the anode. Furthermore, the ions will get drawn to the cathode, where they produce electrons through secondary emission. This will produce secondary electrons, that will cause some ionizations in the cathode. The ionization rate is low in this stage, because the mean free path of the electons is larger than the cathode dimensions. The electrons will be rapidly leaving the more massive ions behind. This causes a positive space charge to form near the borehole. This causes the anode potential to penetrate into the hollow cathode. A virtual anode forms into the hollow cathode. This causes the largest part of the potential drop to occur in a small region near the cathode wall. This creates an electric field that traps the electrons.

The trapped electrons will start to move around in the cathode, ionizing neutral atoms. High-energy electrons, which have a low ionization cross section, do not cause much ionization. The reflections in the sheath causing the so-called pendulum effect, are needed to produce high levels of ionization. The electron density rapidly rises, while the plasma conductivity rapidly increases. This causes a very large (kA range) current to flow from the hollow cathode to the anode. Such a massive current trough a small borehole will cause the Lorentz force to significantly influence the movement of the electrons. This will cause the current to contract, further increasing the Lorentz force, creating a pinch plasma. This pinch has such a high energy dissipation density, that the particles in it can be strongly ionized. Some gasses, like the aforementioned lithium, tin and xenon, can emit light in the EUV range.

During the pinch, a lot of current is drawn. The voltage difference between cathode and anode, which is sustained by a capacitor, gets smaller as the capacitor is drained of charge. The pinch is not stable, and cannot be sustained in steady state. The pinch will stop, and the voltage difference between anode and cathode will be small. After that, the cathode is still filled with electrons and ions. This plasma must decay before the next shot. Because there is no energy source, the diffusion losses will not be replenished, and the plasma will decay. Applying a positive voltage to the trigger electrode may accelerate this effect. When the plasma has decayed sufficiently[3], one can start recharging for the next shot.

2.5 Summary

In this chapter, a global overview of the design and operation of a hollow cathode pseudospark discharge has been given. The discussion has been focused on a particular use of the hollow cathode discharge, namely its potential use as a high-power EUV source. A more detailed discussion, using a model with realistic parameters, will be discussed in the following chapters.

It is an objective to be able to simulate all the stages of the hollow cathode discharge. PLASIMO was chosen as the modeling toolkit to use for this project. Not only does PLASIMO already have much of the physics on board, it can also create models that are easily adapted. This means, that only the physics that are important during a certain stage need to be incorporated. Also, the user-friendly GUI and sophisticated output make it attractive to use PLASIMO. While not all the physics necessary to simulate the hollow cathode is in PLASIMO, its easy extensibility makes it possible to incorporate the missing physics. Indeed, a large part of this report discusses the creation of new modules that describe physics necessary to create a realistic model of the hollow cathode discharge, such as pinching (Chapter 6) and Knudsen flow (Chapter 7).

[3]If there are too many electrons, the plasma will break down before the voltage is high enough.

Chapter 3

A brief tour of PLASIMO

3.1 Introduction

PLASIMO is a general toolkit that can be used to generate models of plasmas [8, 9, 10]. In this chapter, a general overview of the implementation of the physics in PLASIMO will be given, followed by a demonstration of PLASIMO calculations on a few simple systems. Central in this discussion is the canonical transport equation, usually denoted by ϕ-equation.

3.2 Transport physics in PLASIMO

In fluid mechanics, or in a plasma, which is by definition a non-equilibrium system, transport is of paramount importance. The important role of transport in a plasma is respected in PLASIMO by representing physical quantities as special cases of a general conservation equation, in which transport plays a key role: the ϕ-equation. With this approach, we use the fact that the transport of different variables can be described in a similar fashion.

3.2.1 The ϕ-equation

Patankar [3] defines the ϕ-equation as:

$$\frac{\partial \rho\phi}{\partial t} + \vec{\nabla} \cdot (\rho\vec{v}\phi) = \vec{\nabla} \cdot (\Gamma\vec{\nabla}\phi) + S \tag{3.1}$$

Here, S is the source term, ρ the density, Γ a generalized diffusion coefficient, $\vec{v}$ the velocity, and ϕ is a specific[1] physical property, for which conservation laws apply. Specific physical properties are represented in this way in PLASIMO. Basically, the ϕ-equation is nothing more than a general conservation equation.

PLASIMO uses the control volume method to discretize the equation. In this method, the equation is discretized on a mesh of control volumes. For one such volume, the integral conservation laws are exactly[2] satisfied, even for a coarse

[1]per mass unit

[2]In practice, up to machine precision

grid. As the complexity of the problem often makes it necessary to use fairly coarse grids[3], this is an important feature. In a control volume, the values of ϕ are stored in the nodal points. These are points which generally lie at the center of the control volumes. The value of the ϕ in the nodal point is assumed to be valid troughout the control volume. Note that the ϕ-equation in this form implicitly assumes a fluid[4]; a way to also treat some non-fluid systems is discussed in chapter 7. The ϕ-related transport (the convection and diffusion terms) are dealt with at the control volume boundaries.

3.2.2 Examples of the ϕ-equation

In this section, various examples of the use of the general ϕ-variable to represent physical properties will be given. As examples, the specific momentum (momentum per mass unit) and enthalpy will be discussed.

The specific momentum equation

The momentum equation in the x-direction is given by [9]:

$$\frac{\partial \rho v_x}{\partial t} + \vec{\nabla} \cdot (\rho \vec{v} v_x) = \vec{\nabla} \cdot (\mu \vec{\nabla} v) + F_x \tag{3.2}$$

Here, v_x is the speed in the x-direction, μ is the viscosity and F are forces. Comparing this with (3.1), the ϕ can be identified with v_x, Γ with μ and S with F_x. The other velocities can be treated similarly.

The enthalpy equation

In this example, a simplified enthalpy equation [3] for ideal gases will be discussed.

$$\frac{\partial \rho h}{\partial t} + \vec{\nabla} \cdot (\rho \vec{v} h) = \vec{\nabla} \cdot (\frac{k}{c} \vec{\nabla} h) + S_h \tag{3.3}$$

Here, h is the specific enthalpy, k the thermal conductivity, c is the constant-pressure specific heat and S_h is the volumetric rate of heat generation. Comparing this with (3.1), the ϕ can be identified with h, Γ with $\frac{k}{c}$ and S with S_h.

3.2.3 The ϕ-equation in PLASIMO

In PLASIMO, a physical property is represented with a ϕ when this is appropriate. Depending on the properties of the plasma under study which are implemented in the model, PLASIMO generates different sets of ϕ-equations to represent them. These are subsequently solved iteratively, until a converged solution is obtained. Different problems require different sets of ϕ-equations. It is the task of PLASIMO to to assist the developer to assemble the appropriate set of ϕ-equations, and to solve the set of differential equations.

[3]40 grid points in each dimension is typical

[4]This means, that the dimensions of the system are small compared to the mean free path of the molecules

3.2.4 Increasingly complex systems

In this sections, we will deal with a number of increasingly difficult problems and the corresponding larger set of ϕ-equations. Starting from a simple heat transport problem, restrictions will be released until the almost totally free Non-Local Thermal Equilibrium (NLTE) plasma is reached. The most important ϕ-equations that describe the model are discussed.

Heat conductivity

The heat transfer problem in a stationary medium of uniform density can be described by a simplified version of equation (3.3), noting that

$$\vec{\nabla} h = c\vec{\nabla} T \tag{3.4}$$

With (3.4) and the other assumptions, (3.3) becomes

$$\vec{\nabla}(k\vec{\nabla} T) + S_h = 0 \tag{3.5}$$

This equation describes heat transport. This equation, or its more general form (3.3), are often part of more complex calculations. Note that specific enthalpy is a specific quantity, while the temperature is not. This equation can be used to describe the heat transfer in an uniform bar.

Fluid flow

In this example, a constant-density (low Mach number) fluid flow is treated. The problem is described by the mass conservation equation and the momentum conservation equations. The mass conservation equation is given by

$$\vec{\nabla} \cdot (\rho\vec{v}) = 0 \tag{3.6}$$

The momentum conservation equation for each direction is given by (3.2). In this equation, the forces term F_x can be rewritten, and the equation becomes:

$$\frac{\partial \rho v_x}{\partial t} + \vec{\nabla} \cdot (\rho\vec{v} v_x) = \vec{\nabla} \cdot (D\vec{\nabla} v) - \frac{\partial p}{\partial x} + B_x \tag{3.7}$$

with p the pressure and B_x the other forces. This formulation makes the important role of the pressure gradient explicit.

PLASIMO currently uses the SIMPLE algorithm [3] to solve these equations[5] and obtain the pressure and the flow field. The SIMPLE algorithm basically works like this [9]:

- From the initial guess of the pressure field the velocities are calculated using equation (3.7)
- This velocity field will generally not satisfy (3.6). This equation is used to obtain a correction for the velocity and pressure fields. The equation is rewritten as a pressure-correction equation, which is treated as a ϕ-equation

[5]It is planned to add the more general and more robust SIMPLER algorithm [11] as a flow solver

- If present, other ϕ-equations are solved, and if necessary, transport coefficients such as the diffusion coefficient are updated.
- This procedure is repeated until convergence is achieved.

A very simple variety of this problem is a fluid with constant values of μ and ρ. The energy equation is not solved; the only ϕ-equations are the momentum equation (3.7) and the continuity equation (3.6).

A LTE plasma

The abbreviation LTE stands for Local Thermal Equilibrium [12], meaning that all the species in the plasma have (approximately) the same temperature, and that the chemical processes which create and destroy particles are so fast that the densities of species can be given by analytical relations, based on the standard laws of statistical mechanics. Apart from these relations, the plasma is described by a set of ϕ-equations consisting of the energy equation (3.5), the momentum equation (3.7) and the continuity equation (3.6).

In an LTE plasma, there is much more freedom than in a classical fluid. The chemical processes may produce a very wide variety of particles, each with different transport properties. This composition generally depends nonlinearly on temperature; thus, it is necessary to take into account the energy equation to calculate the temperature. This also means that the density is not uniform, as the chemical composition is temperature-dependent and the temperature is generally not uniform due to contact with the wall.

What makes this problem particularly challenging is that various ϕ-equations may influence each other. For instance, the temperature may influence the composition, which influences the density distribution, which influences the velocity field. However, the velocity influences the energy transport, which influences the temperature, closing the circle. The strong influence of the ϕ-equations on each other and the general nonlinearity of the dependencies make it necessary to use an iterative procedure to solve the ϕ-equations.

A NLTE plasma

In many plasmas, the assumption that the energy transfer between species keeps them at the same temperature is not valid. In PLASIMO, we restrict ourselves to a single electron temperature and a single heavy particle temperature. These two temperatures are most likely to differ significantly because the energy transfer between heavy particles and electrons is relatively inefficient, due to the large difference in mass. This means that instead of one ϕ-equation, we now need two ϕ-equations for the temperature. In the NLTE approach in PLASIMO, there is in fact always a temperature difference between heavy particles and electrons. This is because only the electrons receive the power that sustains the plasma. Because the energy transfer from the electrons to the heavy particles requires a temperature difference, the electrons will always be hotter in this approach[6].

[6]This is not a fundamental restriction, and if necessary, it is certainly possible to also couple power into the ions

The chemical reactions may also not be frequent enough to compensate for other loss and creation processes of particles, thus making it impossible to use the analytical expressions to determine the composition. Chapter 4 deals with this in depth. In this case, the transport for the individual species becomes important. This means, that the for each species, the creation and destruction of this species by chemical processes has to be taken into account, and that the transport also has to be described. This transport is described by a ϕ-equation. This greatly increases the number of equations.

3.3 Examples of PLASIMO models

It is one of the aims of this study to test PLASIMO with a series of models with increasing complexity. First, simple fluid flows will be studied. A model of a duct flow and a model of pipe flow will be created, executed, and the results will be compared with the analytical solutions. After this, an argon LTE plasma, which has more degrees of freedom, will be studied. A study of the even more complex Ar NLTE plasma will be postponed until chapter 5, in order to first introduce the reader to the theory about Disturbed Bilateral Relations in chapter 4, which will facilitate the understanding of the results.

3.3.1 Fluid Flow

It is possible to consider fluids as special plasmas, namely those in which the ionization degree is zero, the chemical composition is constant and the transport properties are constant. PLASIMO is indeed able to simulate these "plasmas". In this section, three different flows will be discussed: a simple duct flow, a simple pipe flow and a barometric flow. In all of these simulations, the fluid simulated is water with its viscosity ν at room temperature of 10^{-3} Pa s [13] and its density ρ of 1000 kg m^{-3}. Furthermore, there is a no-slip condition at the wall, meaning that $\vec{v}_{wall} = \vec{0}$.

While PLASIMO uses generalized coordinates, presently only 2D coordinates are used. The generalized coordinate system means, that it is possible to use many different 2D systems, such as cartesian and cylinder-symmetric systems. With relatively little effort, other 2D coordinate systems can be added. Even 1D and 3D coordinate systems can be used while maintaining large parts of the code. This study will be restricted to straight cartesian and cylindrical grids, to make it possible to compare the results with the results on an analytical model.

Duct Flow

PLASIMO has been used to create a model that simulates simple duct flows. The simulated problem is the classical Poiseuille problem, presented in Figure 3.1. Because turbulence is not yet implemented in PLASIMO, only viscous flows will be investigated. The following key parameters were used for this simulation:

- The piece of duct is 2D. It is assumed that for all variables, the derivative in the x-direction is 0.

- A piece of duct with a length L_z of 1 meter and a height L_y of 1 meter is simulated.
- The upper (y= 1 m) and lower (y= 0 m) boundaries are walls, meaning that $\vec{v}(z,0) = \vec{v}(z,L_y) = \vec{0}$. This is the no-slip condition.
- The pressure gradient is -0.001 Pa m^{-1} in the z-direction. There is no pressure gradient in the y-direction
- The left (z=0 m) and right (z=1 m) boundaries are open and are chosen in a way that this piece of duct can be considered as a part of a very long duct, in which the flow is totally developed.

Running a model with these parameters yield the velocity profile for v_z of figure 3.2. v_y turns out to be 0.

For this problem, the exact analytical solution exist. The problem is described by the following differential equations:

$$\frac{\partial v_z}{\partial z} = 0 \tag{3.8}$$

$$\frac{\partial p}{\partial z} = \nu \frac{\partial^2 v_z}{\partial y^2} \tag{3.9}$$

$$v_z|_{y=0} = v_z|_{y=1} = 0 \tag{3.10}$$

Equation (3.8) is the mass conservation law, with v_z the speed in the z-direction. Equation (3.9) is the equation of momentum conservation in the z-direction, with p the pressure. Equation (3.10) defines the boundary conditions

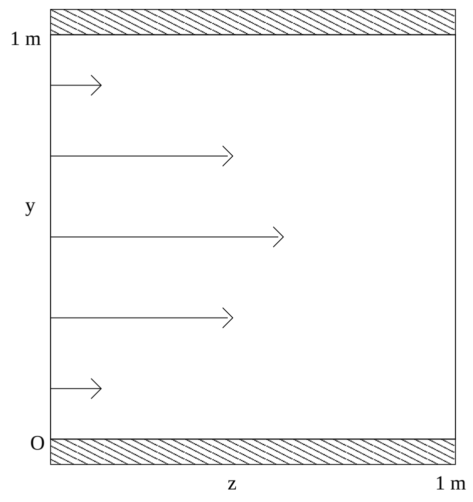

Figure 3.1: A sketch of the computational domain of the duct flow problem, with a sketch of a fully developed Hagen-Poiseuille flow profile. The flow is assumed to be uniform in the x-direction, which is perpendicular to the page.

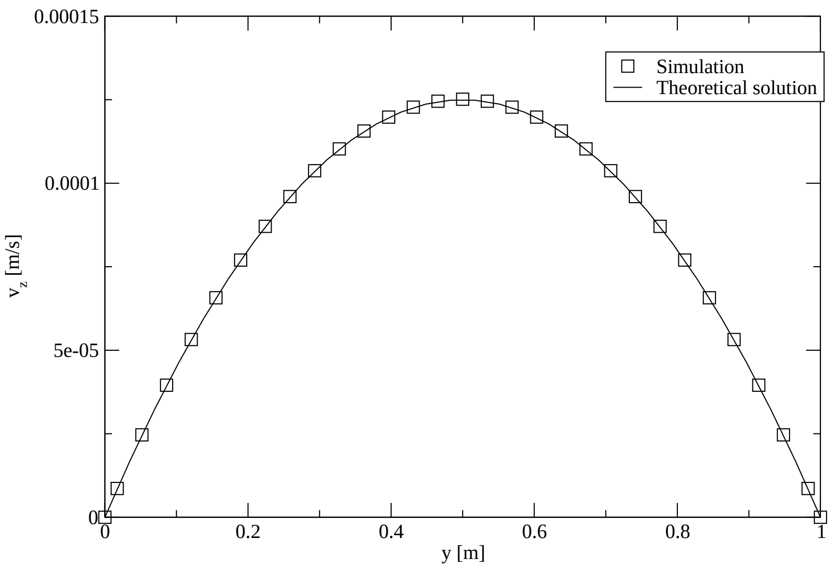

Figure 3.2: The velocity profile of the duct flow.

at the south boundary $z = 0$ and the north boundary $z = 1$. The solution of this set of equations is a parabolic profile, given by:

$$v_z(y, z) = \frac{\partial p}{\partial z} \frac{(0 - L_y)^2}{8\nu} \left(1 - \left(\frac{2y - 0 - L_y}{0 - L_y}\right)^2\right) \tag{3.11}$$

for $0 < x < 1$. Equation (3.11) is also plotted in Figure 3.2, with appropriate numerical values substituted. Figure 3.2 compares (3.11) with the numerical results. The theoretical and simulated curves match excellently. The model converges in about 2600 iterations to a residual of $1{\cdot}10^{-11}$.

Pipe flow

PLASIMO has been used to model the flow of a fluid in a pipe. The problem is quite similar to the duct flow problem described in section 3.3.1: the only differences are the cylindrical geometry and the different pressure gradient. The duct is replaced by a tube that has a radius r of 0.01 m and a length L_z of 1 m. The pressure gradient is -0.01 Pa m^{-1}. Water is used, with a viscosity of 10^{-3} Pa s [13]. The results are given in figure 3.3.

It is also possible to give an exact, analytical solution for this flow problem. The equations that describe the system are:

$$\frac{\partial r v_z}{\partial z} = 0 \tag{3.12}$$

$$\frac{\partial p}{\partial z} = \nu \frac{1}{r} \frac{\partial}{\partial r}\left(r \frac{\partial v_z}{\partial r}\right) \tag{3.13}$$

$$\frac{\partial v_z}{\partial r}|_{r=0} = 0; v_z|_{r=0.01} = 0 \tag{3.14}$$

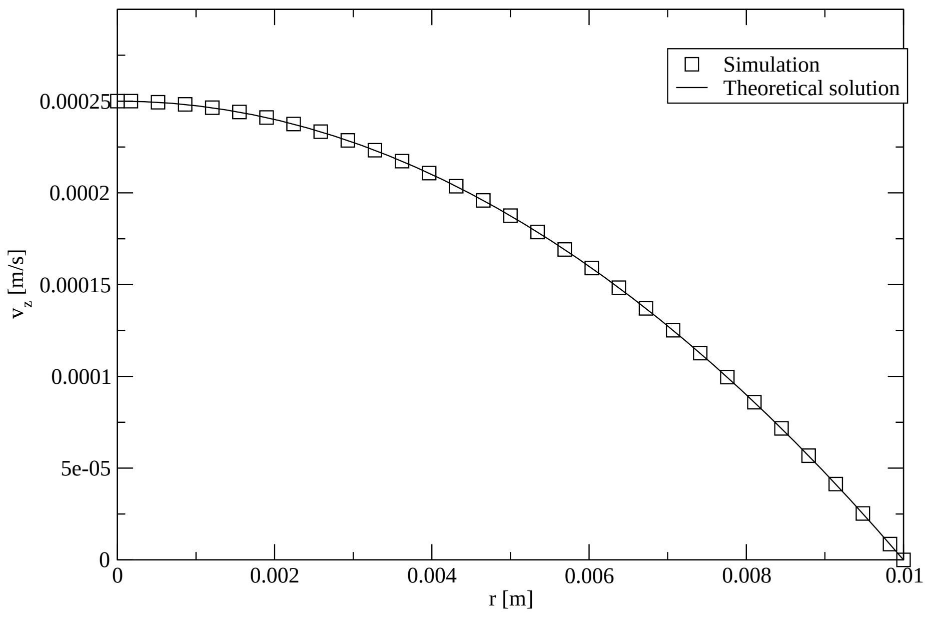

Figure 3.3: The velocity profile of the pipe flow

These equations can be solved analytically, giving a parabolic velocity profile. This analytical solution is given in figure 3.3; it matches the simulated solution very well. The model converges in 516 iterations to a residual of $1{\cdot}10^{-11}$.

Barometric Flow

PLASIMO is able to simulate the effects of gravity. This ability will be investigated on a duct flow problem similar to the problem presented in section 3.3.1.

The simulated fluid is water. The flow is driven by gravity and not by a pressure gradient as in the preceding examples. The gravitational acceleration g is 9.81 m s^{-2}. The duct has to be much smaller in this case to make sure that the flow is not turbulent. Therefore a width of 1 mm is selected. The solution of this problem, calculated with PLASIMO, can be found in figure 3.4.

The analytical solution can be calculated from the differential equations that describe the problem. These are equation (3.8), equation (3.10) and equation (3.15):

$$-g = \nu \frac{\partial^2 v_z}{\partial y^2} \tag{3.15}$$

The solution of this equation, a parabolic profile, matches with the PLASIMO result of figure 3.4.

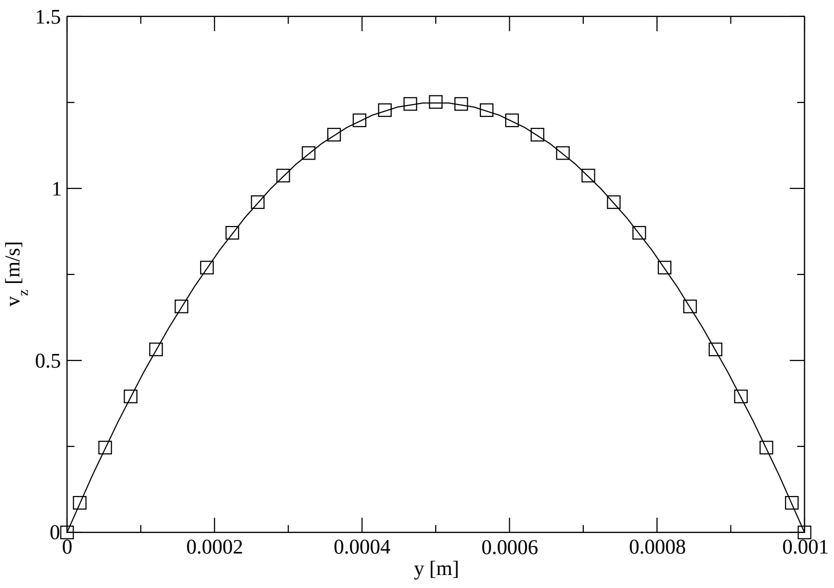

Figure 3.4: The velocity profile of the barometric duct flow.

Concluding remark

The astute reader will have noticed that the velocities in figure 3.2 and 3.3 are low. These PLASIMO models failed to converge for systems settings that had higher velocities. The cause is thought to lie in the value of the Reynolds number Re. This is defined by

$$Re = \frac{vL}{\nu} \tag{3.16}$$

with L the typical length scale of the problem. When this number comes above 2300, the fluid stops being laminar, but becomes turbulent, and the Navier-Stokes equations, in the form used by the flow solver in PLASIMO, no longer apply. For the flow problems in section 3.3.1 and 3.3.1, Re equals 1000. This means, that the system is close to the transition from laminar to turbulent. Thus, the fact that PLASIMO apparently cannot handle these high Reynolds number problems is not a big restriction, as the results would be inaccurate anyway, due to the inapplicability of the set of equations used to describe the problem. In the future, a $k - \epsilon$ model may be added, which extends the flow code in a way that makes it capable of handling turbulent flows. Such a model can also be described with the ϕ-equation method.

It is worth noting that not all PLASIMO models have problems for this range of Reynolds numbers. Models where instead of the inlet pressure and outlet pressure the inlet flow and outlet pressure are specified converge much better. It is thought that the lack of numerical "freedom" in the former case negatively impacts the stability.

3.3.2 An Ar LTE plasma

The previous examples dealt with simple fluids. This means, that in contrast to plasma the degree of ionization us zero, while the chemical composition is uniform. Other chemical reactions also influence the plasma composition. In LTE, however, transitions are balanced by their inverse process. Radiation is free to escape, but this transport phenomenon does not disturb the forward-backward balances significantly. This means, that the temperature of all species is equal, and that the composition is ruled by equilibrium relations, such as the Saha relation (3.17) for ionization and the Guldberg-Waage equation for dissociation. These relations reduce the number of degrees of freedom considerably.

The case of NLTE has a virtually unlimited number of degrees of freedom. However, departure from LTE moves along stages. For instance, thermal equilibrium, which means that the electrons and heavy particles have the same temperature, may be maintained, while the chemical equilibrium, which means that the degree of ionization and dissociation obey the Saha and Guldberg-Waage equation, may not be maintained [12].

In practice, there are always small disturbances to this picture. However, when these disturbances does not significantly influence the state densities, LTE can be a good and relatively simple approximation. Chapter 4 deals with these disturbances in detail.

As stated, LTE plasmas have more degrees of freedom than fluids, but less than NLTE plasmas. PLASIMO can be instructed to use different solving techniques for simple fluid flows, LTE plasmas and NLTE plasmas. This enables PLASIMO to ignore irrelevant degrees of freedom, which would cost computing time and may even cause a failure to converge. Chapter 5 deals with this in detail.

The plasma under investigation here is a segment of a non-flowing Ar plasma. Only the ground state and the ion state are taken into account. This oversimplification is made because the purpose of this investigation is a study of PLASIMOs behavior and not that of Ar plasmas. The segment has a radius of 0.25 m and a length of 0.1 m. This gives the plasma a volume of 0.020 m^{-3}. The wall is kept at 300 K. This provides the cooling. Radiation is not taken into account. A power of 5000 W is coupled into the vessel by an uniform electric field in the z-direction, which results in a power density of $2.5 \cdot 10^6$ Wm^{-3} The amount of Ar is set by filling it with 1000 Pa Ar at 273K. This gives an initial neutral density of $2.65 \cdot 10^{23}$ m^{-3}. The total number of Ar atoms is thus $5.3 \cdot 10^{21}$. The pressure at plasma conditions is about 0.3 bar. The electron density n_e as predicted by PLASIMO is compared in figure 3.5, with the value of n_e calculated with the Saha equation:

$$n_p = \frac{n_e n_i}{2g} \left(\frac{h^2}{2\pi m_e k_b T} \right)^{\frac{3}{2}} \exp \left(\frac{E}{k_b T} \right) \tag{3.17}$$

Here, n_p is the neutral density, n_i the electron density, g the degeneracy of the ion state, h Planck's constant, m_e the electron mass, k_b Boltzmann's con-

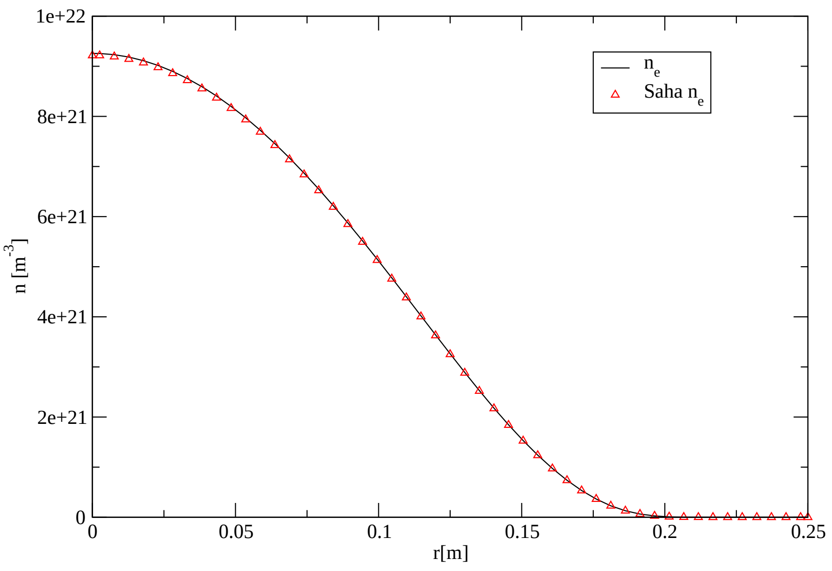

Figure 3.5: The electron density of the Ar plasma, calculated with the LTE module. It is compared with the Saha density of the electrons, that has been derived from the temperature and neutral density calculated by PLASIMO.

stant, and E the energy level of the ion state. The temperature profile of this simulation can be found in figure 3.6.

The temperature profile has a maximum at about 16000K and decreases rather sharply near the wall. n_e is close to the Saha density, which is not surprising, because PLASIMO uses the Saha formula to calculate the electron density. Note that the ionization is small near the wall. This means that the heat conductivity of the electrons becomes very small. This causes a significant drop in the overall heat conductivity. Thus, it requires a larger temperature gradient to transport the heat to the wall.

The low electron density is caused by a low temperature. In a NLTE plasma, the electron temperature could drop less steeply near the wall than the heavy particle temperature. This higher electron temperature could keep the ionization higher. By using the LTE module, this is made impossible. Therefore, the results of the LTE calculation are not correct near the wall. This may influence the results in other parts of the plasma. This can be investigated by using non-LTE models, which will be discussed in chapter 5.

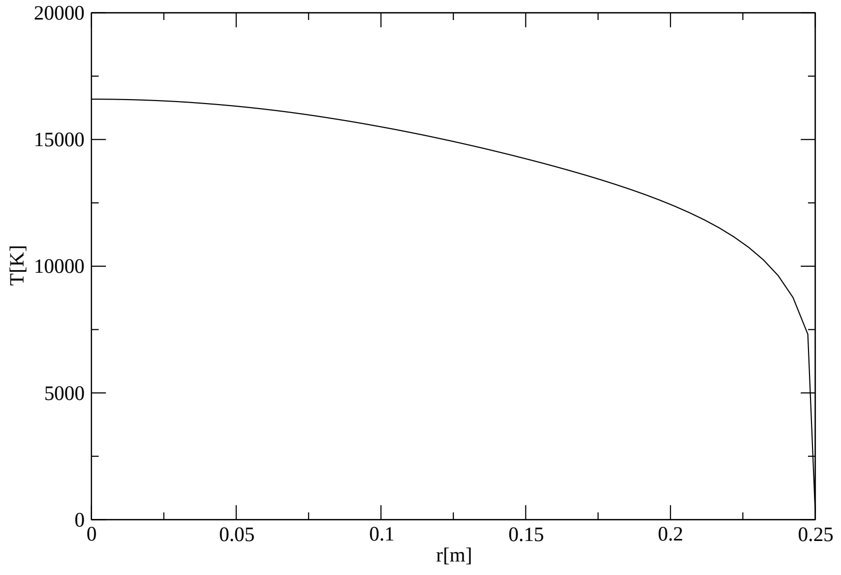

Figure 3.6: The temperature profiles of an Ar plasma, calculated with the LTE module.

Chapter 4

Creating a 0-D model using Disturbed Bilateral Relations

4.1 Introduction

In order to start an iterative model, such as a PLASIMO model, one generally needs a starting condition. Usually, the closer the starting condition resembles the results of the calculation, the better, as the calculation will need less iterations to find a self-consistent solution. Failure to provide a starting condition that is sufficiently close to the results will cause slower, more irregular convergence and may even result in divergence. For this reason, a program has been written that can approximate the electron density n_e, the neutral density n_a, the electron temperature T_e, the heavy particle temperature T_h and the pressure p for a given power, heavy particle density n, geometry and physical and chemical properties of the main plasma constituent[1]. The latter are used to determine the transport coefficients using constitutive equations. This approximate calculation is based on 3 conservation equations: the electron particle balance, the electron energy balance and the heavy particle energy balance. It is very important to note that it is not accuracy, but stability and versatility that are the main design parameters. The model will, however, be restricted to atomic plasmas, and single-ionized ions. These limitations are not fundamental, and it is possible to lift them in the future.

The usefulness of the 0-D program, however, is not limited to a start-up module of PLASIMO. As it contains much fundamental plasma physics, it can be used to predict certain scaling laws, the relative importance of various energy loss processes, equilibrium validity criteria and stability criteria of plasmas. The approach is based on the concept of Disturbed Bilateral Relations and will therefore be named as such.

[1]The atomic mass, rate coefficient for ionization, temperature exponent in the Arrhenius equation, energy level of the ion ground state, degeneracy of the ion ground state, and the cross sections for hard-sphere, ion-neutral and ion-electron collision

4.2 Disturbed Bilateral Relations

The method of Disturbed Bilateral Relations (DBR) is thoroughly explained in [4]. A thorough explanation is beyond the scope of this study. A brief explanation is offered below. In this report, we use three of them:the electron particle balance, the electron energy balance and the heavy particle energy balance.

In literature, global models can be found that are similiar to the one presented here; Lieberman [14] decribes a model that is suitable for systems that are far from equilibrium. In these models, only the transport (the improper term) and not the inverse process (the proper term) are taken into account. In our model, the proper terms are taken into account too. While the DBR model is designed for systems that are far from equilibrium, it thus can also be used for systems that come close to LTE. It can be used to explore the road from NLTE to LTE.

The reason for the interest in plasmas that are far from equilibrium is that plasmas are created to get something out of them, usually light or particles. This implies that transport is an important aspect. But transport causes deviation from equilibrium. However, for a steady state, this means that there has to be input. On the other hand, as the reaction(s) that process the input to the desired output always have inverse processes, there are also processes which restore equilibrium. A way of looking at a plasma is considering it a set of disturbed bilateral relations between states or energy levels, where transport and equilibrium compete. A very schematic picture is in figure 4.1 For some states, the transport is dominant, and their densities are determined by it. For others, the transport is small, and they are in equilibrium. This can be described by

$$P - \phi D = T\phi \tag{4.1}$$

Here, T represents transport, P represents production and D destruction.

Disturbed Bilateral Relations can be used to determine which states can be described by equilibrium relations, for instance which species have the same kinetic temperature and which energy levels are populated according to the Saha equation. Here, it will be used to find scaling laws and starting conditions. This will yield a useful tool to characterize plasmas.

4.2.1 The electron particle balance

The electron particle balance is a balance that equates the production and the loss of electrons. In a simple form it reads

$$n_p n_e k_{ion} - n_e^3 k_{rec} = \frac{n_e D^*}{n_p \Lambda^2} \tag{4.2}$$

with k_{ion} the ionization rate coefficient and k_{rec} the three-particle recombination rate and D^* is equal to Dn_p, with D the normal diffusion coefficient. The diffusion length Λ is usually given by half of the smallest dimension of the plasma. For instance, in a typical TL tube, the diffusion length should be equal to half the diameter of the tube. Recombination, the second term in (4.2), is the proper "loss" channel, as it is the inverse process of ionization, the first term in

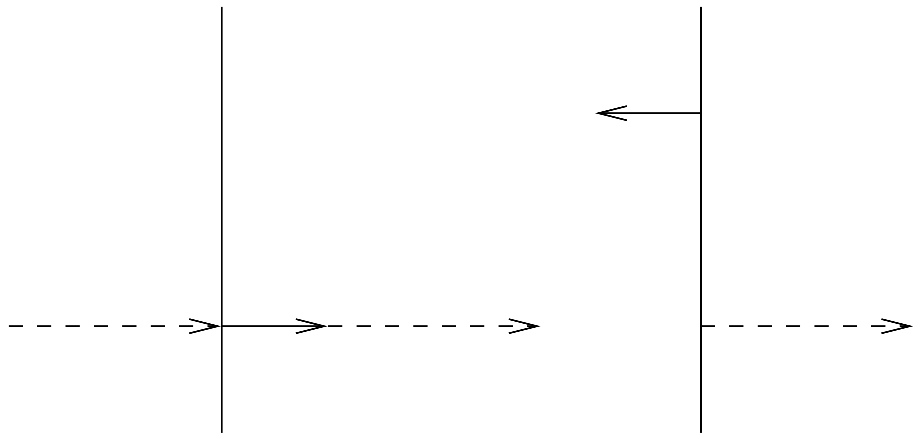

Figure 4.1: A schematic overview of a disturbed bilateral relation. The solid arrows represent the equilibrium processes, while the dashed arrows refer to the transport induced disturbances.

(4.2), while the spatial transport term, the third term in (4.2), is a disturbance. Note the similarity with (4.1).

In equation (4.2) it is assumed that the only production process is ionization of neutrals by electrons. This is a temperature dependent process, because the k_{ion} is generally strongly temperature dependent[2]. This process is balanced by the loss processes.

Equation (4.2) applies to the central plasma. However, there is a close connection with the outer plasma region, more specifically the wall, where recombination takes place. Because of this, a concentration gradient exists. This causes diffusion of electron an ion pairs from the central plasma to the wall. This effect is especially important in relatively small plasmas that are not very dense. In these plasma, T_e is largely determined by the condition that it must be high enough to provide sufficient ionization in order to compensate for the diffusion losses, otherwise, the plasma would extinguish.

The second loss process, three-particle recombination, is usually insignificant unless the electron density gets very large (typically above 10^{21} m^{-3}). If three-particle recombination becomes strongly dominant over diffusion losses, Saha equilibrium is established.

Equation (4.2) is a good example of a Disturbed Bilateral Relation.The proper balance would be

$$n_p n_e k_{ion} - n_e^3 k_{rec} = 0 \tag{4.3}$$

because ionization and three-particle recombination are each others inverse processes. The balance (4.2) is called “Disturbed” because there is an extra loss

[2]For instance, the common Arrhenius-like reaction rate has superexponential dependence of reaction rate on temperature (4.11)

term. This loss term is the diffusion (a transport term), and it is quite often dominant. (In the solution procedure, it is indeed assumed that diffusion is dominant, and that the three-particle recombination is a small correction).

4.2.2 The electron energy balance

The electron energy balance equates the heating to the cooling of the electrons. It is given by equation (4.4).

$$\varepsilon = n_e n_p k_{heat}(T_e - T_h) + (n_p n_e k_{ion} - n_e^3 k_{rec}) E_{ion} \tag{4.4}$$

In this equation, ε is the power density, k_{heat} the heat transfer coefficient between electrons and heavy particles and E_{ion} the ionization energy of an atom. The power density ε is essential to sustain the plasma; without a power supply, conduction and diffusion will remove the heat necessary for ionization, thereby turning the plasma into a non-ionized gas. The terms on the right-hand side are proper balances; the term on the left-hand side can be seen as a disturbance to that. Often, the diffusion losses represent the most imortant loss process. This can be made explicit by rewriting (4.4) using (4.2) to

$$n_e n_p k_{heat}(T_e - T_h) = \varepsilon - \frac{n_e D^*}{n_p \Lambda^2} E_{ion} \tag{4.5}$$

The heavy particles get heated by elastic collisions with the hotter electrons. In each collision, the electron loses a small portion of its energy to the heavy particle. This represents a loss term for the electron energy.

When a particle recombines at the wall, it takes energy from the plasma with it. The majority of the energy is chemical: the ionization energy of the gas. In fact, also the thermal energy of the electron and of the ion are lost; these are, however, generally small and not taken into account. Radiation losses, such as radiative decay, are not taken into account, either.

Equation (4.5) assumes that all the energy is coupled into the electrons. Also, heat conduction by the electrons is not taken into account. This may lead to substantial errors if the ionization degree becomes large (larger than approximately 1%). A correct treatment of this effect is beyond the scope of this project, because it would require knowledge of the electron wall temperature, which is generally not known unless the profile of the electron temperature is calculated. More on the complexities of the electron heat conductivity can be found in chapter 5.

4.2.3 The heavy particle energy balance

The heavy particle energy balance equates the heating of the heavy particles to the conductive energy loss.

$$n_e n_p k_{heat}(T_e - T_h) = k_{cond} \frac{T_h - T_{wall}}{\Lambda^2} \tag{4.6}$$

Here, k_{cond} is the heat conduction coefficient of the heavy particles. T_{wall} is the wall temperature of the heavy particles. The left-hand side term represents the energy gained in collisions with electrons, and is identical to the second term of equation (4.5). It is assumed that this is the only heating process. The heavy particles lose their energy through conduction. This effect is described by the second term.

4.3 Using the DBR to analyze a plasma

The DBR provide us with a set of equations that describe the plasma. If these equations are solved for known input parameters, a prediction can be made about the main plasma parameters temperature, pressure and composition of the plasma. We assume that the power, the geometry, Λ, the main plasma constituent and the heavy particle density n are known. This serves as input. Furthermore, a cylindrical geometry is assumed. The power and geometry are used to calculate ε.

This yields a total of 8 unknowns, 3 plasma variables and 5 parameters, with only 3 equations. As can be seen in equations (4.2), (4.5) and (4.6), there are 5 other variables that are generally not known beforehand: k_{heat}, k_{ion}, k_{cond}, k_{rec} and D^*. These variables all depend on T_e, T_h or both, and on the physcal and chemical chemical properties of the main plasma constituent. In order to solve this problem, we need to find equations for these variables in terms of T_e, T_h and these parameters. Until now, argon is simulated; however, any gas can in principle be dealt with.

4.3.1 An equation for k_{heat}

The second term in equation (4.5) describes the heat transfer between electrons and heavy particles. When an electron and a heavy particle collide elastically, the electron and heavy particle exchange energy and momentum. This exchange is not very efficient, because of the large difference in mass between heavy particles and electrons. In the model, only the electron/neutral collisions are taken into account. This is a good approximation for plasmas with a low (less than 1% for Ar) degree of ionization. For higher degrees of ionization, the more efficient electron/collisions become important.

The amount of heat transferred per second is basically the average amount of thermal energy per collision ΔE transferred times the number of collisions of atoms and electrons per second ν. The amount of heat transferred per collision is given by [1]:

$$\Delta E = \frac{3m_e}{M} k_B (T_e - T_h) \tag{4.7}$$

with M the heavy particle mass, m_e the electron mass and k_B the Boltzmann constant. The collision frequency per cubic meter is given by

$$\nu = n_e n_p \sigma_{ea} \sqrt{\frac{8k_B T_e}{\pi m_e}} \tag{4.8}$$

with σ_{ea} the collision cross section between electrons and neutrals. This cross section is generally a function of T_e. We can now write an expression for k_{heat}.

$$n_e n_p \sigma_{ea} \sqrt{\frac{8k_B T_e}{\pi m_e}} \frac{3m_e}{M} k_B (T_e - T_h) = n_e n_p (T_e - T_h) k_{heat} \tag{4.9}$$

Using this, we obtain

$$k_{heat} = \sigma_{ea} \sqrt{\frac{8k_B T_e}{\pi m_e}} \frac{3m_e}{M} k_B \tag{4.10}$$

This leaves us with a definition of k_{heat} that depends only on T_e. It can be calculated if T_e is known.

4.3.2 An equation for k_{ion}

The ionization reaction rate k_{ion} is approximated by an Arrhenius form in the program. It has the general form of equation (4.11):

$$k_{ion} = k_{rate} T_e^q \exp\left(\frac{-E_{ion}}{k_B T_e}\right) \tag{4.11}$$

The input parameters are the rate constant k_{rate}, the ionization energy E_{ion} and the temperature exponent q. These are properties of the plasma constituent and must be supplied for the calculation to be able to start. The power exponent q is generally between 0 and 1.

Equation (4.11) can be combined with equation (4.2) to determine the electron temperature, if k_{rate} is known. In practice, a minimum value of k_{ion} is required to compensate for losses of electrons to the wall by diffusion and three-particle recombination. This determines the electron temperature T_e. The same approach of determining the electron temperature is used in the DBR program.

4.3.3 An equation for k_{cond}

The heat conduction plays an important role in the determination of the heavy particle temperature. [1] offers a derivation of this variable.

$$k_{cond} = \sqrt{\frac{8k_B T_h}{\pi M}} \frac{\sqrt{2}}{\sigma_{aa}} k_B \tag{4.12}$$

with M the heavy particle mass and σ_{aa} the neutral-neutral collision cross section. This is one of the input parameters. Note that k_{cond} is a function of T_h.

4.3.4 An equation for k_{rec}

Three-particle recombination is taken into account in the DBR model. The three-particle recombination rate has been obtained using the principle of detailed balancing:

$$n_1^s n_e k_{ion} = n_e^3 k_{rec} \tag{4.13}$$

where n_1^s is the Saha density. The three-particle recombination rate has been derived using the Saha equation (3.17). In Saha equilibrium, the three-particle recombination rate equals the ionization rate. This means that equation (4.3) holds. Equation (3.17), equation (4.11), equation (4.3) and the fact that the ion density n_i equals n_e yield an expression for k_{rec}:

$$k_{rec} = \left(\frac{h}{\sqrt{2\pi m_e k T_e}}\right)^3 \frac{k_{rate}}{2G} T_e^q \tag{4.14}$$

G is the ratio of the degeneracy of the ion state and the ground state. The factor 2 in the denominator comes from the the degeneracy of the electron. It can be concluded that k_{rec} also depends on T_e and T_h and the basic physcal and chemical parameters of the gas.

4.3.5 An equation for D^*

In the DBR program, the assumption of ambipolar diffusion is made. In this case, the electrons diffuse away, but due to the Coulomb forces, the ions get dragged along. The result is that the electron-ion pairs diffuse with the inertia of the ions, but with the temperature of the electrons. D^* has to be expressed as a function of T_e and the basic plasma parameters.

$$D^* = D_i^*(1 + \frac{T_e}{T_h}) \tag{4.15}$$

Equation (4.15) expresses D^* as a function of T_e, T_h and of the reduced ion diffusion coefficient D_i^*.

D_i^* also has to be expressed as a function of T_h and the physical and chemical properties of the plasma constituent. It is given by:

$$D_i^* = \frac{k_B T_h}{M} \tau_{ia}^* \tag{4.16}$$

with τ_{ia}^* the reduced collision time. This is the collision time multiplied with the thermal velocity. This is equals the time it takes for a particle to sweep one cubic meter of space, as can be seen in equation (4.17):

$$\tau_{ia}^* = \sqrt{\frac{\pi M}{8 k_B T_h}} \frac{1}{\sigma_{ia}} \tag{4.17}$$

Here, σ_{ia} is the ion-atom collision cross section, which generally depends on temperature. The diffusion coefficient is calculated by combining equation (4.15), equation (4.16) and equation (4.17), yielding:

$$D^* = (1 + \frac{T_e}{T_h}) \frac{k_B}{M \sigma_{ia}} \sqrt{\frac{\pi M T_h}{8 k_B}} \tag{4.18}$$

4.3.6 Conclusions

We now have a complete set of equations: eight nonlinear equations with eight unknowns. Of these, three equations provide the basic plasma parameters T_e,

T_h and n_e, whereas 5 equations take these parameters and use them to calculate the coefficients, which are again used by the first three equations. We do need to supply a number of constants that depend on nature of the particles. These are:

- the mass of the particle M
- the rate constant k_{rate}[3]
- the Arrhenius temperature exponent q
- the ionization energy E_{ion}
- the ratio of the degeneracy of the ion ground state and the atom ground state G

Furthermore, there are three energy dependent cross sections, σ_{ia}, σ_{aa} and σ_{ea}. These can be supplied to the program in the form of look-up tables. The program then makes simple interpolations and extrapolations to estimate the cross section. With this information, and several physical constants that are coded, we can solve the equations and determine T_h, T_e and n_e.

It is important to note, however, that results of this paragraph ar not valid for all plasmas. They apply, strictly speaking, only to plasmas that are made from a pure gas, that have an ionization degree of less than 1 % and that there are only single ions. If any of these conditions is not fulfilled, the approximate answer generated by this set of equations may be highly inaccurate. On the other hand, if the ionization degree becomes very small, maxwellization of the electrons may cease to be valid. In this case, the program also becomes unusable. There are, however, methods to tackle this problem.

In this section, the decrease of neutral particles caused by ionization is not taken into account. In the actual implementation, this effect is taken into account. Because the program only functions well for plasmas with a low (less than 1 %) degree of ionization, this effect is small. Therefore, it will not be discussed in depth; the interested reader can see the implementation in the source code.

4.4 The program*

In the previous sections it has been shown how the Disturbed Bilateral Relations can be used to make a set of equations that may be solved to produce the approximate values of T_h, T_e and n_e. However, solving a set of eight non-linear equations analytically is not easy; in fact, it is not generally possible. For this problem, an *ad hoc* method has been devised that exploits the form of the equations and the underlying physics for a relatively easy way of solving the equations. This method has been cast into a C++ program.

[3]In literature, the term rate coefficient is often used to indicate the speed of a reaction, called k_{ion} in this case. This is not what is meant here; here, k_{rate} represents the constant before the two temperature-dependent parts of the Arrhenius equation. This part is also called the frequency factor.

The heart of the program is the `class DBR`. The rest of the program operates the machinery provided in this class. When the program is executed, the necessary data, as explained in Section 4.3.6, will be loaded into the class. The actual solving of the equations is performed by calling the `calculate` member of this class. This member iteratively calculates T_h, T_e and n_e. The main program then accesses the class members and displays them. These steps will be discussed in more detail. The program can be found in appendix B. A table with key variables represented by names in the program is presented in Table 4.4.

4.4.1 The input

The first thing the main program does is creating the `DBR` object `b` by calling the constructor. When this object is created, the constructor `DBR:DBR` loads the values of several physical constants of nature. It also gives starting values for a few variables.

The main program then reads the name of the particle. The class member `getFileName` then checks if the file *foo*, *foo*ea, *foo*aa, *foo*ia are present, with *foo* the name that has been given as input. These files contain data that is specific for the particle. The main data file *foo* should contain the following items:

- the atomic mass in amu
- the rate constant k_{rate} in $\mathrm{m}^{-3}\mathrm{s}^{-1}$
- the Arrhenius temperature exponent q
- the effective ionization energy E_{ion} in eV
- the ratio of the degeneracy of the ion ground state and the atom ground state G

The other three files contain the electron-neutral collision cross section, the neutral-neutral collision cross section and the ion-neutral collision cross section. These are generally a function of the energy. In the program, these energies are translated to temperatures[4]. They are stored in the files as a look-up table. The files start with a number that gives the number of lines this look-up table contains. Then, it contains a table with in the left column the temperature and in the right column the cross section. The length of the table and the temperature values at which they are taken are arbitrary, as long as the table length does not exceed 10000 entries. For optimal performance, there should be sufficient entries in an energy range where the cross section varies strongly with the energy, such as the Ramsauer minimum. In in a range where the cross section hardly varies, there can be less. It is also recommended that the plasma temperature is covered by the table, as interpolation is more reliable than extrapolation.

[4]Note that this is not accurate. It is better to integrate the product of cross section and the Maxwell distribution at a certain electron temperature over the energy. This, however, is computationally very expensive

Table 4.1: The names of the most important variables in the program and the physical quantities they represent.

`ArrheniusPower`	q
`chemical`	The part of ε lost by diffusion
`conduction`	The part of ε lost by conduction
`D`	D^*
`Degrat`	G
`eps`	ε
`eVtoK`	eV/k_B, with eV an electronvolt, or $1.602 \cdot 10^{-19}$ J
`GuessTe`	An initial guess for T_e, used to start the calculation
`GuessTh`	An initial guess for T_h, used to start the calculation
`Hpcond`	k_{cond}
`ionEnergy`	E_{ion}
`kB`	k_B
`kheat`	k_{heat}
`kion`	k_{rate}
`l`	The length of the vessel
`lambda`	Λ
`mass`	The mass of the particle in kg
`me`	The mass of an electron kg
`mp`	The mass of a proton in kg
`na`	The neutral density in m^{-3}
`necenter`	n_e
`np`	The heavy particle density in m^{-3}
`pi`	π
`Planck`	h
`Power`	ε
`pressure`	p
`Power`	The input power
`r`	The radius of the vessel
`recrate`	k_{rec}
`Te`	T_e
`Th`	T_h
`Twall`	T_{wall}

Next, T_{wall} is read. The program uses the `getdouble` function for a robust way of reading a double[5]. It also checks for a positive temperature. If the entered number passes these checks, the class member `getTwall` loads the value into the class. Similarly, the heavy particle density is read and loaded into the class by the member `getNp`. The power, radius and length (note that this assumes a cylindrical geometry) are also read and the member `getPowerDensity` is called. This calculates the volume and uses it to calculate the power density, which is the stored in the class. Next, the diffusion length Λ are stored by the member `getSmallestDimension`. Furthermore, initial guesses of the electron temperature and the heavy particle temperature, necessary for a start of the iteration, are put in the class by the members `getguessTe` and `getguessTh`.

4.4.2 The calculation

The main program calls the `calculate()` member next. This member, as its name suggests, makes the iterative calculation that determines T_h, T_e and n_e. It furthermore calculates the pressure p, the amount of energy lost by the diffusion of electrons and ions ε_{chem} and the amount of energy lost by heavy particle heat conduction ε_{cond}[6].

The solver basically uses the theory outlined in section 4.2. There are two small differences, however. Firstly, it splits equation (4.4) in two parts: a chemical part and a heat transfer part. These are given in equation (4.19) and (4.20):

$$\varepsilon_{chem} = (n_p n_e k_{ion} - n_e^3 k_{rec})E_{ion} = \frac{n_e D^*}{n_p \Lambda^2} E_{ion} \tag{4.19}$$

$$\varepsilon_{cond} = n_e n_p k_{heat}(T_e - T_h) \tag{4.20}$$

Secondly, we take into account that the ionization of neutral atoms reduces the amount of neutral atoms present in plasma. This effect should be small, because as explained in subsection (4.3.6), the program only works well for low degrees of ionization (1%). This results in the program using the variable `n_a` for the neutral density and the variable `n_p` for the heavy particle density. These two effects, however, increase the number of equations and unknowns to 10. On the other hand, equation (4.11) is incorporated in equation (4.2), thus eliminating k_{ion} and setting the number of equations that have to be solved to 9.

The loading of the particle properties

The procedure starts with determining the amount of iterations. It is set at 100, which is more than enough in practice. Then, n_e is estimated as the square root of the value of n_a, both in m^{-3} [7]. This is in fact a first guess, that will be updated later in the code. Next, the `loader()` member is called. This member loads the contents of the four particle data files into memory. It does

[5] A floating-point type with double accuracy

[6] This process relies on first transferring the heat from electrons to heavy particles, and then the heavy particles lose their energy by conduction.

[7] This seems a highly counterintuitive and awkward choice, especially from a dimensional point of view. Looking at the `CalculateTe` member, we see that this means that the recombination part becomes small and independent of the electron density and heavy particle density, which is the reason for the choosing this value for n_e.

so by calling the `readspecies()` member, which loads the main data file, the `readLUTia()` which loads the ion-neutral collision cross section look-up table, and `readLUTaa()` and `readLUTea()` which load the other two look-up tables. If any of these procedures gives an error, the `loader()` will return an error and the calculation will be aborted.

The pre-calculation

If the loading is successful, a precalculation will be started. This precalculation determines T_e for the initial guesses. This way, we can get a more or less reliable estimate of the electron temperature T_e before the real calculation is started. The advantage is that the simple initial calculation is not very likely to go wrong (unless conditions have been entered that cannot result in a stable plasma), while the enhanced initial guess of the electron temperature will result in improved stability of the main iterative procedure.

Firstly, `n_a` is set to `n_p` (Here, the loss of neutrals by ionization is not yet taken into account for stability reasons[8]). Next D^*, is calculated by the `calculateD()` member. The calculation is performed with Equation 4.18. Note that the `calculateD()` member calls the `getcrosssectia()` member. This calculates the ion-neutral collision cross section by linearly interpolating the look-up table. Should the requested temperature fall above (or below) the maximum (or minimum) value in the table, the maximum (or minimum) value will be given as return value. Note that this is generally not what is desired, so the range of the look-up table should contain the temperature found. Note that `getcrosssectaa()` and `getcrosssectea()` work similarly for the other look-up tables.

Next, k_{rec} is calculated by the `calculateThreeParticleRate()` member. Note that for both of these calculations, the initial guess of T_e is used. Both D^* and k_{rec} do not depend very strongly on T_e, therefore, these are reasonable starting conditions. Then, T_e is calculated from the particle balance (equation (4.2)). Here, we solve k_{ion}, because it depends strongly on T_e. Determining T_e from k_{ion} is not easy, because it depends on a power of T_e and an exponent with T_e in it (see equation (4.11)) This problem is tackled by taking the T_e that appears in the power as a constant (The value from the last iteration is used. If the calculation converges, the difference will approach zero). This constant is then removed by division. Then, T_e is calculated. Next, this new value of T_e is used to calculate D^* and k_{rec}. These new values are the used to calculate T_e, etc. etc.

The estimate of T_e thus obtained is used to make acceptable initial guesses for other parameters. `calculatekheat()` uses equation (4.10) to calculate k_{heat}. Next, `calculateHPcond()` calculates k_{cond} using equation (4.12). The member `calculateThfromPower()` calculates T_h, by using a combination of equation

[8]This effect is generally unimportant, as the calculation fails to give accurate results when ionization becomes so large that the difference becomes significant. Increased accuracy is not the reason it is implemented, though; rather, it prevents the program from giving nonsense output in the case of high ionization.

(4.6) and equation (4.20):

$$\varepsilon_{cond} = k_{cond}\frac{T_h - T_{wall}}{\Lambda^2} \tag{4.21}$$

T_h should be lower than T_e, because the energy is supplied to the electrons, and they transfer it via a temperature difference to the heavy particles. `calculateThfromPower()` can be set to check whether this is true, and if it is not true, it will lower the amount of heat lost by conduction until this condition is fulfilled. This way, we not only get a first estimate for T_h, but also for `chemical()`. Now, we can calculate n_e from equation (4.4). Note that both members are linear in n_e, which greatly simplifies the calculation. The calculation is performed by `calculatene()`. Next, `na` is calculated by the member `calculatena()`. This member also makes sure that n_e does not exceed n_p.

The main iterative calculation

At this point, sensible starting values of the variables needed in the iteration are generated. This iteration starts by calculating n_e and T_e. Next, `calculateThfromPower()` is called. Its result is overwritten by `calculateThHP()`, so only the fact that it puts a ceiling on `chemical` is important. `calculateThHP()` calculates T_h using equation 4.6.

With n_e, T_e and T_h, we use them to calculate the other necessary variables. `calculatePowerDistribution()` calculates `chemical` and thus `conduction`. It uses an equation similar to equation (4.19). Next, k_{cond}, D^*, k_{heat}, n_a and k_{rec} are calculated.

These steps are then repeated for the number of times specified by `imin` and `imax`. Next, the pressure is calculated. The function then returns a value that indicates whether an error was encountered.

4.4.3 The output

The code used for generating output is trivial. If there has been an error during the calculation, it tells you that this has happened. If not, it writes down n_e, n_a, T_e, T_h, p, `chemical` and `conduction`. There are class members that are inline functions that return the appropriate variable as a function value.

4.5 DBR and PLASIMO

The previous sections contain a description of the DBR theory and the program. In this chapter, we will investigate whether certain trends are accurately predicted and for which range of parameters the program yields acceptable predictions. Furthermore, the effect of the improved starting condition on the speed of PLASIMO is investigated.

It is important to note that the DBR program does not calculate profiles, but only central values. The user still has to make a sensible guess for the profile. In practice, a parabolic profile works well in PLASIMO.

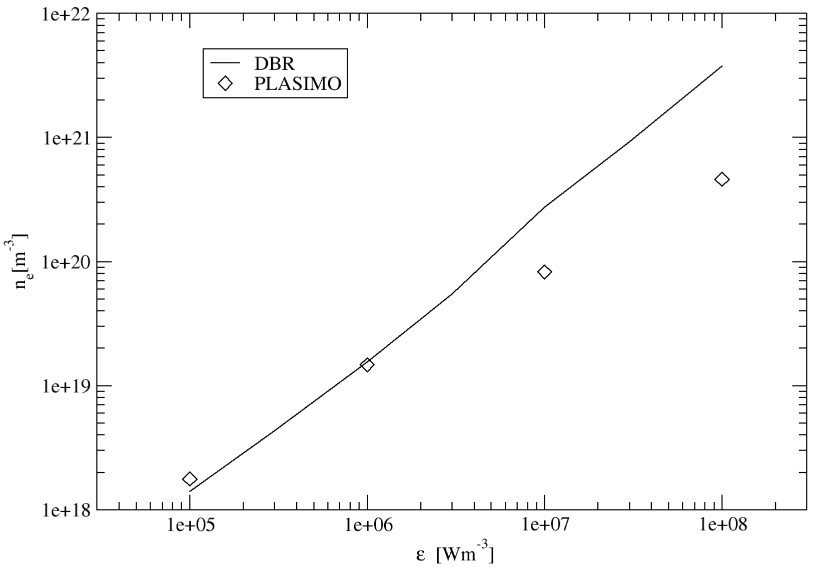

Figure 4.2: The central n_e as a function of ε, for an Ar plasma in an infinite cylinder with a radius of 10 mm and a central value of n of $1{\cdot}10^{22}$ m^{-3}. The graph contains the results obtained by PLASIMO and the DBR program.

4.5.1 Predicting trends with the DBR program

In this section, we will examine whether some general trends are accurately predicted by the DBR program. Although this simple and somewhat crude program cannot be expected to yield very accurate results, general trends should be predicted by the program. The following trends will be investigated: varying the power, varying the number of particles, and varying the diffusion length.

Varying ε

The plasma model under study is time-independent. There is no flow involved. We use Ar as the main plasma species. The vessel is a 10 mm segment of a cylinder with a radius of 10 mm that is infinitely long. The central heavy particle density n is equal to $1{\cdot}10^{22}$ m^{-3}. The power P is varied between 10 Wm^{-1} and 10000 Wm^{-1}, resulting in an ε that varies between $1{\cdot}10^{5}$ Wm^{-3} and $1{\cdot}10^{8}$ Wm^{-3}. These parameters are then used as input for the DBR program. This program then calculates various quantities, such as T_e, T_h and n_e.

In order to validate our simple model, we compare the results with a PLASIMO calculation of these quantities. n_e as a function of ε can be found in figure (4.2) and T_e and T_h as a function of ε can be found in figure (4.3)

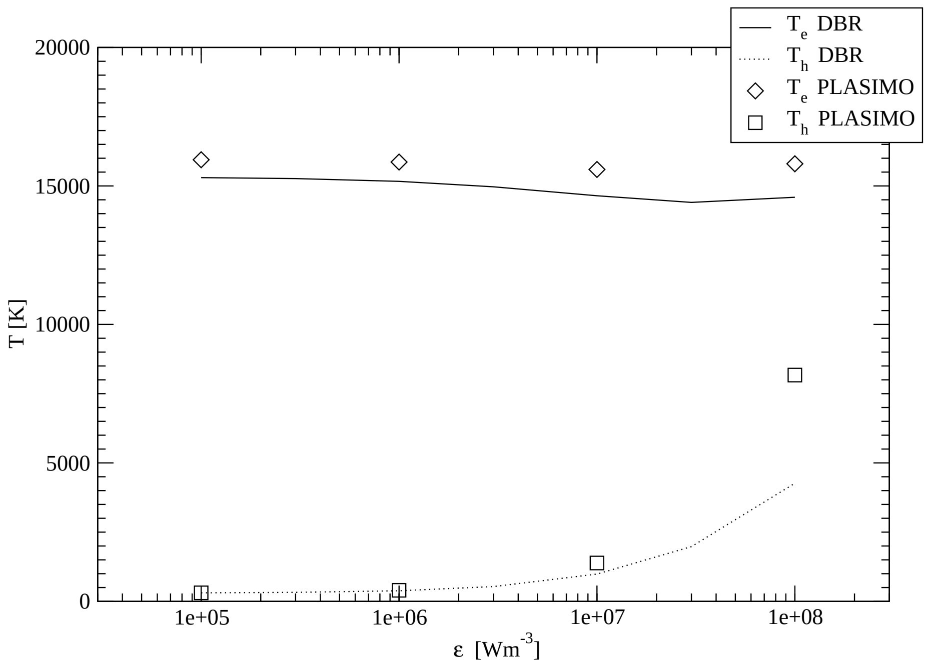

Figure 4.3: The central T_e and T_h as functions of ε, for an Ar plasma in an infinite cylinder with a radius of 10 mm and a central value of n of $1{\cdot}10^{22}\mathrm{m}^{-3}$. The graph contains the results obtained by PLASIMO and the DBR program.

Figure 4.2 shows that the DBR program predicts an almost linear dependence of the electron density on the power. This means, that the energy per electron does not change. This is a direct result from equation (4.5) where it can be seen that both loss processes are linear in n_e.

Figure 4.3 shows that T_e is almost independent on ε. In this plasma, diffusion and not recombination is the primary loss mechanism. In this case, the main dependence of the loss mechanism on input power is its linear dependence on n_e, which depends linearly on ε. The same is true for ionization, the production mechanism. These dependencies therefore cancel. As the electron temperature is determined by the requirement that it should produce sufficient ionization to keep the discharge going (equation (4.2)), and as the dependencies on ε cancel, T_e does not depend on ε.

Figure 4.3 shows that T_h increases with ε. The increase is roughly linear. The following explanation is offered: n_e increases almost linearly with ε (figure 4.2). The amount of energy lost by diffusion is linear in n_e and thus in ε. This means, that the fraction of the energy lost by conduction remains roughly constant, meaning that ε_{cond} increases roughly linearly with ε. T_h is now dependent on two equations: equation (4.12) and equation (4.20). Equation (4.12) shows, that k_{cond} depends on the square root of T_h (σ_{aa} is assumed to be constant). For most of the data points, T_h is close to 300 K, so k_{cond} is almost constant. If k_{cond} is constant, T_h-T_{wall} is linear in ε_{cond}. Therefore, T_h is roughly linear in ε. For values of T_h significantly above 300 K, deviations are expected and found.

As can be seen in both figure 4.2 and 4.3, PLASIMO shows the same general trends as the DBR program. There are, however, deviations, which will be discussed in the following paragraphs.

For high power, PLASIMO does not predict a linear increase of n_e with ϵ as the DBR program does. This difference is caused by an omission in the DBR program. For higher degrees of ionization, electron heat conductivity becomes important. This can dramatically increase the rate at which energy is lost, thus decreasing the amount of electrons. This is not implemented in the DBR program, because it is virtually impossible to implement this in an essentially one-control volume approach, like the one used in the DBR program.

The DBR program predicts the value of T_e with an error of less than 10 % with respect to PLASIMO. The general trend is identical for PLASIMO and DBR.

The DBR program accurately predicts T_h for low ionization. However, when it fails to produce correct n_e, it also fails to produces a correct T_h. When n_e is deviating, the electron energy balance (equation (4.5)) deviates. The electron energy balance is essential for a prediction of T_h; thus, when the energy balance deviates, deviations in T_h are to be expected.

Varying the particle density

The models discussed in this section are used as a basis for a study on the dependence of various plasma parameters on neutral particle density. In this case, the power is fixed at 100 Wm^{-1}, but n is varied between $3{\cdot}10^{23}$ m^{-3} and $1{\cdot}10^{20}$ m^{-3}. PLASIMO models in the same parameter range have been used to check the results. The resulting values of n_e as a function of n are presented in figure 4.4. The resulting values of T_e and T_h as a function of n are presented in figure 4.5.

Figure 4.4 shows a n_e that increases linearly with n for low values of n. Then, n_e reaches a peak value, after which it drops again. The explanation for this behavior lies in the energy balance, equation (4.5). When n is low, most of the energy is lost by diffusion. An increase in n causes a linear decrease in diffusion speed. This is compensated by an increase in n_e. This trend breaks down when conduction, and thus heat production, becomes significant. For conduction, an increase in n increases the speed at which energy is transferred from electrons to heavy particles, thus decreasing the amount of energy available for sustaining electron/ion pairs, so an increase in n causes a decrease in n_e.

Figure 4.5 shows that T_e increases as n decreases. This can easily be understood: when n decreases, the amount of diffusion increases, and thus a higher electron temperature is needed to sustain the plasma. Note that this will eventually cause the plasma to collapse, as the loss processes will become so fast

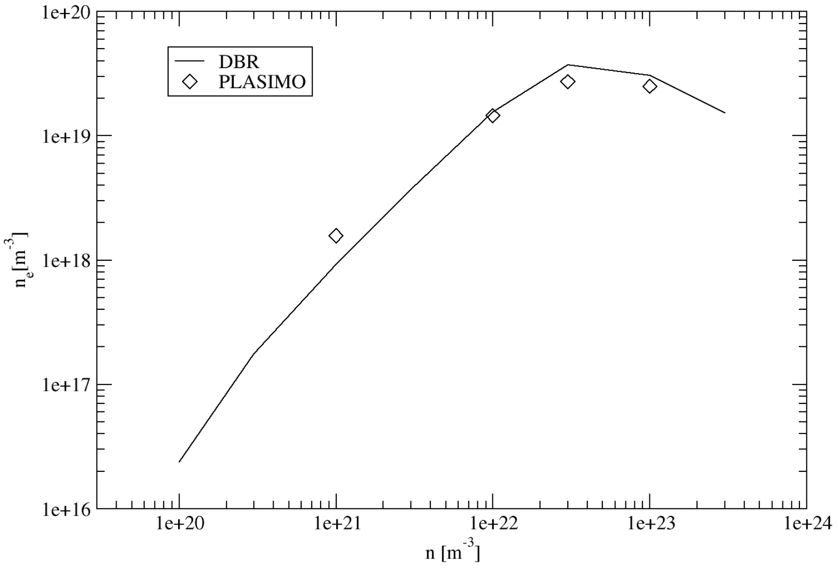

Figure 4.4: The central n_e as a function of n, for an Ar plasma in an infinite cylinder with a radius of 10 mm and an ε of $1{\cdot}10^6$ Wm^{-3}. The graph contains the results obtained by PLASIMO and the DBR program.

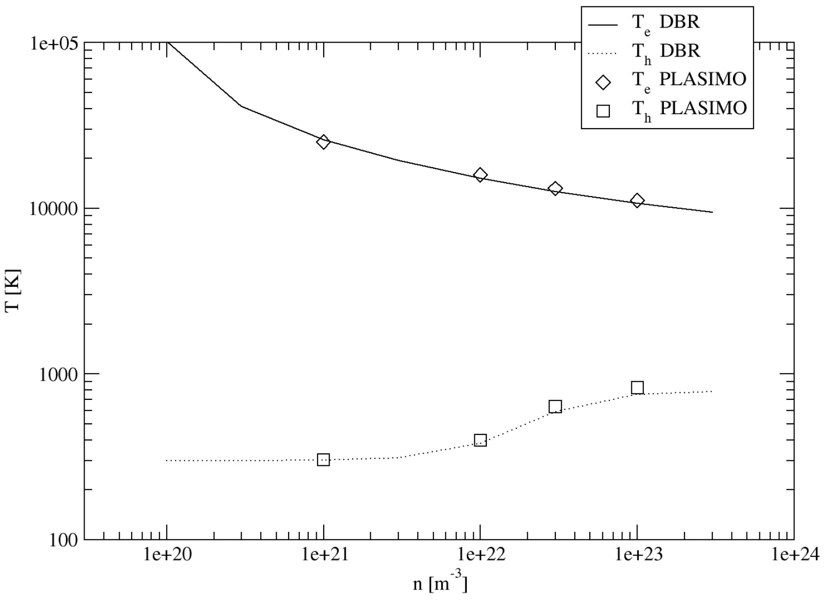

Figure 4.5: The central T_e and T_h as a function of n, for an Ar plasma in an infinite cylinder with a radius of 10 mm and an ε of $1{\cdot}10^6$ Wm^{-3}. The graph contains the results obtained by PLASIMO and the DBR program.

that the plasma cannot be sustained. For the DBR program, this means that the temperature will rise to infinity and the program will give errors. Using the DBR program for parameters for which the discharge can only barely be maintained may lead to large errors, because in this range, there are significant deviations from the Maxwell distribution, which are not taken into account in the DBR program. Also, Knudsen flow would need to be taken into account to accurately describe the diffusion (See chapter 7).

Figure 4.5 shows that T_h increases as n increases. This can be explained by the fact that as n increases, more heat is transferred from the electrons to the heavy particles, and in order to transport this heat, a larger temperature gradient is needed, as the heat conduction coefficient is more or less independent of n. Thus, T_h has to be high at the center.

The results of the DBR program match the results produced by PLASIMO closely. Not only are all the trends identical, the temperatures match within 10%, and the electron density deviates less than a factor of 2.

Varying the diffusion length

The models discussed in this section are used as a basis for a study on the dependence of various plasma parameters on diffusion lenght. In this case, ε is fixed at $1{\cdot}10^6$ Wm^{-3}, but Λ is varied between 0.001 m and 0.1 m[9]. PLASIMO models in the same parameter range have been used to check the results. The resulting values of n_e as a function of Λ are presented in figure 4.6. The resulting values of T_e and T_h as a function of Λ are presented in figure 4.7.

Figure 4.6 shows that the electron density increases as Λ increases. When Λ increases, the amount of particles lost by diffusion decreases. This means, that less energy is lost per electron, and that the amount of electrons may thus rise. This trend continues until Saha equilibrium is reached in the limit of no diffusion.

Figure 4.7 shows that T_e decreases as Λ increases. An increasing Λ means decreasing diffusion, and thus the amount of ionization can be lower to sustain the discharge. This results in a lower value of T_e.

Figure 4.7 shows that T_h increases as Λ increased. Increasing Λ decreases diffusion, which means that more heat must be transferred by conduction. This requires a larger heavy particle temperature gradient. The increase in Λ and the increase in heavy particle temperature gradient result in an increase in T_h if Λ increases.

For short diffusion lengths (much diffusion) the electron density is low and the DBR program and PLASIMO do not differ much. For longer diffusion lengths, electron heat conduction starts to become important in PLASIMO, and the energy distribution will start to differ, resulting in large differences in n_e and

[9]In this case, Λ is equal to r.

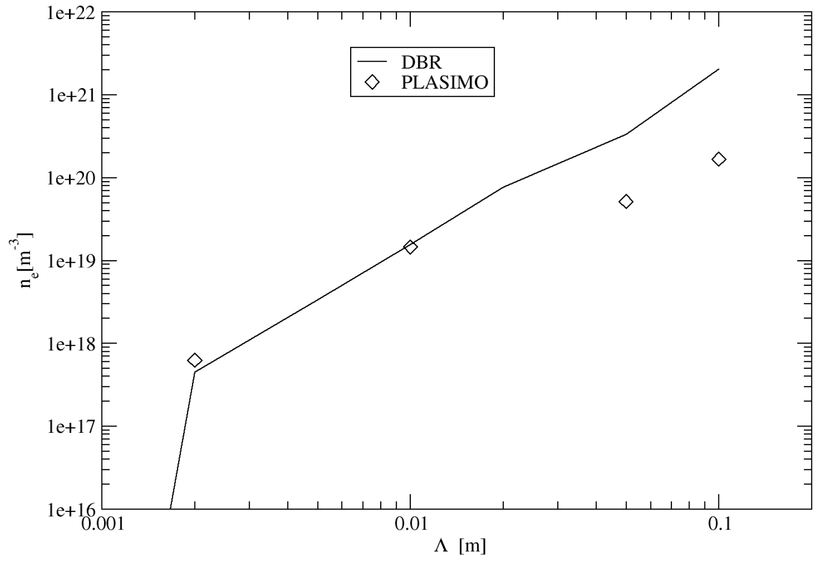

Figure 4.6: The central n_e value as a function of Λ, for an Ar plasma in an infinite cylinder. The ε is fixed at $1{\cdot}10^6$ Wm^{-3}, and n is set to $1{\cdot}10^{22}$ m^{-3}. The graph contains the results obtained by PLASIMO and the DBR program.

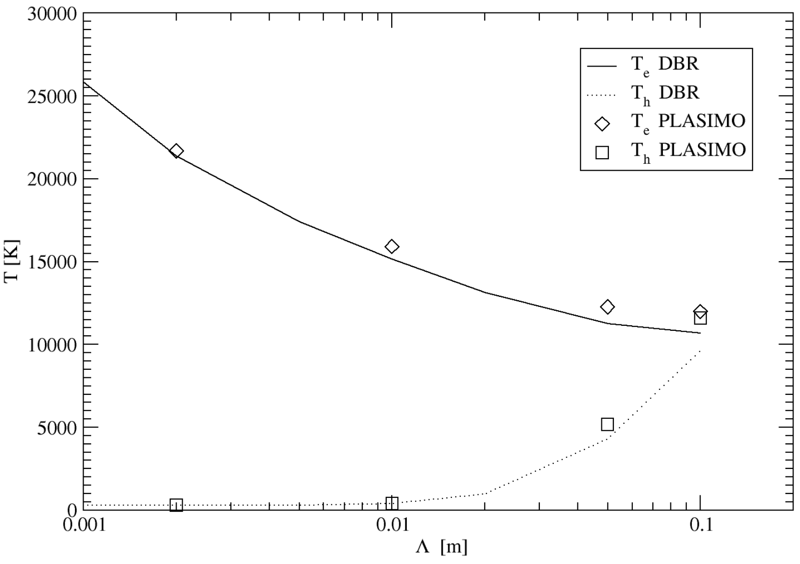

Figure 4.7: The central T_e and T_h as a function of Λ, for an Ar plasma in an infinite cylinder. The ε is fixed at $1{\cdot}10^6$ Wm^{-3}, and n is set to $1{\cdot}10^{22}$ m^{-3}. The graph contains the results obtained by PLASIMO and the DBR program.

Table 4.2: Some key features of the three plasma settings under study to determine the effect of the starting condition on convergence speed.

	Λ [m]	ε [Wm^{-3}]	n [m^{-3}]
setting 1	$1{\cdot}10^{-2}$	$1{\cdot}10^{6}$	$1{\cdot}10^{22}$
setting 2	$1{\cdot}10^{-2}$	$1{\cdot}10^{6}$	$1{\cdot}10^{21}$
setting 3	$1{\cdot}10^{-1}$	$1{\cdot}10^{6}$	$1{\cdot}10^{22}$

T_h. On the other hand, there will not be large differences between DBR and PLASIMO for T_e, because the DBR program does not use the electron energy balance, where the deviation occurs, for the calculation of T_e.

4.5.2 Performance

One of the reasons to construct the DBR program was to obtain good starting conditions for PLASIMO, improving PLASIMOs stability. To investigate the ability of the DBR program of providing good initial conditions for PLASIMO we studied the convergence behavior of three different plasma settings. The parameters of these models are presented in table 4.2.

For each setting, the number of steps necessary to converge to a PLASIMO residual of $1{\cdot}10^{-6}$ is given. Three different start conditions were taken:

- An simple starting condition of T_e=12000 K, T_h=3000 K and n_e=0.01n (called Simple in table 4.5.2)
- The starting condition generated by DBR (called DBR in table 4.5.2)
- The central values calculated by PLASIMO are used to construct parabolic profiles, which are used as starting condition (called PLASIMO in table 4.5.2)

In each case, it is assumed that T_e at the wall is the same as in the center, that T_h drops parabolically to T_{wall} and that n_e drops parabolically to 1 m^{-3}.

These models are run by PLASIMO with the different starting conditions. The number of iterations these models need to converge to $1{\cdot}10^{-6}$ is given in table 4.5.2. It shows that the DBR program is better than the simple first guess in the first two cases, as expected. It does not perform well in the last case. In this case, there is lots of electron heat conductivity, and this causes the DBR program to yield inaccurate results, as explained above. As expected, the starting values that were obtained from the output of the PLASIMO model give the fastest convergence, as they are closest to the solution. For the range of parameters where the model is applicable, the DBR program produces data that it suitable for use as an input parameter.

4.6 Scaling laws and DBR

As stated before another important background of the construction of the DBR was to get insight in scaling laws. We start with the electron particle

Table 4.3: The number of iterations needed for the models to converge for the different starting conditions

	Simple	DBR	PLASIMO
setting 1	46496	44481	9829
setting 2	157595	78227	151941
setting 3	8001	24433	7566

balance which is reproduced here without the recombination term so that it reads:

$$k_{ion} = \frac{D^*}{n_p^2 \Lambda^2} \tag{4.22}$$

Now, the electron temperature T_e is determined by the value of k_{ion} necessary to sustain the plasma. This means, that T_e depends on the product $n_p\Lambda$: if this product remains constant, so does T_e, because the dependence of D^* on T_h is of minor importance for a broad range of the other variables. In practice, this picture becomes less clear when step-wise ionization becomes important.

Equation (4.19) shows similar behavior, if appropriately rewritten:

$$\frac{P_{chem}}{Ad} = \frac{n_e D^*}{n_p \Lambda^2} E_{ion} \tag{4.23}$$

with A the area of cylinder wall. Now, if it is assumed that d is equal to Λ, and that the portion of the heat removed by conduction is negligible compared to the portion lost by diffusion, we obtain:

$$\frac{P}{A} = \frac{n_e D^*}{n_p \Lambda} E_{ion} \tag{4.24}$$

In this equation, it is seen that also n_e solely depends on the product $n_p\Lambda$, provided that P/A is kept constant. With the assumption that d is equal to Λ, it can be seen that both T_e and n_e scale with the value of the product $n_p d$. This means that if n_p is varied, while $n_p d$ and PA^{-1} are kept constant, T_e and n_e should remain constant. This will be used to verify the DBR program.

To this end, a set of models is made. These model are made with the following parameters:

- n is varied between $2{\cdot}10^{20}$ m^{-3} and $1{\cdot}10^{23}$ m^{-3}
- The product nd is kept at a constant value of $1{\cdot}10^{20}$ m^{-2}
- An Ar plasma is used
- A power of $3.18{\cdot}10^5$ Wm^{-2} is used.
- The plasma is a 0.1 m long segment of an infinite cylinder

The resulting values of n_e as a function of n are presented in figure 4.8. The resulting values of T_e and T_h are presented in figure 4.9.

Figure 4.8: n_e as a function of n for a plasma in which n is varied, while keeping nd and the power per surface area constant.

Figure 4.9: T_e and T_h as a function of n for a plasma in which n is varied, while keeping nd and the power per surface area constant.

Figure 4.8 shows that n_e is indeed almost independent of n_p. The slight dependence of n_e on n_p may be caused by the fact that T_h does vary a bit (figure 4.9), thus changing $D*$ and causing deviations.

Figure 4.9 shows that T_e is indeed almost independent of n_p. Here, the small deviations (less than 1% with variations of n_p of more than two orders of magnitude) are attributed to the small variations of $D*$ caused by the variations in T_h.

It can clearly be seen in figure 4.9 that T_h, and in particular T_h-T_{wall} does depend strongly on n_p. This can be explained using a modified electron energy balance. If it is assumed that Λ equals d, then equation (4.21) can be rewritten to:

$$\frac{P_{cond}}{A} = k_{cond}\frac{T_h - T_{wall}}{\Lambda} \tag{4.25}$$

It can clearly be seen that T_h depends on Λ.

4.7 Conclusions

In this study, the Disturbed Bilateral Relations have been used to create a program that can be used to obtain a rough estimate of the main plasma parameters n_e, T_e and T_h, usable as a starting condition for an iterative procedure.

The DBR program has been tested, and the test results have been compared to results obtained by PLASIMO and compared to theoretical trends. The DBR program compares well with PLASIMO and the theoretical trends for low degrees (less than 1%) of ionization. For higher degrees of ionization, the fact that electron heat conductivity is not taken into account causes the DBR program to yield inaccurate results. It is therefore not recommended to use the DBR program in this regime.

The speed improvement obtained by using the results of the DBR program as a starting condition have been investigated. The results of the DBR program indeed seem to result in a substantial speed improvement. This, however, is not the case if the DBR program is used to get a starting condition for plasmas with more than 1% ionization.

The DBR program can also shed light on various scaling laws. It is also useful in determining the relative important of transport and equilibrium processes.

Chapter 5

From NLTE to LTE

5.1 Introduction

In chapter 3, fluid flows and LTE plasmas were discussed. Armed with the knowledge on equilibrium and equilibrium disturbing processes obtained in chapter 4, we will now discuss the next stage in departure from equilibrium: the NLTE plasma.

PLASIMO is capable of simulating NLTE plasmas. In a NLTE plasma, we not only have ϕ-equations for momentum and mass conservation, but also two ϕ-equations for temperature, one for the electron temperature and one for the heavy particle temperature. Moreover, various ϕ-equations are needed to describe the transport of various individual species. This gives us many more ϕ-equations than in the LTE-case, where we only have one temperature ϕ-equation and where the chemical composition is handled by equilibrium relations.

PLASIMO uses its NLTE module for simulating NLTE plasmas. In this section, it will be tested whether this module can handle plasmas that are near LTE. By gradually taking a plasma from NLTE to LTE, it should be possible to see the system reach thermal and chemical equilibrium, where thermal equilibrium stands for T_e=T_h and chemical equilibrium to the conditions for which the chemical composition obeys the laws of statistical mechanics, more specifically, when the degree of ionization obeys the Saha equation. This has been done in two ways: the power density can be increased and the pressure can be increased. Of course, total equilibrium is a concept that exists only in theory; in this section, a system will be considered to be in thermal equilibrium when T_h and T_e differ less than 1% and a system will considered to be in chemical equilibrium when n_e and the electron density predicted by the Saha equation n_{Saha} differ by less than 1%.

5.1.1 Increasing The Power Density

The plasmas simulated here are similar to the plasma described in section 3.3.2. They consist of Ar with a pressure of 1000 Pa at 273 K, in a segment of a long tube that is 0.1 m in length and a radius of 0.25 m. Only the ground

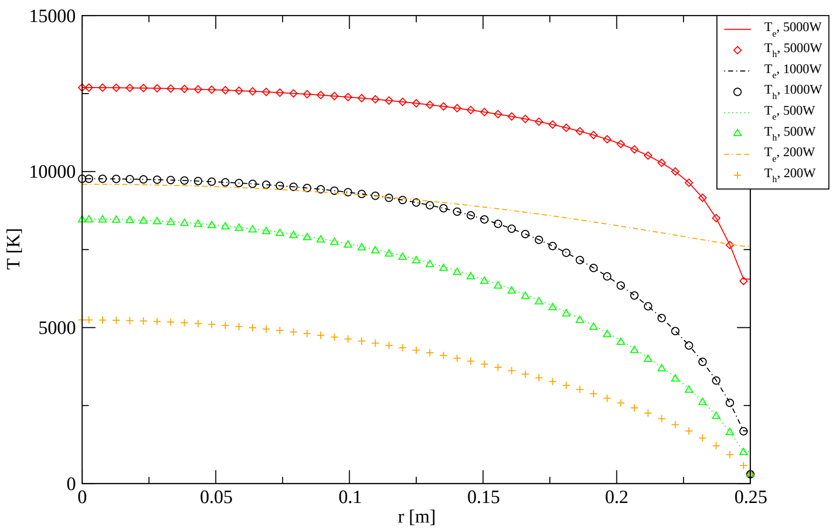

Figure 5.1: T_h an T_e of an Ar plasma, calculated by a PLASIMO NLTE model. The plasma remains close to LTE for powers of 500 W and up. Note that T_e for 200 W is higher than for 500 W. For T_h, some of the wall points have been omitted for clarity. It was in all cases equal to the boundary condition T_h=300 K.

state of the ion and the ground state of the atom are taken into account. The electric power is coupled in by an uniform electric field in the z-direction. The plasma recombines at the wall. The wall temperature for the heavy particles is fixed at 300 K; the boundary condition at the wall of the electron temperature is a Homogeneous Neumann boundary condition (meaning that the derivative in the direction perpendicular to the boundary is zero). The power input is varied between 200 and 5000 W. This results in the temperature profiles presented in figure 5.1 and the n_e profiles presented in figure 5.2.

Increasing the power generally increases the electron density 4. The increase in electron density causes more frequent electron-heavy particle collisions, which reduce the temperature difference between electrons and heavy particles. It also increases the ionization and recombination reaction rates, making transport relatively less important. Figure 5.1 shows the temperatures and shows that for lower power densities, the less frequent collisions between electrons and heavy particles allow the decoupling of T_e and T_h, whereas for higher temperatures, the system is close to thermal equilibrium. The higher T_e means that the electron density can stay relatively high, which can be seen in figure 5.2. This figure also shows that large deviations from Saha equilibrium occur at higher power than deviations from thermal equilibrium. Also, the system is not in Saha equilibrium near the wall. The main cause is that the drop in electron density predicted by Saha's formula is diminished by diffusion from other parts of the plasma.

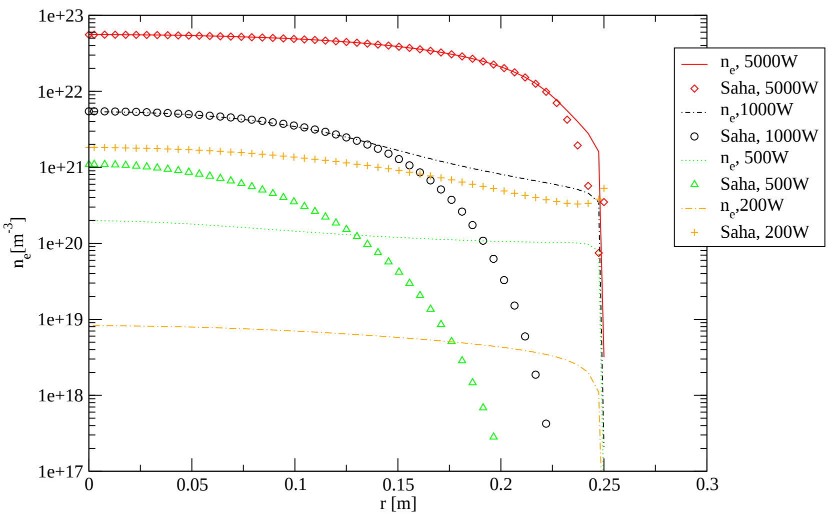

Figure 5.2: n_e of an Ar plasma, calculated by a PLASIMO NLTE model. It is compared with the Saha density.

5.1.2 Increasing Pressure

Another strategy to influence the collision rate and thus the degree of equilibrium departure is to vary the pressure. The plasma used in this case is similar to the one presented in section 5.1.1. The power is fixed at 1000 W, while the pressure is varied between 100 Pa and 2000 Pa.

This approach causes only small variations in temperature, therefore, T_e - T_h rather than T_h and T_e have been plotted in order to provide clear graphs. This graph can be found in figure 5.3. A graph of the electron density, compared to the Saha value of the electron density, is presented in figure 5.4.

The collision rate between n_e and n_a scales with n_e and n_a. Therefore, the difference between T_h and T_e is inversely proportional in $n_e \cdot n_a$. A comparison between figure 5.3 and figure 5.4 shows that this is indeed approximately true. It can also be seen that the differences are larger near the wall. This is caused by the boundary condition: the heavy particle temperature is fixed at the wall, while the electron temperature is not. There are large gradients in the heavy particle temperature. This means that a lot of energy has to be transferred from the electrons to the heavy particles. This causes a relatively large difference between T_e and T_h.

For higher pressures, Saha equilibrium is established in the center of the tube. A decrease in pressure will cause an increase in diffusion, and therefore Saha equilibrium will be violated when the pressure drops. The nonequilibrium region near the wall will expand and fill the whole tube, and the deviations from equilibrium will become larger.

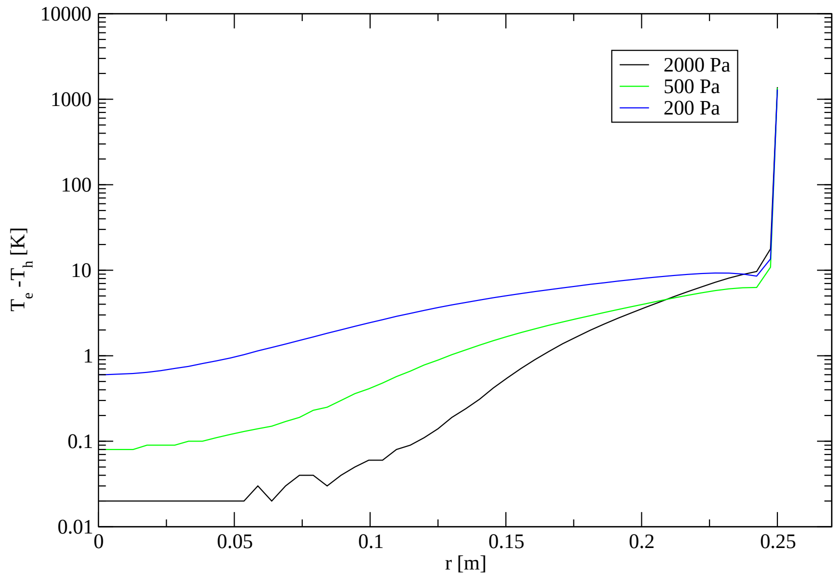

Figure 5.3: (T_e-T_h) of an Ar plasma, calculated using a NLTE approach, for various pressures. The precision is 0.01 K

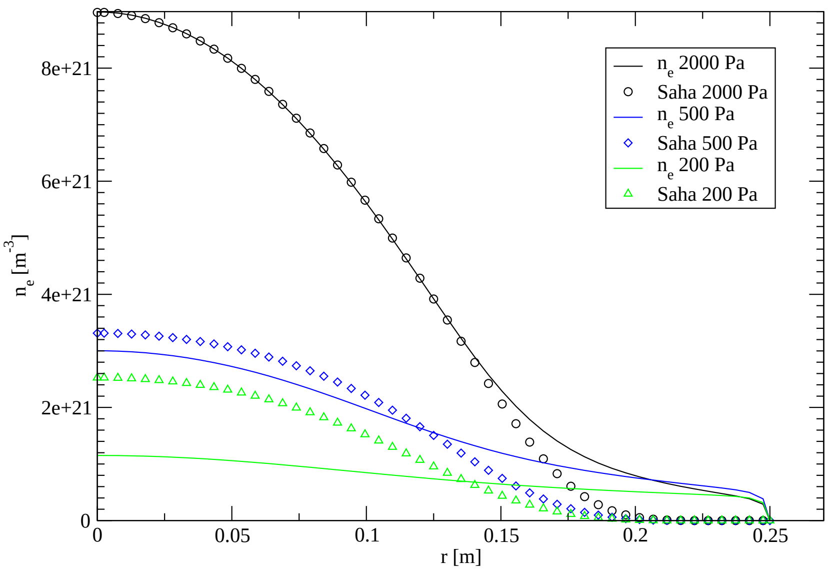

Figure 5.4: n_e of an Ar plasma, calculated using a NLTE approach. It is compared with the Saha density, derived from the numerical values of the electron temperature and the neutral density.

5.1.3 Computational Speed

Using the NLTE code for a plasma that is nearly in LTE turned out to have an important disadvantage. The computational speeds for the models that were relatively far from equilibrium (all plasmas that had a power of 1000 W or less) was roughly the same (about $5 \cdot 10^4$ iterations to get the residual down to 10^{-11}). The other plasmas reached an intermediate solution that resembles the real solution closely in about $2 \cdot 10^4$ iterations. However, the final convergence to a residual of 10^{-11} takes considerably longer for 2000 W (about $3 \cdot 10^5$ iterations) and even longer for 5000 W (about $2 \cdot 10^6$ iterations). This stage in the convergence is quite smooth, but extremely slow.

A possible cause for this is in the way PLASIMO calculates the energy flows from heavy particles to electrons and back. In equilibrium, the effective transport by these flows approaches zero. This will also happen in PLASIMO. However, the way in which this happens may cause problems. The energy flow equation can schematically be represented by

$$Tr\phi = P - D\phi \tag{5.1}$$

where ϕ represent temperature in this case, Tr represents transport, P represents production and D represents destruction. The production and destruction terms, called S_c and S_p in PLASIMO, become very large if we get close to equilibrium. This causes two problems:

- The fact that of the absolute value of $|S_pT|$ and $|S_c|$ are large means that the transport become almost negligible. Thus, changes in the temperature in a neighboring grid point will only very slowly affect the value in the grid point, slowing convergence [3]. Or, simply put, using a transport equation when transport is negligible doesn't work very well.

- The increasing $|S_c|$ and $|S_pT|$ and the decreasing difference between them may cause cancellation. This occurs, when two numbers that differ little are subtracted. This leads to inaccurate, or in the worst case, arbitrary results. This is obviously not desireable. Note that this is a computer-caused problem.

5.2 Deviations between the LTE and NLTE approaches

In the previous section we have seen that the convergence of a NLTE model becomes poor when it approaches LTE. A comparison of the results, however, reveals a far more serious problem. Comparing figures 3.6 and 5.1 shows a sizable difference between the results of the NLTE and LTE calculation. Even when the NLTE model gives an electron density close to Saha equilibrium and T_e and T_h values almost equal to eachother, we see that the central temperature is 4K smaller than for the comparable LTE case. In this section, we will investigate the cause of these deviations and attempt to draw general conclusions about the validity of LTE treatments.An important observation guiding this study is the fact that the n_e found in the NLTE model is almost equal to the value prescribed

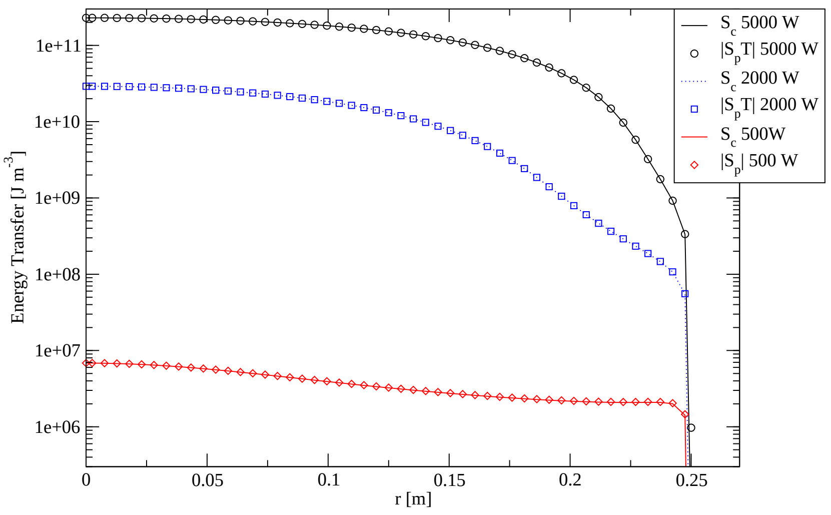

Figure 5.5: The energytransfer between heavy particles and electrons. The linear and constant part of the source term are of near equal magnitude. Thus, the difference is very small compared to either one. This is unfavorable from a numerical point of view.

by the Saha equation (3.17) in which the NLTE temperature is substituted. Thus, the set of NLTE results is self-consistent. All we need to do is find the cause for the temperture difference.

5.2.1 The models

In order to investigate the difference between the NLTE and LTE approach, two models has been made that describe the same plasma, but use either a NLTE or LTE approach. In order to facilitate the comparison, the models are kept as simple as possible, and every effort has been made to make the NLTE and LTE models as similar as possible, even if this would mean sacrificing some physical accuracy.

Adapting the model

The NLTE model has a different boundary condition than the LTE model. In the LTE model, the electron wall temperature, like the heavy particle wall temperature, is fixed at 300 K. In the NLTE case, the electron wall temperature is determined by the Homogeneous Neumann boundary condition. In order to make a cleaner comparison between NLTE and LTE, the boundary condition for T_e is set to 300 K. This will eliminate any possible influence of different boundary conditions on the deviations under study.

Of course, this boundary condition has less physical foundation than the Homogeneous Neumann boundary condition that is commonly chosen because of the inefficient heat transfer from the electrons to the wall. As our purpose is the investigation of the differences between LTE and NLTE, rather than obtaining

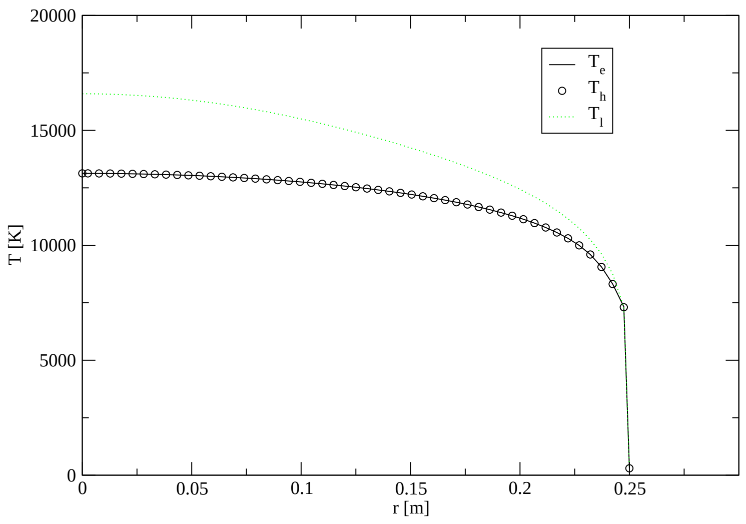

Figure 5.6: The temperature calculated with the LTE module T_l and T_e and T_h calculated with the NLTE module. The difference between T_e and T_h is very small, but differs strongly from the temperature calculated with the LTE model.

a quantitatively significant result, the reduced reliability of the numerical value is not relevant.

The model with the new boundary condition has been run with the PLASIMO code, using both the LTE and NLTE code. This has produced data on the temperature, which is presented in figure 5.6. It also produced data on the n_e, which is presented in figure 5.7.

Comparing figure 3.6 and 5.1 with figure 5.6 shows that the change in the boundary condition has no significant effect on the results of the temperature calculation. The electron density calculation is also not significantly influenced, as can be seen by comparing figure 3.5 and figure 5.2 with figure 5.7. From this, we conclude that the difference between the results of the NLTE and LTE calculations is not caused by de electron temperature boundary condition, and that changing the electron temperature boundary condition does not significantly impact the results.

In order to investigate the cause of the deviations between LTE and NLTE, the model is run with different settings. The new settings are chosen to amplify the effect causing the difference. This model differs at two points from the previous model:

- The filling pressure has been increased by a factor of 10 to 10000 Pa.
- The power input has been decreased by a factor of 5 to 1000 W.

This model has been run with the PLASIMO code, both with the LTE code and the NLTE code. The value of the temperature thus obtained is presented in

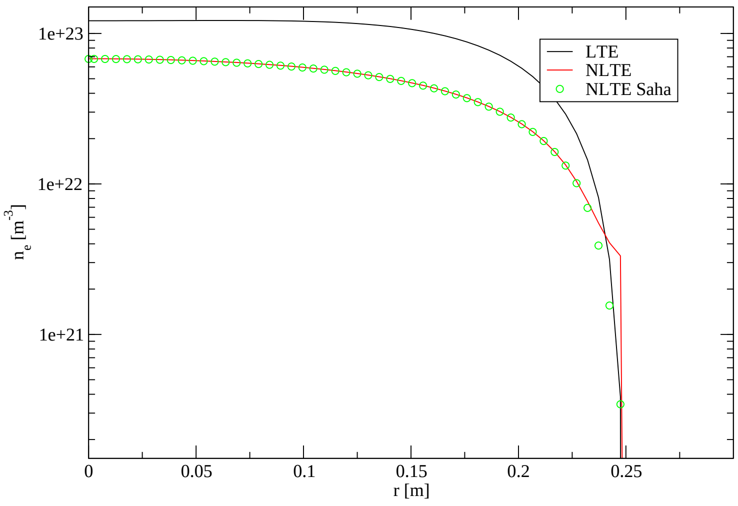

Figure 5.7: The electron density calculated with the LTE module and the electron density calculated with the NLTE code. Also, the Saha density for the NLTE case is presented. For the NLTE case, the system is very close to Saha equilibrium in the center, while the electron density near the wall remains fairly high, despite the very small Saha density. This is because the electrons can diffuse from the high-electron density part to the low electron-density part.

figure 5.8, while the electron densities are presented in figure 5.9. The thermal conductivities obtained by the model are presented in figure 5.10.

5.2.2 The cause of the deviations between NLTE and LTE

The LTE model gives a much higher temperature at the center of the plasma than the NLTE model. For the LTE case, the thermal conductivity seems to be higher than for the NLTE case. This would mean that the temperature gradient should be lower, as the losses are proportional to both the thermal conductivity and the temperature gradient. Because the wall temperature is fixed, and a smaller gradient implies a smaller energy difference between wall and center, one would expect the LTE plasma to have a lower, and not a higher, temperature.

This apparent contradiction will be (partially) explained using figure 5.11. This figure is constructed from the raw data of figures 5.9 and 5.10. This figure shows a scaled value of the temperature difference between the LTE and NLTE treatment, a scaled value of the difference in heat conductivity between the LTE and NLTE treatment, and a scaled value of the difference in temperature gradient between NLTE and LTE.

Figure 5.11 shows a remarkable difference between the thermal conductivity calculated with the NLTE case and the LTE case near the wall. For the NLTE case, the conductivity is much higher. This is caused by a higher electron heat

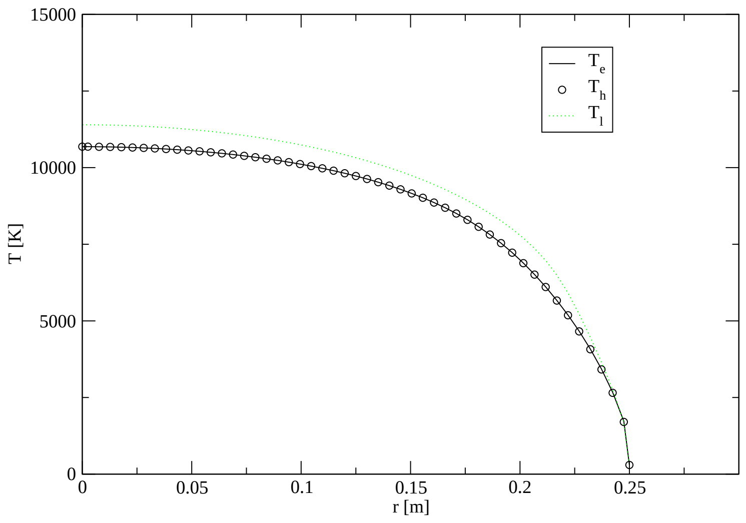

Figure 5.8: The temperature calculated with the LTE module T_l and T_e and T_h calculated with the NLTE module. The deviations start at 2 mm from the wall, and the full temperature difference is formed at 5 mm from the wall. The difference between T_e and T_h is very small

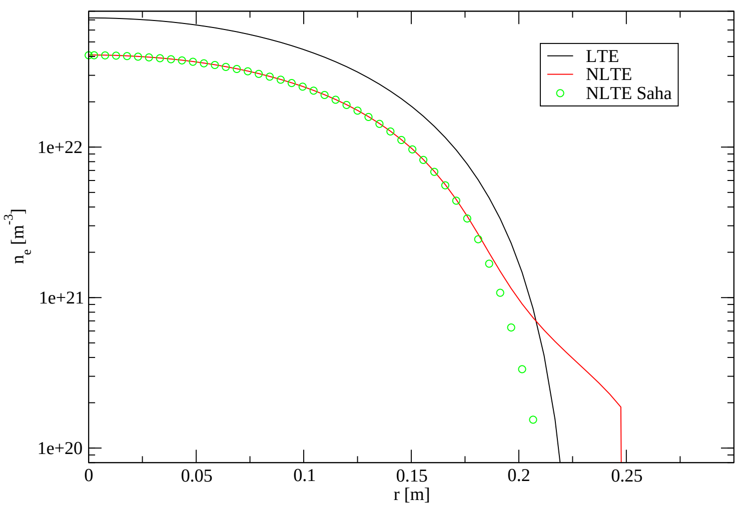

Figure 5.9: The electron density calculated with the LTE module and the electron density calculated with the NLTE code for. Also, the Saha density for the NLTE case is presented. For the NLTE case, the system is very close to Saha equilibrium in the center, while the electron density near the wall remains fairly high, despite the very small Saha density. This is because the electrons can diffuse from the high-electron density part to the low electron-density part.

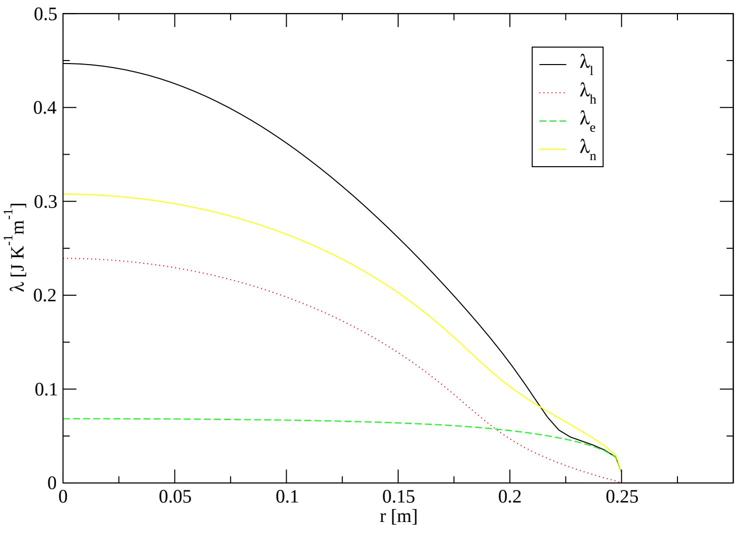

Figure 5.10: The heat conductivity in the LTE case(λ_l), and the electron (λ_e), heavy particle (λ_h) and total (λ_n) heat conductivity in the NLTE case. Note that at the wall, where the conductivity is the lowest, the NLTE heat conductivity is higher than the LTE heat conductivity. while in the rest of the plasma, it is the other way around.

conductivity for the NLTE case, which in turn is caused by the higher electron density near the wall for the NLTE case. This is caused by diffusion.

In the region where the thermal conductivity is higher for the NLTE case than for the LTE case, the temperature gradient is higher for the LTE case than for the NLTE case, and the NLTE temperature is lower. This is in agreement with the equation for the heat flux:

$$J_h \propto \lambda \vec{\nabla} T \tag{5.2}$$

which shows that for a fixed heat flux, $\vec{\nabla} T$ rises when λ drops. This is in agreement with the results in figure 5.11. Here, the higher thermal conductivity for the NLTE case causes a lower temperature gradient near the wall.

The larger temperature gradient in the LTE case is in a region where the temperature gradient is large compared to other regions in the plasma. This means, that the absolute effects of the differing temperature gradients are fairly large[1]. This can also be seen in figure 5.11: the difference in temperature between the LTE and NLTE approach originates close to the wall. The rest of the plasma, in which the thermal conductivity is higher for the LTE case than for the NLTE case, the temperature difference becomes smaller, but does not vanish entirely.

[1]A different way to look at this is by thinking of it as a classic bottleneck. The bottleneck for the heat conductivity is near the walls, so an increase or decrease in the heat conductivity there has a relatively larger impact.

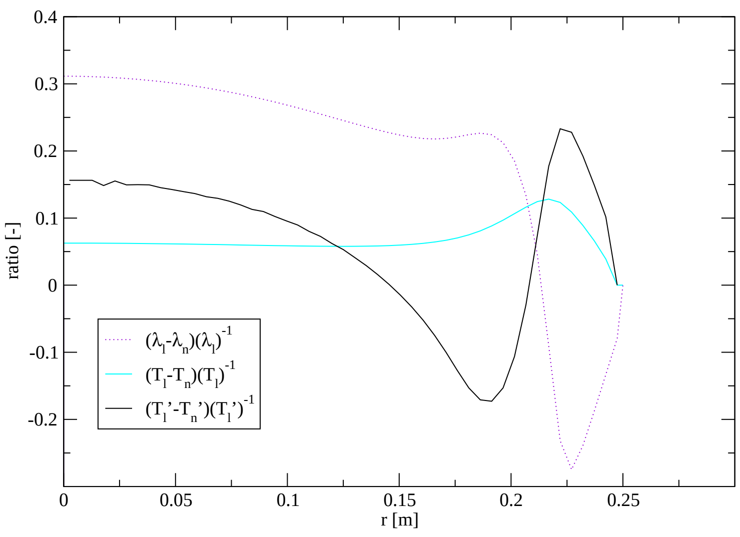

Figure 5.11: The dashed line in the plot is the difference in thermal conductivity between the LTE case (λ_l) and NLTE case (λ_n), scaled by dividing it by λ_l. Notice the very large difference in NLTE and LTE heat conductivity near the wall. The gray line is the difference between the LTE temperature T_l and the NLTE electron temperature T_h, scaled by dividing it with T_l. Notice the peak near the wall. Here, rather than in the center, the temperature difference between the NLTE and LTE originates. The black line is the difference between the temperature gradient in the LTE case and the temperature gradient in the NLTE case, scaled by dividing it with the temperature gradient in the LTE case. The large temperature gradient in the LTE case is in the same region in the plasma as the large gradient in the heat conductivity.

5.2.3 Conclusions

Models have been made to investigate the different central temperatures found when using the LTE and NLTE approach. Significant temperature difference between the NLTE and LTE case is found.

An important cause of the temperature difference between NLTE and LTE is found near the wall. Here, the NLTE approach, which allows deviations from Saha equilibrium, finds an electron density that is much higher from the electron density found using the LTE approach. This higher electron density causes a higher thermal conductivity, which reduces the large temperature gradient near the wall. This also reduces the temperature in other parts of the plasma.

Noticing that it makes a significant difference whether one uses the LTE or NLTE approach, it is important to find out whether the LTE or NLTE approach is better. Near the wall, the disturbances in the Saha balance become larger (See chapter 4). This cannot be handled by the LTE approach, which does not allow for these disturbances. Therefore, it is better to use the NLTE approach near the wall.

Although the difference in electron density near the walls is partially responsible for the difference in central temperature, the difference between the temperatures in the center of the plasma found with the NLTE and LTE approach cannot be fully explained by this. Therefore, another cause of these differences must be present. This is left for further investigation.

Chapter 6

Implementing pinching in PLASIMO

6.1 Introduction

The hollow cathode produces EUV radiation during the pinch phase when the magnetic field produced by the large current pulls the current-carrying particles together. This creates a very hot and dense plasma, where the highly ionized species emit radiation. In order to model this, it is essential that the physics behind the pinching effect is implemented.

As a first step to implement pinching in PLASIMO, the magnetic field produced by the current flowing through the plasma and the Lorentz force this magnetic field exerts on the current will be implemented. In this study we restrict ourselves to a cylinder symmetry. The Lorentz force is treated as a body force.

6.1.1 The Lorentz force in PLASIMO

In PLASIMO, the momentum balance is cast in the form of a ϕ-equation, being 3.7. In this equation, there is an external volume source term B_x. One of the extra forces is the Lorentz force. It will be our objective to write a model that calculates the Lorentz force for a given plasma configuration. This Lorentz force will be used by the momentum ϕ-equation to calculate the flow field. The momentum balance applies the Lorentz force in the nodal[1] point of the grid. Magnetization, which is the "trapping" of charged species by magnetic field lines, thus changing the transport properties, is not taken into account.

6.1.2 The axisymmetric grid

As stated, this study is restricted to cylindrical system. The cylindrical problem is 2D; the cylindrical system is represented by a body which is formed by

[1]The nodal point is the central (for a nonstretched grid) point in a control volume. Properties as temperature, composition, pressure, etc. are stored in these points, and assumed to be constant troughout the control volume

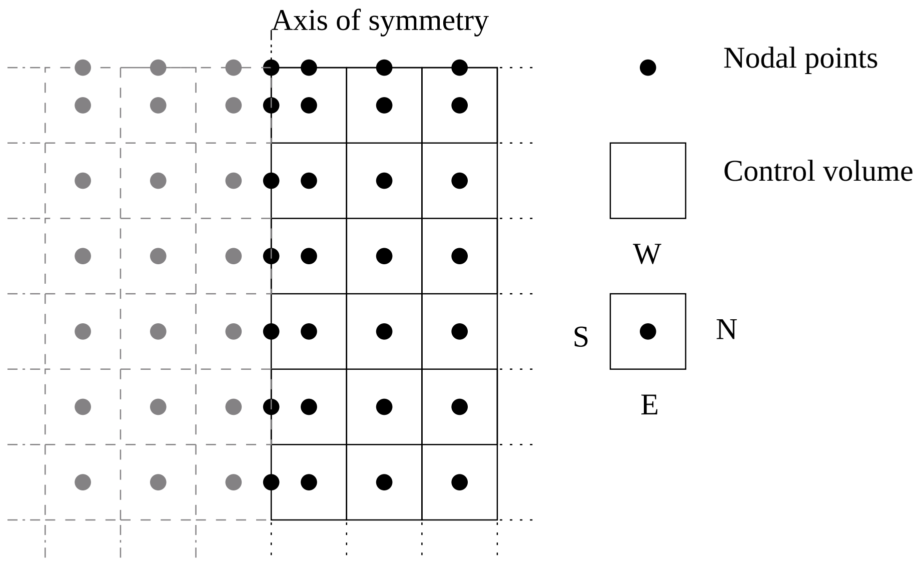

Figure 6.1: An axisymmetric grid. The 2D grid is rotated along the south boundary to obtain a quasi-3D cylinder. Note that the nodal ”points” are in fact rings. Also note that in the boundary control volumes the nodal point does not lay in the center of the control volume.

rotating a grid like the one in figure 6.1 around the south grid boundary, which represents the axis of the cylinder.

6.2 General theory

The pinching is caused by two effects: the magnetic field caused by the current (which consist of moving charged particles) and the Lorentz force that this magnetic field exerts on the current. The magnetic field will be treated first.

6.2.1 The magnetic field

Magnetic fields are created by moving charged particles. In a plasma, both the ions and the electrons can drift when they are exposed to an electric field. This causes a current to flow. For the case of an axisymmetric plasma with a current in the z-direction, it is convenient to use Amp‘ere’s law,

$$\oint \vec{B_\phi} \cdot d\vec{l} = \mu_0 I_{encl} \tag{6.1}$$

to calculate the ϕ-component of the magnetic field $\vec{B_\phi}$. μ_0 is the magnetic permeability of vacuum and I_{encl} is the current that through the surface enclosed by the integral.

The integration in (6.1) is done over a circle, with a radius equal to the distance to the nodal point, where we want to know the Lorentz force. This is illustrated by figure 6.2. Using this control surface, Ampère's Law becomes:

$$B_\phi = \frac{\mu_0 I_{encl}}{2\pi r} \tag{6.2}$$

With (6.2), the Lorentz force in a nodal point can be calculated.

The Ampère's Law control surface has current flowing through it. This current is the sum of the current flowing through the control volumes that are completely cut by the control surface and the control volume that is only partially cut by the control surface. This is the control volume in which the nodal point lies in which the Lorentz force is being calculated. Of the control volumes that are totally cut, all the current flowing trough them is taken into account. Of the control volume which is only partially cut, only the part of the current which is enclosed by the control surface should be taken into account. Assuming a constant current density in this control volume, this means that the part of the current x that is enclosed by the control volume is given by (6.3)

$$x = \frac{\pi r_p^2 - \pi r_s^2}{A} \approx \left(\pi r_p^2 - \pi(r_p - \frac{\Delta r}{2})^2\right)\frac{1}{A} = \left(\pi r_p \Delta r - \pi\frac{\Delta r^2}{4}\right)\frac{1}{A} \tag{6.3}$$

Here, r_p is the position of the nodal point, r_s is the position of the southern wall, A is the total surface area of the control volume and Δr is the width of the control volume. By summing the currents through the control volumes that are totally enclosed and the control volume that is partially covered, I_{encl} is obtained. The magnetic field can now be calculated with (6.2).

The value of these magnetic fields is calculated for all the control volumes in the grid. The implementation in the code will be explained in section 6.3. This gives the ϕ-component of the magnetic field over the whole grid. With this magnetic field the Lorentz force can be calculated.

6.2.2 The Lorentz force

The Lorenz force is a force exerted by a magnetic field on moving charges. It can therefore be interpreted as a force on a current. In PLASIMO, we implement the Lorentz force as a bulk force rather than implementing it on all the individual particles.

The Lorentz force of a magnetic field on a current can be expressed by:

$$\vec{F_l^*} = \vec{J} \times \vec{B} \tag{6.4}$$

Here, $\vec{F_l^*}$ is the Lorentz force density, $\vec{J}$ the current density and $\vec{B}$ the magnetic field. Note that we deal with vector quantities and vector products. In the case studied here (all current in the positive z-direction, the magnetic field in the positive ϕ-direction) $\vec{F_l^*}$ is in the negative r-direction. Its magnitude is then given by

$$F_l^* = JB_\phi \tag{6.5}$$

with F_l^* the Lorentz force density in the negative r-direction, and J is the magnitude of the current density in the positive z-direction. By integrating

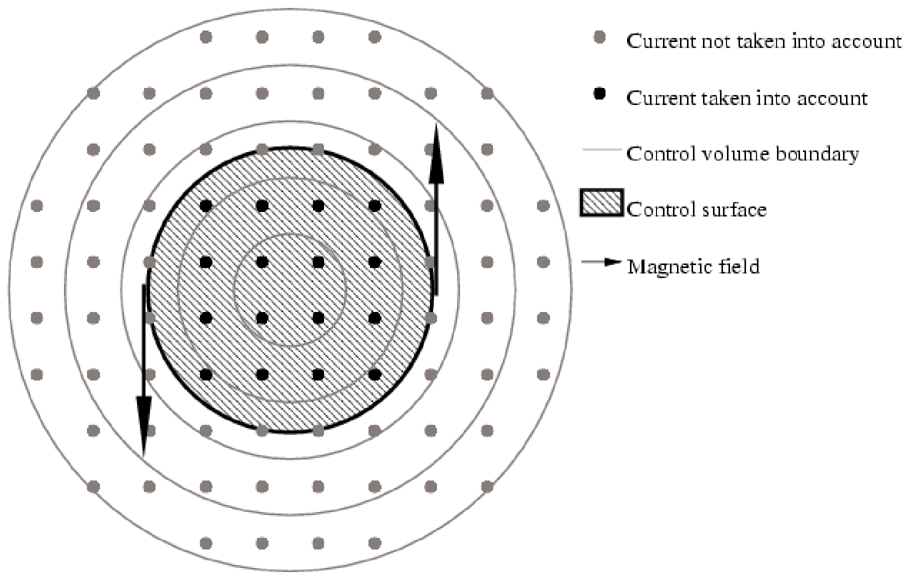

Figure 6.2: The gray lines represent the borders of the control volumes. These control volumes are rings. The dots represent the current flowing through control volumes. Of course, the current does not flows in discrete dots. The current flows out of the paper in this view. PLASIMO represents the current through a control volume by a current that flows trough the nodal point. (Note that the nodal points are in fact circles centered in the middle of the rings which are obtained by rotating the control volume.) The magnetic field produced by the current going through the circular control surface of Ampère's law is represented by the arrows. The control surface extends from the center to the nodal "point".

(6.5) over a control volume, we see that the force exerted on a single control volume is then equal to the magnetic field in the control volume times the current in that control volume times the length of the control volume. This is shown in figure 6.3. It schematically shows the Lorentz force exerted on a single control volume.

6.3 Implementation in the code*

In this section, the implementation of the Lorentz force in the code will be discussed. First, the structure of the EM class and the place where the Lorentz force is implemented will be discussed. Then, the actual implementation of the Lorentz force will be discussed.

6.3.1 Electromagnetism in PLASIMO

In PLASIMO, the electro-magnetic calculations are handled by an EM class and it derived classes. This class uses the plugins and settings provided by the user to do an EM calculation for the plasma. A part of the structure is shown in figure 6.4. The Lorentz force is implemented in a class `em_uniform1d` that is derived from the base class `plEM`. All the classes that are derived from

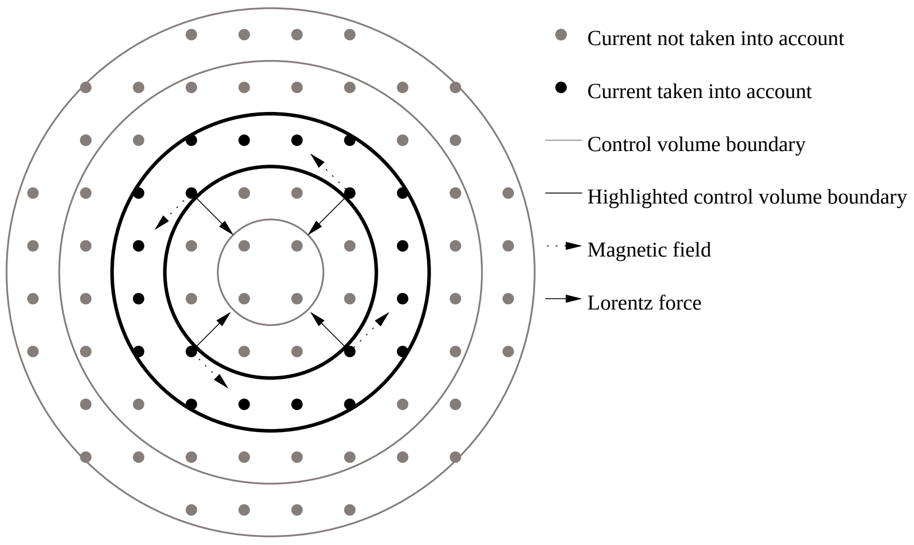
● Current not taken into account
● Current taken into account
Control volume boundary
Highlighted control volume boundary
Magnetic field
Lorentz force

Figure 6.3: The Lorentz force exerted on a control volume (highlighted) . The Lorentz force is only shown for the current on a few points. For the other points, it also points inward. The Lorentz force is stored in the nodal point (not shown).

`em_uniform1d` can now calculate the Lorentz force and take it into account. The results of the calculation are stored in `plEM` or `em_uniform1d`, and can be accessed by PLASIMO from there.

6.3.2 The implementation of the Lorentz force

In this Section, the implementation of the Lorentz force theory in the code will be discussed. The implementation is split in two files: `em_uniform1d.h` and `em_uniform1d.cpp`. First, the header file `em_uniform1d.h` will be discussed; then, the program body in `em_uniform1d.cpp`.

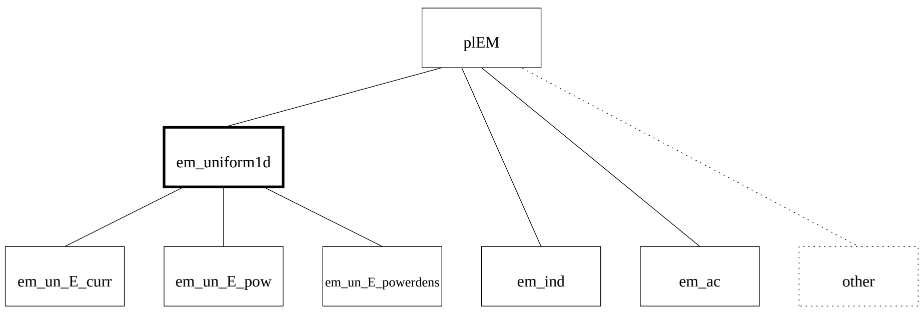

Figure 6.4: A part of the class structure of the EM calculation in PLASIMO. The class in which the Lorentz force is implemented is highlighted.

Table 6.1: The names of the most important variables in the program and the quantities they represent.

`i`	A counter that counts in the z-direction
`j`	A counter that counts in the r-direction
`N1`	The number of control volumes in the z-direction
`N2`	The number of control volumes in the r-direction
`l`	The length in the z-direction of the grid element
`r`	The length in the r-direction of the grid element
`pl`	The position on the z-axis of the nodal point of the grid element
`pr`	The position on the r-axis of the nodal point of the grid element
`length`	The length of grid in z-direction
`radius`	The length of grid in r-direction
`area`	The area of the grid element.
`F`	The Lorentz force
`I`	The current through a control volume
`J`	The current density through a control volume
`sumI`	The current enclosed by the Ampère's law control surface(Figure 6.2)

The header file

The program code of the header file can be found in appendix C. Calling the member `CalculateLorentzForce(void)` calculates the Lorentz force. The boolean `m_calc_lorentz`, supplied by the input file, determines whether the Lorentz force is actually calculated, or is set to 0. Another quality derived directly from the input file is `m_lorentz_urf`, which contains the underrelaxation factor used while calculating the Lorentz force. The ϕ-component of the magnetic field is stored in `CGridVar<REAL> m_B3`. This way, other PLASIMO modules can use it.

The body file

In the body file the actual calculation takes place. When the calculation starts, the constructor is called. The constructor gets the values of `m_calc_lorentz` and `m_lorentz_urf` from the input file.

When the member `CalculateLorentzForce(void)` is called, the Lorentz force calculation is started. If the boolean `m_calc_lorentz` is false, the Lorentz force is set to zero.[2] Otherwise, it is calculated. This calculation is started by declaring a number of variables. The meaning of these variables is explained in table 6.1.

The calculation is carried out by calculating the Lorentz force for increasing values of r for one particular z value, and then repeating this procedure for other values of z. When the calculation for one value of z is started, `sumI` is

[2]Note that just not calculating the Lorentz force and counting on the constructor to set it to zero doesn't work, because the values in the constructor may be overridden with a starting condition. If the user decides to employ a starting condition that has a nonzero value of the Lorentz force, it is not updated and stays at that nonzero value. This would mean that there is a Lorentz force while the Lorentz force is turned off; an obvious error.

set to 0. Then, the Lorentz force calculation for the central control volume is started. It starts by getting the length and the radius of the grid directly from the grid class. Next, the length and radius of the control volume under study are calculated.

The width of the control volume in r-direction and the position of the nodal point are obtained from the grid class. Next, the area of the control volume (see figure 6.2) is calculated. This is not trivial, as the control volumes at the boundaries need special treatment. Next, the current through the volume is calculated. The control surface covers only a part of the control volume: from the beginning of the volume to the nodal point. This part is calculated. The part of the volume covered by the control surface, x (`PartCovered` in the code), can be calculated with equation (6.3). Equation 6.3 is exact if the nodal points are in the middle of the control volume. This is the case if the grid is not stretched. Then, `sumI` is increased by x times the current flowing through the control volume. `sumI` now contains the current enclosed by the Ampère's law control surface. Now, Ampère's Law (equation (6.2)) is used to calculate B_ϕ. Again, the central control volume is excluded, as area of the control surface is 0. Next, the magnitude of F_l^* is calculated using equation (6.5). The reason why F_l^* is calculated rather than F_l, the Lorentz force is twofold: The Lorentz force per volume rather than the Lorentz force needs to be calculated for incorporation in the flow modules and it avoids potential problems with boundary control volumes. The value of `F` now represents the Lorentz force. The rest of the current through the control volume is now added to `sumI`. The calculation is concluded by using the newly calculated Lorentz force update the value of the Lorentz force. At this point the underrelaxation is incorporated.

When the calculation for the next control volume is started, `sumI` contains the total current that went through the previous control volume. In this calculation, it is again incremented by the current flowing through that control volume. This way, the current inclosed by the control surface in Ampère's Law can be obtained quite easily and naturally.

A final point of note is the boundary condition near the axis. As explained in the code, the Lorentz force has to be zero at the axis, but also at the control volume next to the axis. The latter is a result of the way the Lorentz force is incorporated in the momentum equation.

6.4 Testing the code

In this section, the code will be subjected to two tests. In the first test, a totally artificial, but simple, and needed simulation will be run to compare the results generated by the code with analytical results. In the second test, a plasma in which some pinching can be expected (a cascaded arc) will be run with and without a Lorentz force and the results will be compared.

6.4.1 Validation against an analytical solution

The code will be validated against an analytical solution. For this validation, a simple model is taken: a plasma with an uniform conductivity and an uniform electric field, and thus an uniform current I_0. An analytical solution will be derived for this special case, and it will be compared with the results.

Derivation of the analytical solution

The current enclosed by a circular control volume with an r smaller than the radius of the plasma R, is then given by:

$$\vec{I}_{encl} = \vec{I}\frac{r^2}{R^2} = \vec{J}\pi r^2 \tag{6.6}$$

with $\vec{J}$ the current density. If this current is substituted in equation (6.2), and I_0 is substituted for I, Equation 6.7 is obtained:

$$B_\phi = \frac{\mu_0 I_0 r}{2\pi R^2} \tag{6.7}$$

This can be used to calculate B_ϕ. With B_ϕ, the Lorentz force can be calculated.

The Lorentz force per unit of volume $\vec{F}_l^*$ is given by equation (6.4). With a current only in the z-direction, equation (6.4) can be combined with equation (6.7) to get an expression of F_l^* (The component of $\vec{F}_l^*$ in the negative r-direction) that depends only on J_0 (The z-component of the current density $\vec{J}_0$), and r:

$$F_l^* = \frac{\mu_0 J_0^2 r}{2} = \frac{\mu_0 I_0^2 r}{2\pi^2 R^4} \tag{6.8}$$

With a given I_0 and R, (6.8) can be used to calculate F_l^*.

Calculations

For three different situations, the Lorentz force in the configuration has been simulated with the Lorentz force module in PLASIMO. General features of the model are described in above in section (6.4.1). Three cases are studied:

- Model 1 has r=0.05 m, I_0=100 A (J=$1.27 \cdot 10^4$Am^{-2})
- Model 2 has r=0.05 m, I_0=10000 A (J=$1.27 \cdot 10^6$Am^{-2})
- Model 3 has r=0.005 m, I_0=100 A (J=$1.27 \cdot 10^6$Am^{-2})

The magnetic fields calculated by the Lorentz module for these models and the analytical value of the magnetic field are presented in figure 6.5. The match between the analytical and numerical result is excellent for all points, as it should be, because the implementation of the theory is exact for a grid in which the nodal points lie in the middle of the control volumes, as is the case here.

The Lorentz module has also been used to calculate the Lorentz force for these situations. These results are presented and compared to the theoretical results in figure 6.6. This figure clearly shows that the analytical and numerical results match excellently. This should be the case, because the implementation of the calculation in PLASIMO is exact for this grid.

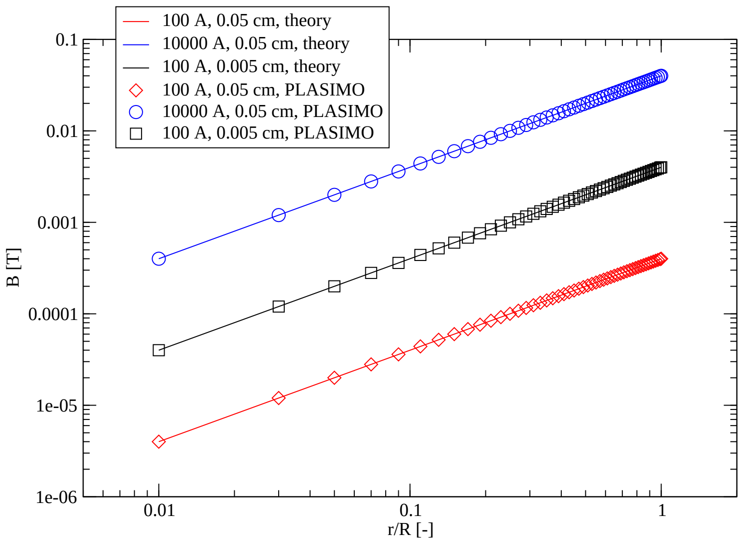

Figure 6.5: The magnetic field calculated with PLASIMO compared to the analytical solution for three different cases. On this double-logarithmic plot, it can easily be seen that B increases linearly with the distance from the center of the plasma. (A slope of 1 on a log/log plot means a linear dependence.)

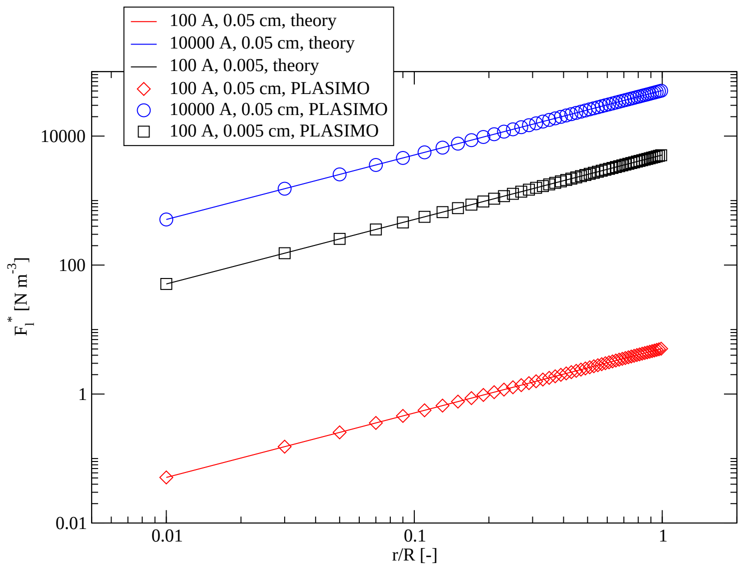

Figure 6.6: The Lorentz force calculated with PLASIMO compared to the analytical solution for three different cases. On this double-logarithmic scale, its can easily be seen that F_l^* increases linearly with the distance from the center of the plasma.

6.4.2 Testing the Lorentz force for a cascaded arc

After having tested the Lorentz force for synthetic test cases (section 6.4.1), it will now be tested for a real-life application of plasmas: a cascaded arc. The reason the arc was chosen is that it has a relatively high current density ($4 \cdot 10^7 \text{Am}^{-2}$), which is hoped to produce a small, but clear, influence on the results. First, a brief introduction to the cascaded arc will be given; then, the results of a calculation of the arc with the Lorentz force module enabled will be compared to a calculation of the arc with the Lorentz force module disabled.

The cascaded arc

The cascaded arc will not be described in detail here; a thorough explanation of the simulation of a cascaded arc is described in [9]. The simulation used in this case is similar, but not identical.

The model consists of a cylindrical grid that has a radius of 2 mm and is 6 cm long. The north boundary consist of a wall, that cools the plasma to T_h=500 K and T_e=6000 K, the east boundary is the outlet, at which a velocity which can be typified with a Mach number of 1 is set, the south wall is the symmetry axis, and the west wall is the inlet, through which 100 SCCS of Argon flows. The current of 50 A is in the z direction, and it is supplied by an uniform E-field. The only reaction taken into account is the ionization of argon. A NLTE treatment is used; thus, a complex system of ϕ-equations is used to determine the composition, although the source term (ionization of argon) is simple.

Two versions of this model are made: in the first, the Lorentz force is not taken into account; in the second, it is. If there is a difference between these models, it will be caused by the Lorentz force.

Results

A cascaded arc has been simulated with PLASIMO with and without a Lorentz force. The results will be compared. First, a graph of the Lorentz force density in the middle of the arc (z=3cm) will be presented in figure 6.7. The Lorentz force density rises when going outside from the center of the arc, just like in figure 6.6. Unlike the plasma in figure 6.6, the Lorentz force density reaches a maximum, and then drops again. This is because the conductivity is higher in the center, because the temperature is higher there. In combination with a uniform E-field, this means that the current drops near the edges. Because the B-field is inversely proportional to the radius and proportional to the current 6.2 enclosed, these two competing effects may cause a maximum to form.

The Lorentz force causes a compression of the plasma. This causes the pressure in the center of the plasma to become higher than the pressure at the edges. This can be seen in figure 6.8. It is clear that the pressure gradient is larger when the Lorentz force is enabled. The pressure drop from center to wall for the arc is 20 Pa larger if the Lorentz force is enabled. If the Lorentz force is

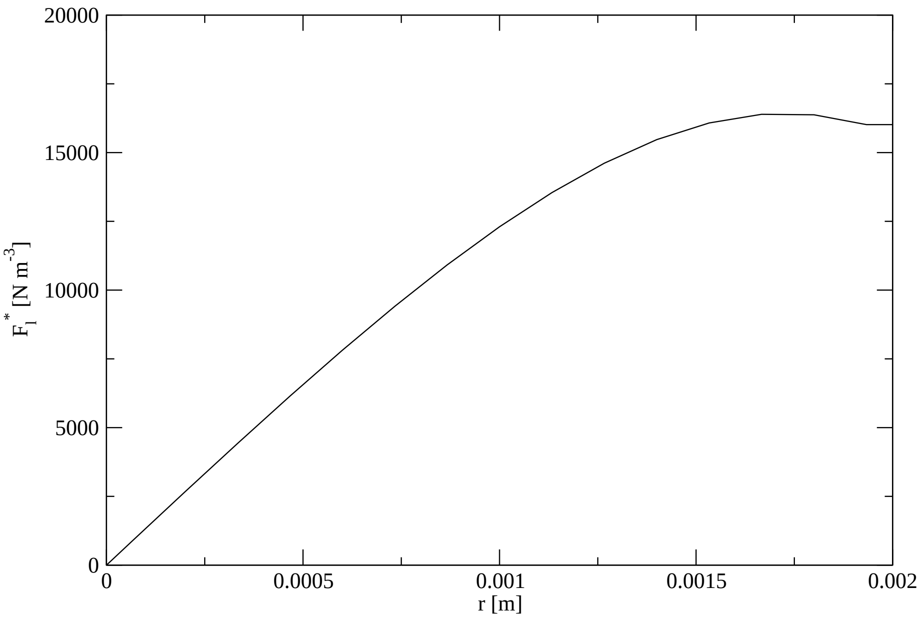

Figure 6.7: The Lorentz force density in the middle of the cascaded arc. It is pointed in the negative r-direction, thus compressing the plasma.

integrated over the radius, the calculated pressure increase is 15 Pa. This is in reasonable agreement with the pressure increase obtained by PLASIMO.

It has been attempted to simulate plasmas that have even larger larger Lorentz forces. It is generally possible to go to higher values of the Lorentz force. However, when the Lorentz force becomes a dominant factor in the problem, the convergence becomes troublesome.

As a final remark, it is noted that the plasma is not significantly magnetized. This can be seen by taking the maximum value of the magnetic field (5 mT) and calculating the Larmor frequency [1] for the electrons:

$$\omega_l = \frac{|q|B}{m_e} \tag{6.9}$$

which yields a frequency of 8.8$\cdot 10^8$ Hz. The electron-neutral collision frequency can be estimated with [15]; for this plasma, it is about 3$\cdot 10^9$ Hz. The electron-ion collision frequency ν_{ei} is given by

$$\nu_{ei} = 4\pi n_i \left(\frac{3\pi M_i}{8k_BT_i}\right)^{\frac{3}{2}} \left(\frac{Ze^2}{4\pi\epsilon_0 m_e}\right)^2 ln\left(\frac{4\pi}{3}\frac{(\epsilon_0 k_B T_e)^{\frac{3}{2}}}{e^3\sqrt{n_e}}\right) \tag{6.10}$$

Here, M_i is the ion mass and Z is the ion charge. This expression, combined with the results from PLASIMO, gives an ν_{ei} of $2 \cdot 10^{10}$ Hz. The sum of these frequencies is much larger than the Larmor frequency, meaning that the electrons collide so often that they do not follow the magnetic field lines.

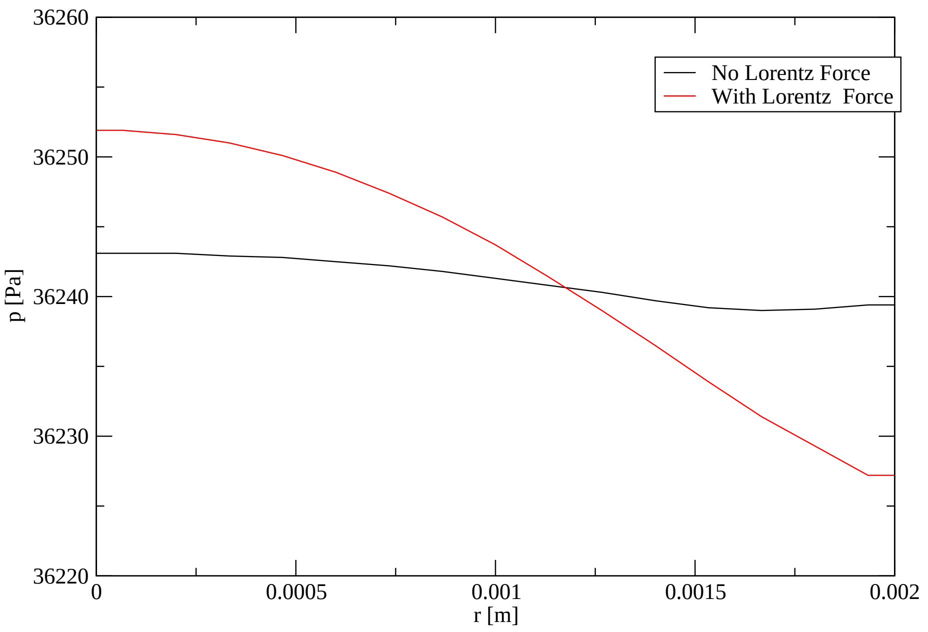

Figure 6.8: The pressure profile of a cascade arc at z=3 cm. The simulation has been carried out for two settings, one with a the Lorentz force enabled and the other with the Lorentz force disabled. has been carried out while taking the Lorentz force into account and also while not taking it into account.

6.5 Conclusions

The Lorentz force has been implemented as a bulk force. This is an adequate approach if the reactions in the plasma are sufficiently fast, so the Lorentz force does not cause large deviations in the composition of the plasma. The approach is currently only valid for a cylindrical geometry. For this geometry, it has been tested for an artificial situation. The results agreed with the theory. It has also been tested in a cascaded arc. In this case, the results were consistent with the estimations based on the Lorentz force theory.

In order to simulate plasmas in which pinching becomes dominant, it may be necessary to make further additions to the code. A stronger coupling between the Lorentz force and the pressure may help convergence. Also, it may be necessary to take into account the effect of magnetization on the transport coefficients. This will require the use of anisotropic transport coefficients which are not yet implemented in PLASIMO.

Chapter 7

Simulating plasma decay in a hollow cathode discharge

7.1 Introduction

In this section, the decay of the plasma in the hollow cathode will be modeled. For this, a substantial extention to the diffusion calculation in PLASIMO must be constructed, because Fick's law of diffusion does not hold for pressures as low as in the hollow cathode, as we will see. When this extention is made, a model of the decay of the hollow cathode can be assembled using PLASIMO. This model can then be used to simulate the decay of the hollow cathode.

7.1.1 Diffusion

Diffusion to the walls is a common loss process of electron-ion pairs in discharges, in particular those at low pressure and of small dimensions. For most discharges, the classic diffusion theory, enhanced to account for ambipolar diffusion, is a good approximation.

Diffusion theory breaks down, as will be explained below, when the mean free path of particles becomes of the same magnitude or larger than the size of the vessel. In the regime where the mean free path is much larger than the typical dimensions of the vessel, Knudsen free molecular flow theory can be used to describe the electron-ion drift to the wall. In the intermediate regime, a combination of both methods should be used. In this chapter, an approximate this will be presented, implemented, and validated.

7.1.2 Contents of this chapter

In section 7.2, the theory will be outlined. Diffusion and Knudsen flow will be treated, and a model by Mason and Malinauskas [16] to incorporate them both simultaneously is presented. In section 7.3, the implementation of this theory in PLASIMO will be discussed. In section 7.4, the improved treatment of diffusion and Knudsen flow will be applied to a hollow cathode discharge, which operates at the boundary of the two regimes. This means, that neither can be neglected,

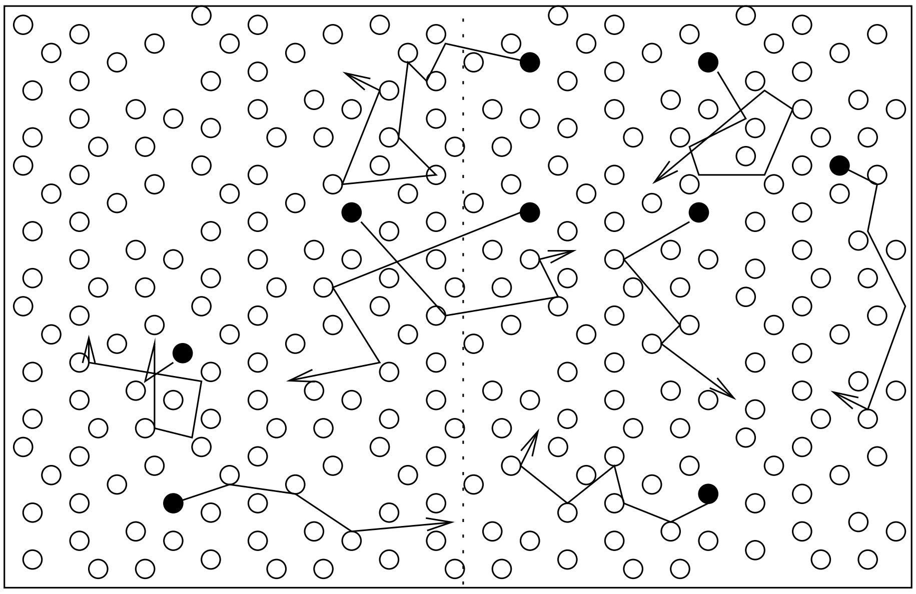

Figure 7.1: A schematic, kinetic view on diffusion. The random motion of particles will eventually create equal densities of the species under consideration (the black dots) on the left and right. The molecules collide more frequently with the background gas than with the vessel walls. This means that the effective distance they travel is determined by a random-walk process. This distance is proportional to the square root of the time.

requiring the full combined diffusion/Knudsen flow model. Analytical solutions will be used to validate the answers. In section 7.5 it will be investigated what a good model for the hollow cathode is. In section 7.6, the influence of a few parameters will be investigated by varying them. In section 7.7, the conclusions will be presented.

7.2 Theory

7.2.1 Diffusion

Diffusion is the movement of particles through a background medium, here a gas, driven by a concentration gradient. This effect is illustrated by Figure 7.1, which shows that diffusion is related to the random walk of molecules. The result is that differences in concentration will be minimized. In the case of Figure 7.1, the diffusing species has a higher density on the right than on the left. This means, that the random motion of the particles will cause more particles of this species to go from right to left than form left to right. In the end, this will cause equal densities left and right.

The temporal change in the density N of a species due to diffusion through a background gas is given by [17]. This is in fact a particle balance without sources.

$$\frac{\partial N}{\partial t} = -\vec{\nabla} \cdot \vec{\Gamma} = \vec{\nabla} \cdot (D_s \vec{\nabla}) N \tag{7.1}$$

Here, D_s is the diffusion coefficient, t the time and $\vec{\Gamma}$ the flux. Notice that the diffusion time scales with the square of the length. This can be seen by approximating the derivatives with divisions, yielding:

$$\frac{1}{t} \propto D_s \frac{1}{L^2} \tag{7.2}$$

This scaling is consistent with the random walk of molecules, where the effective distance traveled scales with the square root of time.

Ambipolar Diffusion

In this chapter, we are interested in diffusion in plasmas. Plasmas contain electrons and ions. These cannot diffuse independently, as this would cause very large space charges and thus electric fields to be build up. As a consequence, electrons and ions will be pulled to each other. This means, that they cannot get apart too much. This process is known as ambipolar diffusion. The lighter, and often hotter, electrons are much more mobile than the ions. They diffuse faster, and thus pull the ions with them. In this case, the plasma diffuses with an ambipolar diffusion coefficient D_a that is equal to [17]:

$$D_a = D_i \left(1 + \frac{T_e}{T_h}\right) \tag{7.3}$$

Here, D_i is the ion diffusion coefficient, T_e is the electron temperature and T_h is the heavy particle temperature.

7.2.2 Knudsen flow

The failure of diffusion theory

In order to explore the boundaries of the validity of the diffusion theory outlined in section 7.2.1, some attention will be paid to the diffusion coefficient D_s for which we follow the derivation given in [14]. The result is:

$$D_s = \frac{\pi}{8} \lambda^2 \nu \tag{7.4}$$

Here, λ is the mean free path and ν it he collision frequency between the diffusing species and the background gas [1]. Using

$$\lambda = \frac{v_{th}}{\nu} \tag{7.5}$$

with v_{th} the thermal velocity, equation (7.4) can be rewritten to

$$D_s = \frac{\pi}{8} \lambda v_{th} \tag{7.6}$$

[1]The prefactor in (7.4) is consistent with using an average thermal velocity $v_{th} = \sqrt{\frac{8k_BT}{\pi m}}$; it is also common to use the root mean square thermal velocity $v_{th} = \sqrt{\frac{3k_BT}{m}}$; this gives a prefactor of $\frac{1}{3}$. The actual value should be obtained by integrating over the velocity distribution

Normally, the mean free path λ depends on the density N and the cross section σ [14]:

$$\lambda \propto \frac{1}{N\sigma} \tag{7.7}$$

Here, σ is the cross section. This equation is valid when the mean free path is determined by collisions with background gas particles. When N becomes very small (low pressure), the mean free path predicted by equation (7.7) will get very large. It may thus become larger than the typical dimensions of the vessel. This means the particles will collide more often with the wall than with the background gas. The collisions of the particles with the wall cause the size of the vessel places to place an effective maximum on the mean free path. Here, the diffusion theory breaks down. Qualitatively, (7.6) changes to

$$D_k \propto L v_{th} \tag{7.8}$$

with D_k an effective diffusion coefficient. This gives a typical scaling in this regime of:

$$\frac{1}{t} \propto D_k \frac{1}{L} \tag{7.9}$$

as opposed to the scaling for diffusion in (7.2).

Knudsen flow

Knudsen flow, in contrast to diffusion, is the free movement of particles, unhindered by the background gas. This will be explained using Figure 7.2.

Figure 7.2 is similar to figure 7.1. On the right there are more particles than on the left. Due to the random motion of the particles, more particles will go from the right to the left than form the left to the right; thus, eventually, the amount of particles will become equal on the left and right side.

The main difference between Knudsen flow and diffusion is the way the particles move through the vessel. With Knudsen flow, the particles are essentially free to move through the vessel, whereas in the diffusion case, the particles describe a random walk through the background gas. In the case of free movement, the particles travel a distance that is proportional to their speed and time. In a random-motion case, the distance is proportional to their speed and the square root of time.

After this kinetic explanation, the equations that describe Knudsen flow will be discussed. The flux corresponding to a Knudsen flow Γ from a gas to a vacuum is given by [16]:

$$\Gamma = w v_{th} N \tag{7.10}$$

with w a dimensionless probability factor and v_{th} the average thermal velocity. The particle balance (7.1) can be rewritten to

$$\frac{\partial N}{\partial t} = -w v_{th} \vec{\nabla} N \cdot \vec{e} \tag{7.11}$$

with $\vec{e}$ the unit vector normal on the surface through which the diffusion takes place.

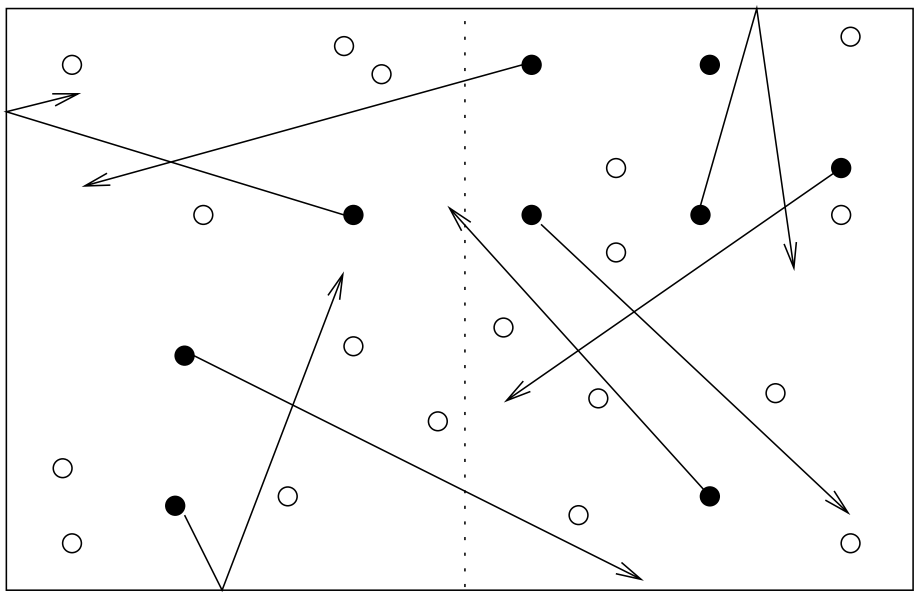

Figure 7.2: A schematic, kinetic view on Knudsen flow. The random motion of particles will eventually create equal densities of the species under consideration (the black dots) on the left and right. The molecules collide more frequently with the vessel walls than with the background gas. This means that the effective distance they travel is not determined by a random-walk process, because the mean free path is much larger than the distance the particles must travel to cross the vessel. This distance is therefore linearly proportional to the time.

For an infinitesimally thin orifice, a canonical example of Knudsen flow, w is $\frac{1}{4}$. This factor comes from the fact that not all particles travel in the direction of the plane. It can be rigorously derived by integration over the Maxwell distribution. The Maxwell distribution is given by [14]:

$$f(\vec{v}) = \left(\frac{m}{2\pi k_B T}\right)^{\frac{3}{2}} e^{\left(-\frac{mv^2}{2k_B T}\right)} \tag{7.12}$$

with $f(\vec{v})$ the particle density in velocity, m the mass of the particle and T the temperature. The average velocity in the positive v_z direction $\overline{v}_z$can be obtained by integrating v_z over half of the velocity space (only the positive half contributes), weighed with $f(\vec{v})$. This can then be used to calculate the flux. This gives the integral [14]

$$\Gamma_z = N\left(\frac{m}{2\pi k_B T}\right)^{\frac{3}{2}} \int_0^{2\pi} \mathrm{d}\phi \int_0^{\frac{\pi}{2}} \cos\theta \sin\theta \mathrm{d}\theta \int_0^{\infty} e^{\left(-\frac{mv^2}{2k_B T}\right)} v^3 \mathrm{d}v \tag{7.13}$$

Here Γ_z represents the effective fraction of particles passing through the z=0 plane. Evaluating this integral yields that

$$\Gamma_z = N\frac{1}{4}\sqrt{\frac{8k_b T}{\pi m}} = \frac{1}{4}N v_{th} \tag{7.14}$$

with N the thermal velocity. The prefactor of $\frac{1}{4}$ is the result of the integrations over ϕ and θ, while the factor of $\sqrt{\frac{8}{\pi}}$ comes from the integration over the velocity.

It is customary to use a Knudsen diffusion coefficient D_k, analogous to a standard diffusion coefficient, to describe the loss process due to Knudsen flow.

$$\frac{\partial N}{\partial t} = \vec{\nabla} \cdot (D_K \vec{\nabla} N) \tag{7.15}$$

This equation can, contrary to (7.11) be cast into a ϕ-equation form. The Knudsen diffusion coefficient depends mainly on v_{th}. This can be made explicit by expressing it as:

$$D_k = \frac{4}{3} K_0 v_{th} \tag{7.16}$$

The Knudsen factor K_0 is difficult to calculate from first principle for a practical geometry. It is of the order of magnitude of the typical vessel dimension. If equation (7.11) and (7.16) are to be consistent, K_0 should be taken as

$$K_0 = f_{Kn}\Lambda = \frac{3}{16}\Lambda \tag{7.17}$$

with Λ the gradient length of the density. The factor f_{Kn} is a geometry factor. The astute reader will notice that this implies an anisotropic diffusion coefficient, as Λ is generally not the same for all directions. As the Knudsen diffusion is determined by plasma-vessel interactions rather than plasma-gas interactions, the generally anisotropic vessel may introduce anisotropies not introduced by a background gas.

Ambipolar Knudsen flow

In a plasma, the electrons and ions generally do not diffuse independently, as this would cause huge space charges. Instead, the electrons pull the ions with them. The same thing goes for Knudsen flow. This is taken into account by using the Bohm velocity. The Bohm velocity is basically the ion thermal velocity, calculated substituting the electron temperature for the ion temperature [2]. This is analogous to the treatment of the sheath in [14]. This way, a consistent description of vessel and wall can be achieved.

7.2.3 The resistor model

In [16], a model is given that can be used to account for both Knudsen and diffusion at the same time. In this model, diffusion and Knudsen flow are both considered a resistor with a value $\frac{1}{D}$, with D the appropriate diffusion coefficient.[3] These resistors are put in series. This means that a particle is hampered

[2]One could consider whether the ion thermal velocity should also be taken into account, for instance by substituting the ion temperature with the sum of ion temperature and electron temperature. This has not been done, for two reasons: 1. It is consistent with PLASIMOs treatment of the sheath. 2. The difference is small, as plasmas in the Knudsen flow regime are likely to be very far from LTE, so the ion temperature is much lower than the electron particle temperature.

[3]The diffusion coefficient is the analog of a conductance, while a resistor is the analog of a friction.

in its movement by both of these effects. Basic electricity theory yields the effective resistance $\frac{1}{D_f}$ of two series resistors. The effective conductance D_f is given by:

$$D_f = \frac{D_s D_k}{D_s + D_k} \tag{7.18}$$

It can easily be seen that this expression yields the correct results for the limiting cases where only one of the processes is significant.

7.2.4 An analytical approach to the decay in the hollow cathode

In order to validate our model, we will compare the results generated by PLASIMO and the results obtained analytically. In this section and appendix D, an analytical solution will be derived, using an analytical description of the decay in a cylindrical vessel.

Generally, the drop in the electron density n_e as a result of diffusion in such a vessel is described by this equation:

$$\frac{\partial n_e}{\partial t} = \vec{\nabla} \cdot (D\vec{\nabla} n_e) \tag{7.19}$$

with D a diffusion coefficient, of which the value will be specified later. This value depends on the conditions. For example, (7.18) could be used. The boundary conditions are that the ions and electrons move unhampered to the wall with the Bohm velocity. This takes into account the fact that the hot, fast electrons pull the heavy, slow ions. Because this velocity is very high and because the distance traveled is short, it is assumed that this does not slow down the rate of diffusion. Thus, the density at the wall is set to 0.[4] [5] If the positions of the walls are at $r = b_r$ and $z = b_z$, then the boundary conditions at the walls are:

$$n_e(b_r, z, t) = n_e(r, 0, t) = n_e(r, b_z, t) = 0 \tag{7.20}$$

The last boundary condition comes from the symmetry axis:

$$\frac{\partial n_e(0, z, t)}{\partial r} = 0 \tag{7.21}$$

To solve equation (7.19), the assumption will be made that the diffusion coefficient is constant and equal to D_f (cf. (7.18)), which can be calculated using the theory of diffusion and Knudsen flow outlined above. This changes equation (7.19) to:

$$\frac{\partial n_e}{\partial t} = (D_f \vec{\nabla}^2 n_e) \tag{7.22}$$

[4] While this is called the "density at the wall", this is actually a bit misleading. The domain stops just before the physical walls.

[5] For most plasmas, this is a fine approach, as the diffusion to the wall is much slower than the free fall through the sheath. However, the plasma currently under study is also near the free fall regime; thus, in this case, the wall density may be fairly large. This is an obvious flaw in the analytical approach. See Section 7.4 for a better treatment of the wall. This treatment, unfortunately, is too complex to use in an analytical model

This is a standard problem, which is easily solved. The solution is derived in appendix D. The result is:

$$\tau = \frac{1}{D_f\left(\left(\frac{2.40}{b_r}\right)^2 + \left(\frac{\pi}{b_z}\right)^2\right)} \tag{7.23}$$

7.2.5 Conclusion

As explained in subsection 7.2.2, the description of Knudsen flow is approximate. Therefore, the results of the Knudsen flow calculation are likely to be less accurate than the results of the diffusion calculation. However, they are still extremely valuable. When used in the resistor model, it prevents the effective diffusion coefficient from reaching unphysically large values, which would yield erroneous results in the calculation.

7.3 Implementation★

7.3.1 The design criteria

In section 7.2, a new method was proposed to extend the validity region of classical diffusion theory. The new `plKnudsenDiffusion` class should do a calculation of the diffusion coefficient, based on the theory of section 7.2. It is the aim to make this calculation as general as possible. The class should be able to calculate either the Knudsen diffusion coefficient, or the classical diffusion coefficient, or the effective diffusion coefficient that results when both are taken into account. This allows the user to select which process he wants to use, giving the user a freedom he never had in an experiment.

Furthermore, a choice should offered to use ambipolar or standard diffusion. Standard diffusion occurs in nonionized gasses, where the electrons cannot influence the diffusion of the heavy particles. The same choice is offered for Knudsen diffusion: the thermal speed should be selected for a gas, while the Bohm velocity is appropriate for a plasma. These two choices can be made by the user by using the input parameters.

Another input parameter is the geometry factor, which would be supplied by the user. The module does not calculate it for the user, instead, the user is free to choose the value he deems most appropriate.

The final input parameter is the diffusion length. While this is normally equal to half the smallest plasma dimension, this is not always true. Therefore, the user is free to choose a value.

Summarizing, the `plKnudsenDiffusion` class should take as input parameters:

- The geometry factor
- The diffusion length

- The type of diffusion (Disabled[6], Ambipolar or Standard)
- The velocity to be used for Knudsen flow. (Disable Knudsen flow, Bohm velocity or thermal velocity)

It should use these parameters to calculate a value for the diffusion coefficient.

7.3.2 The code

When implementing this in PLASIMO, the modular structure of PLASIMO allowed the reuse of very large pieces of code, making this a relatively simple task.The `plKnudsenDiffusion` class is a derived class from `plAmbipolarDiffusion`. Thus, all the code of the diffusion is readily available and does not need to be reprogrammed. The input parameters are processed by the input parser. The appropriate calculation is made using these input parameters.

The implementation will be explained, using the source code of the `plKnudsenDiffusion` class, which is presented in appendix E.

The code starts with the constructor. This constructor first sets the size of a few vector members of the class. Then it calculates K_0 from the data in the input file using (7.17). After this, it translates the different modes of diffusion and Knudsen flow to two numbers: `m_DiffusionMode` and `m_KnudsenMode`. A parser exception will be thrown if one tries to disable both diffusion and Knudsen flow as this would be totally unphysical and fatal for the calculation.

Next, the virtual function `DoCalculate` is declared. This does the actual calculation. The private member `m_ParticleMasses` stores the particle masses, `m_DDiffusion` stores the diffusion coefficients (one for each species) and `m_DKnudsen` the Knudsen flow coefficient (again, one for each species). The real number `m_K` is K_0. The members `CalculateDDiffusion` and `CalculateDKnudsen` calculate the diffusion and Knudsen flow coefficient, respectively.

After the declarations, the actual implementations of the functions follows. The public member `DoCalculate` does the calculation of the diffusion coefficient. If both diffusion and Knudsen flow are enabled by the user, these both get calculated. After that, they are “added” as in equation (7.18). If diffusion is disabled, only Knudsen flow gets calculated. If Knudsen flow is disabled, only diffusion gets calculated.

`CalculateDDiffusion` calculates the diffusion coefficient. It either uses ambipolar diffusion or standard diffusion, depending on the users’ choice.

`CalculateDKnudsen` calculates the Knudsen flow coefficient. Based on the type of particle and the type of diffusion to be used, either the thermal velocity of the particle (when the thermal velocity mode is used for all particles or only for the neutral particles in the Bohm velocity mode) or the electron temperature (in

[6]This disables the limitation of the loss rate by diffusion; diffusion losses are infinitely fast. This is only meaningful when Knudsen flow is enabled.

case of ions when Bohm velocity is used). Next, the mean velocity is calculated. This is then used in equation (7.16) to calculated the Knudsen flow coefficient.

7.4 Validation

7.4.1 Introduction

In this section, the code will be validated by applying it to a model of the decay in a hollow cathode. A wide range of models could have been used, the hollow cathode is chosen for convenience, as a model for the hollow cathode is needed later anyway. First, an over-simplified model of the hollow cathode discharge will be made. This model will be discussed in section 7.4.2. The various Knudsen submodels will be verified by running this simple model of a hollow cathode discharge for the various options provided by the `plKnudsenDiffusion` class in section 7.4.3. The resulting diffusion coefficient is then compared to analytical calculations of that diffusion coefficient. This is in fact the main point of this validation, as the only thing added to the code is a new way of calculating the diffusion coefficient. After validating the model this way, the diffusion time, calculated from an analytical model, will be compared to the diffusion time from numerical results. The conclusions will be presented in section 7.4.4. The insights gained in this chapter will be used in chapter 7.5 to construct an accurate model of the hollow cathode discharge.

7.4.2 A simple model for the decay in the hollow cathode discharge

The system under study is a hollow cathode discharge. Here, only the hollow cathode part of the device will be studied. It consists of a cylinder with a height of 16 mm and a radius of 16 mm. This device is filled with 10 Pa of Xe, through a hole in the top. (See figure 2.1). This hole has a diameter of 4 mm. In this case, however, the hole is treated as a normal wall. This is done to let the numerical and the analytical model be a similar as possible. The plasma is estimated to have an electron density of $1.1{\cdot}10^{20}$ m^{-3}. The electron temperature is estimated at 8000 K, while the gas temperature is estimated at 1000 K. The wall temperature is set equal to the heavy particle temperature. Thus, the boundary condition for T_h is a Constant Dirichlet condition with a temperature of 1000 K. For the electron temperature, the boundary temperature is set to 8000 K. In reality, the electron temperature is not set by the wall. By doing this, we keep the diffusion coefficient, that depends on T_e, constant during the simulation. For the Knudsen flow, the geometrical factor f_{Kn} is estimated at 0.1875 [7], while the gradient length is set to 8 mm. The simulation is performed on a 40 by 40 grid; this means the control volumes are rings with a square cross section. Setting the ion density at the wall to 0 m^{-3}, similar to the approach used in the analytical model, did not work in practice; instead, a very fast loss process was specified at the wall, by speeding up the normal loss process, where

[7]One might be tempted to set f_{Kn} to 0.375 to account for Knudsen flow in two directions. This is, however, not correct, because this effect is already incorporated in the simulations, and taking it into account twice will yield an erroneous result.

Table 7.1: The diffusion coefficient D_f in ms^{-2}, as calculated by PLASIMO, compared to the theoretical value. The different ways of calculating the diffusion coefficient are in separate columns, whereas the different ways of calculating the Knudsen flow coefficient are placed in separate rows. The standard diffusion coefficient is the diffusion coefficient that is obtained when the electric attraction between ions and electrons is ignored.

	disabled		ambipolar		thermal	
	PLASIMO	analytical	PLASIMO	analytical	PLASIMO	analytical
disabled	-	-	1.68	1.70	$1.87 \cdot 10^{-1}$	$1.88 \cdot 10^{-1}$
Bohm	2.27	2.30	$9.9 \cdot 10^{-1}$	$9.6 \cdot 10^{-1}$	$1.7 \cdot 10^{-2}$	$1.8 \cdot 10^{-1}$
thermal	$8.1 \cdot 10^{-1}$	$8.4 \cdot 10^{-1}$	$5.4 \cdot 10^{-1}$	$5.4 \cdot 10^{-1}$	$1.5 \cdot 10^{-1}$	$1.6 \cdot 10^{-1}$

ions and electrons recombine with a speed limited with by the Bohm velocity with a factor of 20.

7.4.3 Results of the validation

The three possible diffusion modes (disabled, ambipolar and standard) have been combined with the three Knudsen flow modes (disable, Bohm velocity and thermal velocity). The models are started up, and tracked until the central electron density drops to $1 \cdot 10^{14}$ m^{-3}. The time to reach various densities has been recorded, as is D_f.

The diffusion coefficient

The value of D_f of Xe$^+$ has been recorded when the Xe$^+$ density is $1 \cdot 10^{19}$ m^{-3}. This has been compared with a theoretical value of D_f, calculated using the equations in section 7.2. Table 7.1 shows that the values of D_f calculated with PLASIMO closely match the values calculated with theory. This is an indication that the theory is implemented correctly. Note that the new class solely calculates the diffusion coefficient. This means that the diffusion coefficient is the key parameter to be checked in the validation; the other results, while interesting, are less relevant for checking whether the diffusion calculator is implemented correctly, but rather for showing flaws in the model and illustrating the difference between the various diffusion calculator options.

Decay times

The decay process for these eight modes has been graphically displayed in figure 7.3. It shows that the decay is mainly exponential, which is consistent with both equation 7.1 and 7.15. Note that there are deviations from this behavior during the beginning of the decay. These are caused by three effects:

- The dissimilarity between the flat starting profile and the parabolic profile in the z-direction and the Bessel profile in the r-direction caused by diffusion. It takes time to build up this profile, and in this time, the central density does not drop. This is related to the fact that the higher-order modes decay first (appendix D).

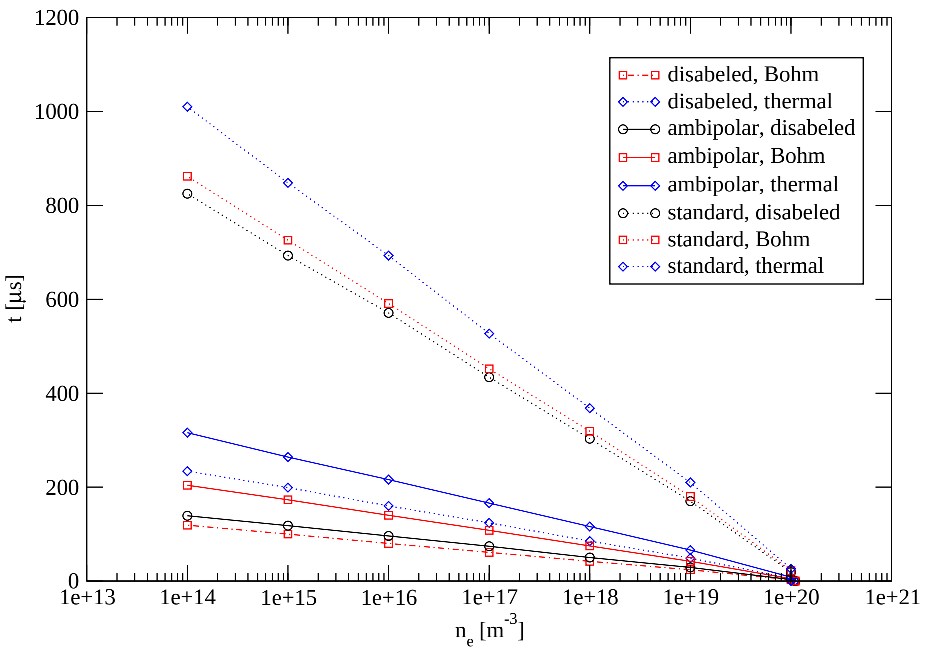

Figure 7.3: The decay in the hollow cathode, calculated by various methods. In the legend, the first word refers to the method used for calculating diffusion, the second word refers to the method used to calculate Knudsen flow.

- A significant part of the plasma is ionized. This means that there are less particles that hamper diffusion, and therefore, the diffusion coefficient will be larger, causing a faster drop.
- The wall condition is not an homogeneous Dirichlet condition; instead, a very fast loss process is specified. This does mean, that the loss will be slowed somewhat by the time this loss process takes.

The results clearly show that the first effect is the most important effect; the decay is much slower during the initial stages. The decay is very slow when standard diffusion is used. This is consistent with the fact that standard diffusion is much slower than ambipolar diffusion in this case. The results from ambipolar diffusion, which are a better description of reality than standard diffusion, show that the Knudsen flow process has a significant impact on the decay time, but also that the Knudsen flow is not the dominant process.

The values of τ, obtained by PLASIMO, are compared to the results of the analytical approach presented in section 7.2.4. The result of both these calculations can be found in table 7.2. Table 7.2 clearly shows that the diffusion time calculated with PLASIMO is generally larger than the diffusion time calculated with the analytical model. This is caused by the boundary condition. This has been investigated by modifying the model so that it uses the aforementioned boundary condition, but the ions and electrons flow to the wall with 20 times the rate determined by the Bohm velocity. The results generated by this model are quite close to the analytical results, where the density at the boundary is

Table 7.2: The decay time τ in s, as calculated by PLASIMO, compared to the theoretical value. The different ways of calculating the diffusion coefficient are in separate columns, whereas the different ways of calculating the Knudsen flow coefficient are in separate rows.

	disabled		ambipolar		standard	
	PLASIMO	Analytical	PLASIMO	Analytical	PLASIMO	Analytical
disabled	-	-	$1.0 \cdot 10^{-5}$	$5.5 \cdot 10^{-6}$	$5.9 \cdot 10^{-5}$	$5.0 \cdot 10^{-5}$
Bohm	$8.6 \cdot 10^{-6}$	$4.14 \cdot 10^{-6}$	$1.47 \cdot 10^{-5}$	$9.6 \cdot 10^{-6}$	$6.2 \cdot 10^{-5}$	$5.5 \cdot 10^{-5}$
thermal	$1.68 \cdot 10^{-5}$	$1.14 \cdot 10^{-5}$	$2.27 \cdot 10^{-5}$	$1.63 \cdot 10^{-5}$	$7.8 \cdot 10^{-5}$	$7.3 \cdot 10^{-5}$

fixed at 0. This indicates that the loss processes are calculated correctly from the diffusion coefficient. The effect of the different boundary condition is large when the diffusion time is short; this is to be expected, as the process at the wall is relatively fast, and the additional slowing it causes is likely to relatively small if the diffusion/Knudsen flow process is slow.

7.4.4 Conclusion

The fact that the diffusion coefficients calculated with the Knudsen submodel match the analytical values of these coefficients almost exactly indicates that the theory is indeed implemented correctly.

The differences in the decay time found while comparing the analytical result with the results from the model provide insight in the flaws of the analytical approach.

7.5 The model used for the hollow cathode

Now the code has been validated by this procedure, the best model of the hollow cathode will be determined. In section 7.4, a model of the hollow cathode is suggested, based on the physics of the problem. This model is oversimplified, and a number of aspects were not treated correctly. These include:

- The validation has been performed for various diffusion calculators. For the model, the one that matches the physics best has to be determined.

- The boundary condition has to be improved.

- A suitable treatment for the borehole has to be determined.

- The effect of the dropping particle densities and temperatures on the pressure has to be dealt with.

- Finally, the effect of not fixing the electron temperature must be studied.

Here, these aspects will be discussed, and based on this discussion, and the model presented in chapter 7.4.3 a model of the hollow cathode will be made.

7.5.1 The diffusion calculator

In section 7.4, the calculation has been performed for any combination of diffusion and Knudsen flow calculators available. Not all of those are physically significant. All options will be discussed here.

- Amipolar diffusion, no Knudsen flow: This is a good way to treat a dense plasma, where the mean free path is much shorter than the vessel dimensions.
- Standard diffusion, no Knudsen flow: A good treatment for normal gases if the mean free path is much shorter than the vessel dimensions.
- No diffusion, Bohm velocity Knudsen flow: In the regime where the mean free path is very long compared to the vessel dimensions, this could be a feasible treatment. However, the Knudsen flow treatment is not very accurate. A PIC model is a much better approach for situations like this. Therefore, this treatment is not recommended.
- Ambipolar diffusion, Bohm velocity Knudsen flow: This model takes into account both the diffusion process and the Knudsen flow process. It gives a treatment that is valid over a very large parameter range. It is the default treatment for a plasma. This treatment is especially good if diffusion is still the most important, but Knudsen flow cannot be neglected. In that case, the additional treatment of Knudsen flow improves the calculation, while the inaccuracies in the Knudsen flow treatment do not contribute much to the overall inaccuracy, because Knudsen flow is of relatively minor importance.
- Standard diffusion, Bohm velocity Knudsen flow: This model takes the ambipolar field into account for Knudsen flow, but not for diffusion. This is generally not an approach that makes sense. Therefore, it should not be used.
- No diffusion, thermal Knudsen flow: This model only takes Knudsen flow into account. It could be used for a nonionized gas, but there are much better approaches to that problem, such as PIC codes.
- Ambipolar diffusion, thermal velocity Knudsen flow: This model has little physical basis, as it takes the ambipolar field into account only for diffusion and not for the Knudsen flow.
- Standard diffusion, thermal velocity Knudsen flow: This is an appropriate approach for gases. It is valid over a wide parameter range.

The approach that uses ambipolar diffusion and Bohm velocity Knudsen flow is used for the rest of the models. This is because it is valid for all the parameter ranges used in the models.

7.5.2 The boundary condition

The exact mixture boundary condition also deserves attention. The easiest approach is setting the electron density at the wall to zero, and using the Knudsen module to calculate the flux to the wall. However, in practice, the density at the wall changes stepwise. Using a differential approach (equation (7.15)) in a situation where a derivative is not defined is not likely to give good results. Therefore, an approach that uses equation (7.10) is preferred.

In this approach, the particle density at the wall is determined by the free flow of particles that are close to the wall. For an isotropic velocity, this means that the flux to the wall Γ is equal to

$$\Gamma = \frac{1}{4} P \overline{v}_z N \tag{7.24}$$

However, if the chance P of a reaction is close to 1, virtually no particles will be coming back from the wall. In an environment where particle-particle collisions are rare, such as the hollow cathode, this means that the velocity distribution is not isotropic. It will be assumed that velocity distribution is similar to the isotropic case, only with v_z always positive. This changes the $f(v)$, by giving it a normalization factor that is twice as large. When this expression is integrated, the result is:

$$\Gamma_z = 2N\left(\frac{m}{2\pi k_B T_e}\right)^{\frac{3}{2}} \int_0^{2\pi} \mathrm{d}\phi \int_0^{\frac{\pi}{2}} \cos\theta \sin\theta \mathrm{d}\theta \int_0^{\infty} e^{\left(-\frac{mv^2}{2k_B T_e}\right)} v^3 \mathrm{d}v = \frac{1}{2} N v_b \tag{7.25}$$

The extra factor of two is caused by the renormalization. The thermal speed is replaced by the Bohm velocity v_b to account for the field over the sheath.

7.5.3 The borehole

The treatment of the borehole deserves special attention. On the other walls, it is assumed that the plasma diffuses with the Bohm velocity and recombines at the wall. In the borehole, this wall is absent. The obvious choice would be to implement an homogeneous Neumann boundary condition. This would effectively disable diffusion from the hollow cathode to the borehole. Because the borehole has a lower electron density, caused by faster diffusion, this is not accurate. An alternative is to assume that the density in the borehole is so low that effusion of ions and electrons from the borehole to the cathode is negligible compared to diffusion from the cathode to the borehole. This would give us that the plasma diffuses with the Bohm velocity to the wall, similar to the other walls. It will also be assumed that when an electron-ion pair diffuses out, a neutral will effuse back in.

In order to investigate which treatment is better, the decay of the borehole has been modelled to find its decay time τ_b. It has been assumed that influx and efflux of particles through the top and bottom cancel, in order to eliminate these effects. The decay time is 7.7 μs, which is much shorter than the decay time of the hollow cathode (24.5 μs) Thus, the electron density in the borehole will be much lower than the density of electrons in the cathode for the largest part of the decay. This means that the second treatment, in which the effusion of electron-ion pairs is not compensated by effusion of an electron-hole pair from the borehole back into the cathode is a better approximation.

If the first treatment is used, the decay time is 25.9 μs rather than 24.5 μs when the second treatment is used. This is not a very large error compared to the errors introduced by the unknown electron temperature, the uncertainty in the Xe mobility, etc. Therefore, the second treatment will be used, without further modification to obtain the actual answer, which lies between the two extremes.A simultaneous treatment of both borehole and cathode is beyond the abilities of our code. While the physics is in it, the geometry is too complex for our code.

7.5.4 The pressure in the hollow cathode

Another point of attention is the pressure in the hollow cathode. It is assumed to be 10 Pa. When the hollow cathode region is significantly ionized ($> 10\%$), this means that a large part of this pressure comes from electrons, as the pressure p depends on

$$p = nk_BT \tag{7.26}$$

with T the temperature of the particles. For electrons, the temperature is much (about a factor 10) higher than for the heavy particles. This means, that when the electrons get lost due to recombination, the pressure will drop. This will cause flow from the anode region through the borehole into the cathode region. In principle, one should take this flow into account. This has a number of practical problems:

- The problem cannot be approximated with the common fluid approach. This approach does not hold in a Knudsen regime, which is the regime in which the cathode is.

- The problem cannot be solved reliably and fast by our present PLASIMO code. It is thought that the weak coupling between flow equations and the other equations is responsible. This problem is similar to the problem encountered while implementing the Lorentz force (Chapter 6).

It has been attempted to simulate this situation; convergence problems made this practically impossible. Therefore, the following choice is made: it is assumed that the amount of particles in the volume is constant and has a density that sets the pressure to 10 Pa when the gas is not ionized and has a temperature of 1000 K. This means, that the pressure during the beginning of the calculation is much higher than 10 Pa. This causes the model to predict a diffusion coefficient that is too low. This effect is small when the decay is to a very low (e.g. $1 \cdot 10^{14}$ m^{-3}) electron density, because in that case the actual and simulated pressure differ only slightly for the largest part of the calculation.

7.5.5 Using a free electron density

By taking into account the heat capacity of the electrons, it is possible to let PLASIMO calculate the electron temperature during the decay. In this case, the electron temperature is not fixed at the wall (Homogeneous Neumann condition. This has been done for three starting values of the electron temperature: 8000 K, 11600 K and 15000 K. The resulting electron density as a function of time and the electron temperature as a function of time for these three settings,

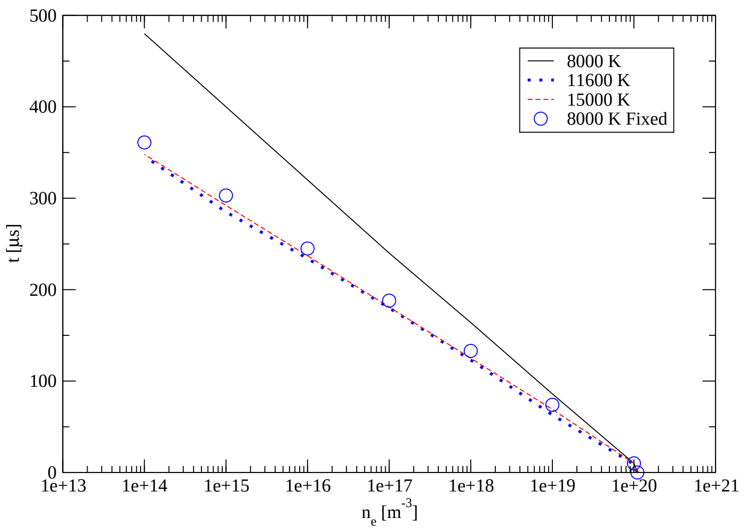

Figure 7.4: The time needed to decay to a certain electron density, when the electron temperature is free to change, for various values of the initial electron temperature. For comparison, the decay when the electron temperature is fixed at 8000 K is also given in this figure.

and a setting in which the electron temperature is fixed at 8000 K, are given in figure 7.4 and figure 7.5, respectively.

Figure 7.4 shows that the decay speed is almost identical for 15000 K and 11600 K starting temperature. In the case the starting temperature was 15000 K, it was observed that ionization caused a very rapid decay of the electron temperature. The minor increase in electron density is removed very fast by diffusion. This partially compensates the higher decay speed caused by the higher electron tempertature.

From figure 7.5, it is concluded that ionization and heat transfer rapidly let the electron temperature drop to about 8000 K. This is therefore chosen as the value for the fixed electron temperature. This does not mean that it is expected that the electron temperature actually is 8000 K at the start; this will result in a much lower average electron temperature.

The models with the free electron temperature converged poorly and slowly compared to the models with fixed electron temperature. This, combined with the fact that the difference appears to be marginal, is why for the rest of the investigation the fixed-temperature model is used.

7.6 Varying the design parameters

Having developed a model for the hollow cathode discharge, it is now possible to use this model to calculate the effect of varying several design parameters. In this section, two parameters will be varied:

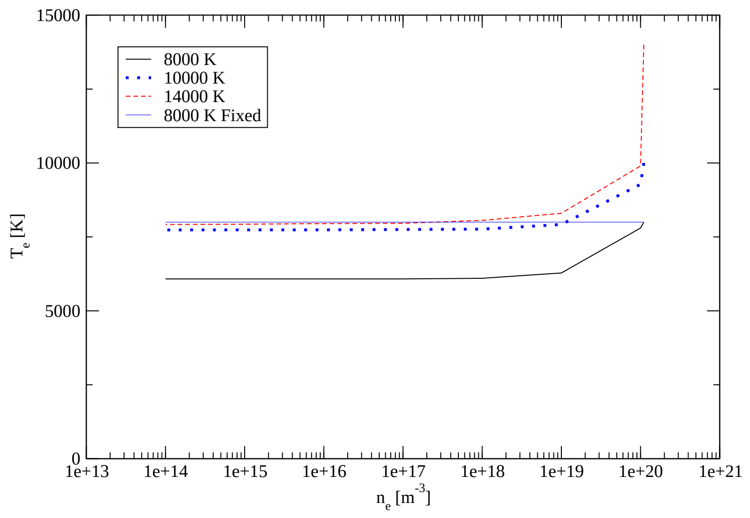

Figure 7.5: The electron temperature at various stages of the decay, when the electron temperature is free to change, for various values of the initial electron temperature. For comparison, the decay when the electron temperature is fixed at 8000 K is also given in this figure.

- The size. The device will be scaled up and down, keeping the aspect ration constant, in order to determine the effect of the size of the device on the decay time. This should show the $\tau \propto L^2$ scaling in the diffusion regime and the $\tau \propto L$ scaling in the Knudsen flow regime.

- The shape. The radius of the device will be varied in order to study the effect of this on the decay time.

7.6.1 Varying the size

In order to study the effect of the hollow cathode geometry on decay time, a series of models with various sizes has been run in PLASIMO. The decay time is very important, as it determines for a large part the repetition rate, which in turn is important for the power output.

The standard model of the hollow cathode has been modified by increasing and decreasing its size. The minimum simulated size is ten times smaller than the actual size, whereas the maximum size is ten times larger. The number of gridpoints has not not changed. For these models, τ has been determined and been set out in figure 7.6. This figure also shows the results of an analytical calculation that uses Knudsen theory and an analytical calculation that uses diffusion theory to determine τ.

The decay time closely matches the results of the analytical calculations for larger vessels. In that case, the mean free path becomes much smaller than the typical vessel dimensions. In that case, the influence of the walls becomes negligible, and the problem reduces to the standard diffusion problem. The

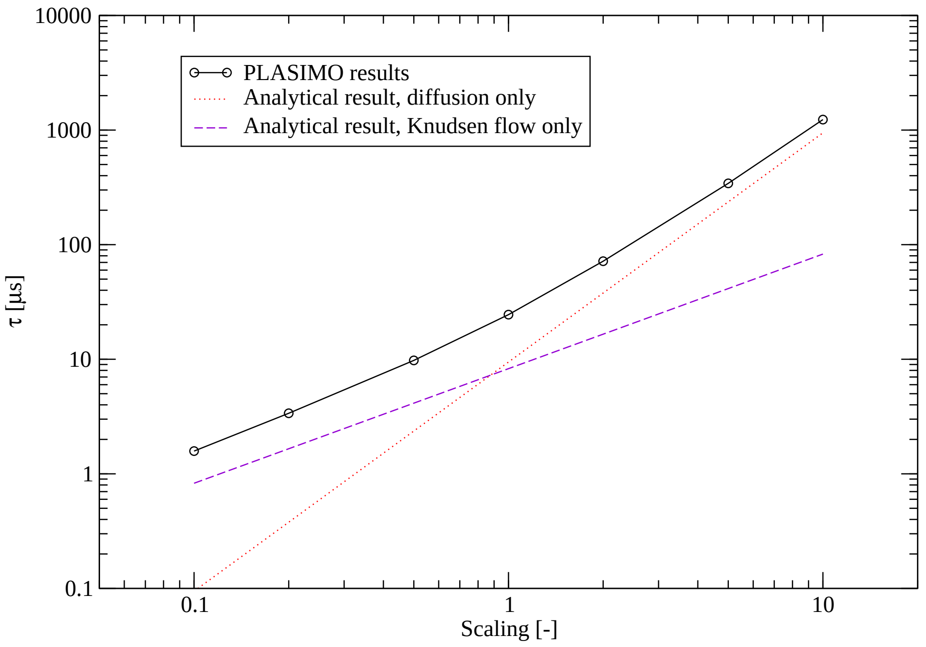

Figure 7.6: The decay time τ as a function of the size of the hollow cathode (solid line). The size is represented by a dimensionless scaling factor which is the ratio of the size of the actual hollow cathode (16 mm height and 16 mm radius.) and the hollow cathode that is being modelled. The dotted lines represents τ calculated with the analytical model. The dashed line is an estimate of τ that uses a Knudsen flow only treatment.

numerical calculation gives a diffusion speed that is slightly slower than the speed given by the analytical calculation. This is largely caused by the boundary condition. The scaling of τ with L^2, typical for diffusion, is indeed reproduced by PLASIMO in the regime in which diffusion is dominant.

For the smaller vessels where Knudsen flow becomes dominant, the analytical and actual solution differ by a factor of two. As stated, the different treatment of the boundary condition causes this difference. While the quantitative value may not be accurate, it is still much better than the value offered by the exclusive use of diffusion theory.

7.6.2 Varying the radius

Having investigated the effect of a general enlarging or reduction of the size of the hollow cathode, the effect of changing only the radius of the hollow cathode will be investigated. The height of the device is fixed at 16 mm, while models are made for various radii : 5.67 mm, 8 mm , 16 mm, 24 mm ,32 mm and 48 mm. The decay times produced are shown in figure 7.7.

Figure 7.7 shows that indeed the smallest dimension largely determines the diffusion speed: increasing the diameter of the device from 10 to 20 mm hardly

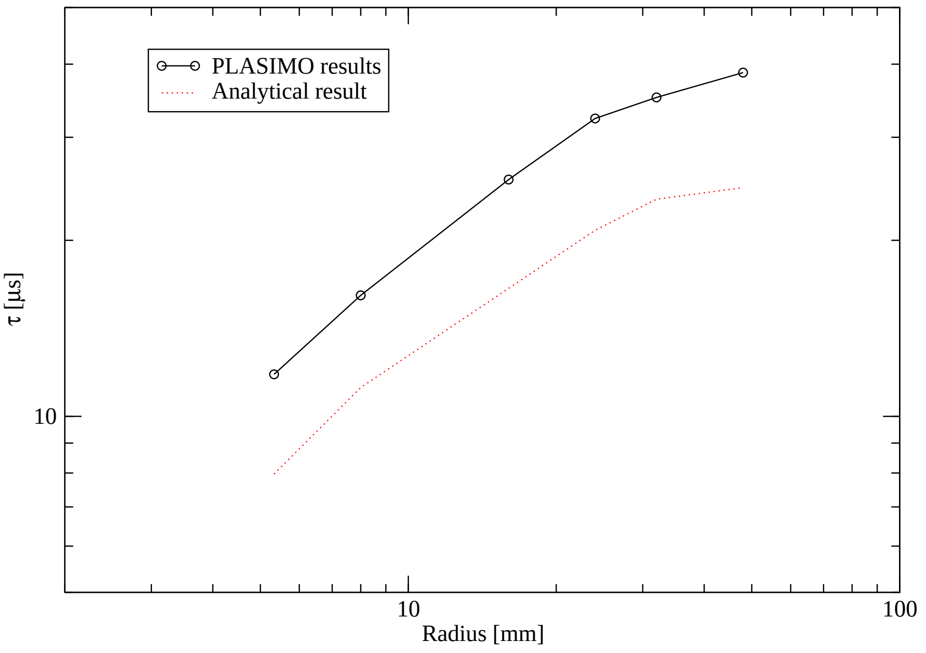

Figure 7.7: The decay time τ as a function of the radius of the hollow cathode(solid line). The height of the device is kept constant at 8 mm. The dashed line is the analytical answer.

influences the diffusion at all. The trend of the analytical result is followed well, but there is a deviation of about 30%. This deviation is caused by the different boundary conditions used for the analytical and numerical model. A reduction of r would result in a reduction of τ, but in order to gain a significant reduction, a fairly large decrease of r is needed. This introduces a variety of undesired effects. For instance, the volume would decrease significantly, causing a much greater heat load on the walls. Also, the pendulum effect, on which the working of the hollow cathode is based, may become less efficient.

7.7 Conclusions

The PLASIMO code has been extended to incorporate an approach to diffusion that combines classic diffusion and Knudsen flow in order to deal with plasmas that have a mean free path which is not much smaller than the vessel dimensions. A validation of the code against a simplified analytical model indicates that the code is indeed implemented correctly.

PLASIMO, with the new approach to diffusion, has been used to create a model that simulates the decay of the plasma in the cathode of a hollow cathode discharge. For the standard geometry, a decay time of 24.5 μs has been found.

An investigation has been conducted on the effect of the size of the hollow cathode on the decay time. A model with a variety of settings has been created in which the size of the hollow cathode was varied between one tenth of the actual size and ten times the actual size. For the larger hollow cathodes, the decay time scales with L^2, where L is a typical length scale. This behavior of τ indicates that diffusion is the dominant process in this case, because this process also scales with L^2, rather than Knudsen flow which scales with L. This is consistent with the fact that the mean free path in this case is considerably smaller than the vessel length. For the smaller hollow cathodes, τ scales with L. This behavior of τ indicates that Knudsen flow is dominant, because Knudsen flow scales with L, while diffusion does not. This behavior is consistent with the fact that Knudsen flow becomes dominant when the mean free path becomes much larger than the vessel dimensions. The normal hollow cathode appears to be between the two regimes. This justifies the use of the hybrid approach. In practice, the scaling of τ with L rather than L^2 means that it becomes more difficult to further reduce τ by reducing the dimensions of the hollow cathode.

Furthermore, the radius r is varied while keeping the height of the device constant. This indicated that the radius of the current device can be increased without seriously affecting the decay time.

Chapter 8

Conclusions

8.1 Conclusions of chapter 3

In this chapter, PLASIMO was tested by running various simple problems, for which an analytical solution exists. Comparison between the results generated with PLASIMO and the analytical solution showed no significant discrepancies.

It was also investigated whether it was possible to approach a plasma that is in LTE by using a series of NLTE models that get ever closer to LTE. While possible, it was found that the slow convergence made it difficult to use the NLTE approach when the plasma is near LTE.

8.2 Conclusions of chapter 4

In this chapter, Disturbed Bilateral Relations have been used to create a simple 0-D model that can provide estimates of various plasma parameters. The results of this model have been compared to the results of PLASIMO simulations for various parameters. For the range of parameters where the DBR model is valid, the DBR approximates the PLASIMO results well. This means, that these results can be used as starting values for a PLASIMO calculation.

8.3 Conclusions of chapter 5

Comparing the results between the NLTE calculation and the LTE calculation showed significant differences between the results of those approaches for identical situations. These can partially be explained by the difference in electron density at the wall between the LTE calculation and the NLTE calculation.

8.4 Conclusions of chapter 6

Here, a plug-in that implements the Lorentz force is described. This is a key part in the effort to model pinch plasmas. Tests revealed that while the results produced by the plug-in are accurate, the current implementation gets unstable if the pinching force becomes dominating, as it is in a real pinch plasma. It is believed that better flow solvers and a stronger coupling of the equations will alleviate this problem.

8.5 Conclusions of chapter 7

This chapter deals with the implementation of a new way of calculating the diffusion coefficient. By taking into account both normal diffusion and Knudsen flow, the diffusion coefficient can be calculated for much lower pressures than if one would use only normal diffusion. This calculator has been validated, and produced the correct results.

A model of the decay in a hollow cathode discharge has been created using the new diffusion calculator. Using this model, it was determined that the $\frac{1}{e}$ time τ is equal to 24.7 μs.

It was also investigated what the effects of reshaping the hollow cathode is. It was found that a reduction of the hollow cathode size has relatively less effect on the decay times than an increase of the hollow cathode size has. This means, that a further decrease of the size of the hollow cathode will not have as much effect as the previous reductions had.

8.6 General conclusions and recommendations

This work can be used as a basis for further study of the hollow cathode. Some aspects, such as the decay, have been studied fairly detailed, while other aspects, such as the ignition, have not been touched at all. Much of this is caused by the fact that during the work on this project, a Monte Carlo code, essential for the accurate simulation of the ignition, was not ready yet.

The obvious next step would be to use a Monte Carlo code and use it to calculate the ignition phase. Also, a comparing the results of the study on the decay with the fluid code with results obtained by using a Monte Carlo approach could be highly insightful.

In order to study the pinch phase, it is probably necessary to use a different approach, in which the equations are more strongly coupled, to obtain convergence. This is the next step in describing the pinch phase.

Appendix A

Technology assessment

Computers using silicon-based logic have been on the rise since their invention. While at first, the machines were large, cumbersome, difficult to operate, and expensive, the continuous innovations made computers more powerful, smaller, and cheaper. The impact that the ever faster computers have had on the world is difficult to overestimate.

In 1965(!), Gordon Moore predicted an exponential growth of the amount of transistors on a computer processor. [18] This exponential growth resulted initially in a doubling of the amount of transistors every year. Since the 1970, this rate has dropped to a doubling every 18 months. In order to keep the processor die (the piece of sillicon in the chip) sizes small, while increasing the amount of transistors, more transistors have to be put on a smaller area[1]. This requires ever more advanced lithographic manufacturing techniques, in order to produce smaller structures[2].

In the near future, the further miniaturization of the structures will make it impossible to use the current UV imaging techniques, as explained in chapter 2. An alternative is EUV techniques. For this, a suitable source of EUV is needed. The hollow cathode discharge is a candidate for this. Gaining a better understanding of the way this device works should make optimizing the design easier.

The speed with which the plasma in the hollow cathode decays is important for practical applications. The repetition rate of the hollow cathode is largely determined by the decay time of the plasma in the hollow cathode. By modeling this decay, it should be possible to optimize the hollow cathode with respect to the decay rate and the other design parameters.

[1]As an example, at the time of writing this (August 2002), a typical PC processor such as the AMD ”Thoroughbred” uses 54 million transistors that are put on a 88 mm^2 processor die.

[2]Smaller structures not only allow for a more complex processor design (more, or more complex, operations per clock cycle), but also for higher clock speeds and lower power dissipation.

Appendix B

The code of the DBR program

B.1 Header file

```
#include <iostream.h>
#include <iostream>
#include <fstream>
#include <string>
#include <math.h>
#include <stdlib.h>

using std::string;
using std::ifstream;//Thank You Bart H.
class DBR
{   private:
    double eaLUT[100][2],iaLUT[100][2],aaLUT[100][2];
    char filename[252];
    int sizeIA,sizeEA,sizeAA; //these are the sizes of the LUTs.
    double necenter,Te,Th,guessTe,guessTh,Twall,power,pressure;
    double ionEnergy,kB,na,r,l,eps,pi,eVtoK,kion,D,mass,mp;
    double conduction,chemical,me,kheat,HPcond,lambda,degrat,
    double np,Planck,ArrheniusPower,recrate;
    bool nameloaded;
    //calculates the diffusion coefficient D
    void calculateD(const double Te_ini, const double Ti_ini);
    //calculates Te from the mass balance.
    void calculateTe(double Te_ini);
    //calculates ne from the e power balance
    void calculatene();
    //calculate Th from HP power balance
    void calculateThHP();
    //kheat
    void calculatekheat();
    //pressure
```

```
void calculateP();
//heat conduction
void calculateHPcond(double Theavy);
//power distribution in e balance. Pretty important!!
void calculatePowerDistribution();
//Calculates Th from the h power balance with
//eps*conduction subsitituted.
//conduction is also checked to avoid losing more heat
//by conduction than is physically possible.
//this is turned off in mode 0.
void calculateThfromPower(int mode=1);
int calculateLTEne();
//obviously, this seems trivial. The tricky part is doing it nicely...
void calculatena();
//calculates the crossection from the ia LUT
double getcrosssectia(double T);
//calculates the crossection from the ea LUT
double getcrosssectea(double T);
//calculates the crossection from the aa LUT
double getcrosssectaa(double T);
int readSpecies();//reads the main species file.
int readeaLUT(); //reads the ea LUT. Function returns error value.
int readiaLUT(); //reads the ie LUT. Function returns error value.
int readaaLUT(); //reads the ia LUT. Function returns error value.
int loader();//loads the species info
//calculates the ThreeParticleRate using Saha.
void calculateThreeParticleRate(double Tel);
public:
DBR(const double Boltzmann=1.38065e-23,
    const double pii=3.1415,
    const double BoltzmannA=11604.0,
    const double mproton=1.672621e-27,
    const double melectron=9.109e-31,
    const double h=6.602e-34);
//gets wall temperature
void getTwall(double Tw=300.00){Twall=Tw;}
void getNp(double n_ini){np=n_ini;na=n_ini;} //gets average Na
//gets power, lenght,radius, and calculates power density
void getPowerDensity(const double Pow,
             const double radius,
             const double lenght);
void getguessTe(double guess=10000.0){guessTe=guess;}
void getguessTh(double guess=1000.0){guessTh=guess;}
void getSmallestDimension(double l){lambda=l;}
int getFileName(const char *name);//reads the filename.
//reads the aa LUT. Function returns error value.
int calculate(); //does the calculation. May return an error.
//Allows the extraction of the calculated ne at the center
double putnecenter(){return necenter;}
double putnacenter(){return na;} //Same for na
```

```
    double putTecenter(){return Te;} //And Te
    double putThcenter(){return Th;} //And Th
    double putPressure(){return pressure;} //And P
    //100% of the users requested this feature :O
    double putChemical(){return chemical;}
    double putConduction(){return conduction;}


};
```

B.2 Body file

```
#include "dbr.h"

int DBR::loader()//this loads all LUTs+file data.
//It does so by calling the loaders for the main file
//foo and the LUTs fooea, fooia, fooaa, and checks
//whether the loading goes OK. The error check is made
//almost redundant by the getFileName member.
{   int error,e;
    error=0;
    e=readSpecies();
    error=+e;
    e=readeaLUT();
    error=+e;
    e=readiaLUT();
    error=+e;
    e=readaaLUT();
    error=+e;
    if (error>1) error=1;
    return error;
}

double DBR::getcrosssectaa(double T)
//This function gets the value of the crosssection for a
//certain energy This energy corresponds to a temperature,
//which is used in the LUT and in the functions that call
//this function. If the temperature is lower than the lowest
//temperature value in the LUT, it returns the crosssection
//of this temperature. If it is higher than the highest
//temperature in the LUT, it returns the crosssection at the
//highest temperature in the LUT If it falls between two
//temperature values in the LUT, it makes a linear interpolation.
{   double crs;
    int i,n;
    n=0;
    i=1;
    if (T<=aaLUT[0][0])
        crs=aaLUT[0][1];
        //lower than the lowest value in the LUT
    else
    {   if (T>=aaLUT[sizeAA-1][0]) crs=aaLUT[sizeAA-1][1];
        //higher than the higarticlehest value
        elsearticle
        {   for(i=1;i<(sizeAA-1);i++)//fence post alert!!
            {   if (T>=aaLUT[i][0]) n++;
            //n is the value of the temperature in the LUT
            //just below T, n+1 lies just above.
            }
            crs=(aaLUT[n][1]-aaLUT[n+1][1])/
            (aaLUT[n][0]-aaLUT[n+1][0])*(T-aaLUT[n][0])+
            aaLUT[n][1];
            //delta sigma/ delta T *T +offset
        }

    }
    return crs;
}

double DBR::getcrosssectia(double T)
//similiar to the function above
{   double crs;
    int i,n;
    n=0;
    i=1;
    if (T<=iaLUT[0][0]) crs=iaLUT[0][1];
    //lower than the lowest value in the LUT
    else
    {   if (T>=iaLUT[sizeIA-1][0]) crs=iaLUT[sizeIA-1][1];
        //higher than the highest value
        else
        {   for(i=1;i<(sizeIA-1);i++)
            {   if (T>=iaLUT[i][0]) n++;
            //n is the value of the temperature in the LUT
            //just below T, n+1 lies just above.
            }
            crs=(iaLUT[n][1]-iaLUT[n+1][1])/
            (iaLUT[n][0]-iaLUT[n+1][0])*(T-iaLUT[n][0])+
            iaLUT[n][1];//interpolation.
        }

    }
    return crs;
}

double DBR::getcrosssectea(double T)
//similiar to the function above
{   double crs;
    int i,n;
    n=0;
    i=1;
    if (T<=eaLUT[0][0]) crs=eaLUT[0][1];
    //lower than the lowest value in the LUT
    else
    {   if (T>=eaLUT[sizeEA-1][0]) crs=eaLUT[sizeEA-1][1];
        //higher than the highest value
        else
        {   for(i=1;i<(sizeEA-1);i++)
            {   if (T>=eaLUT[i][0]) n++;
            //n is the value of the temperature in the LUT
            //just below T, n+1 lies just above.
            }
            crs=(eaLUT[n][1]-eaLUT[n+1][1])/
            (eaLUT[n][0]-eaLUT[n+1][0])*(T-eaLUT[n][0])+
            eaLUT[n][1]; //interpolation.
        }

    }
    return crs;
}

int DBR::getFileName(const char *name)
//This function checks the existance of the files *name, *nameaa,
//*nameea and *nameia. It also puts the contents of *name in the
//filename variable of the DBR class. This funcion is mainly
//a big errorcheck.
{   int error;
    char otherName[254];
    error=0;
    //We start without errors. Let's hope it stays that way...
    strcpy(filename,name);
    //now, the filename is loaded into the class.
    ifstream f(filename,ios::nocreate);
    error+=(!f);
    strcpy(otherName,filename);
    strcat(otherName,"ea");
    ifstream eal(otherName,ios::nocreate);
    error+=(!eal);
    strcpy(otherName,filename);
    strcat(otherName,"aa");
    ifstream aal(otherName,ios::nocreate);
    error+=(!aal);
    strcpy(otherName,filename);
    strcat(otherName,"ia");
    ifstream ial(otherName,ios::nocreate);
    error+=(!ial);
    return (error);
}

int DBR::readSpecies()
```

```
//This function reads the main species data file. This file should
//contain the following:
//The mass of the particle in a.u.
//The ionization reaction rate constant
//The power in which the temperature occurs in the reaction rate
//The energy level of the ion groundstate
//The ratio of the degeneracy of the ion groundstate and the
//groundstate
{   int error=0;
    ifstream mainfile(filename,ios::nocreate);
    if (!mainfile) error++; //file not found error.
    else //we read the file.
    {   string buffer;
        mainfile >> buffer;
        mass = (atof(buffer.c_str()))*mp;
        mainfile >> buffer;
        kion = atof(buffer.c_str());
        mainfile >> buffer;
        ArrheniusPower = atof(buffer.c_str());
        mainfile >> buffer;
        ionEnergy = atof(buffer.c_str());
        mainfile >> buffer;
        degrat = atof(buffer.c_str());
    }
    return error;
}

int DBR::readeaLUT()
//Format: The first line is the number of crosssections.
//Then, it is a table, with left the temperature and right
//the crosssection.
{   int error=0;
    int i,max;
    char d[254];
    strcpy(d,filename);
    strcat(d,"ea");
    ifstream eal(d,ios::nocreate);
    if (!d) error++; //file not found.
    else
    {   string buffer;
        eal >> buffer;
        max = atoi(buffer.c_str());
        //It is hoped the correct information is specified
        //here. An errorcheck will be difficult to implement.
        sizeEA=max; //sets the size of the look-up table.
        //Each table has its size.
        for (i=0;i<max;i++)//reading the file
        {   eal >> buffer;
            eaLUT[i][0]=atof(buffer.c_str());
            eal >> buffer;
            eaLUT[i][1]=atof(buffer.c_str());
        }
    }
    return error;
}

int DBR::readaaLUT()
//format: The first line is the number of crosssections.
//Then, it is a table. It is similiar to the function above.
{   int error=0;
    int i,max;
    char d[254];
    strcpy(d,filename);
    strcat(d,"aa");
    ifstream aal(d,ios::nocreate);
    if (!d) error++; //file not found.
    else
    {   string buffer;
        aal >> buffer;
        max = atoi(buffer.c_str());
        //it is hoped the correct information is specified
        //here. an errorcheck will be difficult to implement.
        sizeAA=max; //sets the size of the look-up table.
        //Each table has its size
        for (i=0;i<max;i++)
        {   aal >> buffer;
            aaLUT[i][0]=atof(buffer.c_str());
            aal >> buffer;
            aaLUT[i][1]=atof(buffer.c_str());
        }
    }
    return error;
}

int DBR::readiaLUT()
//format: The first line is the number of crosssections.
//Then, it is a table. It is similiar to the function above.
{   int error=0;
    int i,max;
    char d[254];
    strcpy(d,filename);
    strcat(d,"ia");
    ifstream ial(d,ios::nocreate);
    if (!d) error++; //file not found.
    else
    {   string buffer;
        ial >> buffer;
        max = atoi(buffer.c_str());
        //it is hoped the correct information is specified
        //here. an errorcheck will be difficult to implement.
        sizeIA=max; //sets the size of the look-up table.
        //Each table has its size
        for (i=0;i<max;i++)
        {   ial >> buffer;
            iaLUT[i][0]=atof(buffer.c_str());
            ial >> buffer;
            iaLUT[i][1]=atof(buffer.c_str());
        }
    }
    return error;
}

void DBR::calculateD(const double Te_ini, const double Ti_ini)
//This member calculates the reduced ambipolar diffusion coefficient.
{   double tau,Di,crs;
    crs=getcrosssectia(Ti_ini);
    tau=1.0/crs/sqrt(8.0*kB*Ti_ini/pi/mass); //tau=1/sigma/V
    Di=tau * kB * Ti_ini / mass; //tau=sigma*V
    D=Di*(1.0+(Te_ini/Ti_ini));
    //verified. Seems to work just fine
}

void DBR::calculateThreeParticleRate(double Tel)
//This calculates the three-particle recombination rate.
{   double z,zz,K;
    z=(Planck/sqrt(2*pi*me*kB*Tel));
    zz=pow(z,3);
    K=kion*pow(Tel,ArrheniusPower);
    recrate=K*zz/degrat/2;
}

void DBR::calculateTe(double Te_ini)
//calculates Te from the particle balance. This is rather
//straightforward and is likely to give rather good results.
//In fact, practical testing has indicated that the prediction
//of Te is more accurate than the prediction of ne and Th.
//the arrhenius reaction rate is hardwired into this member.
{   double K,BoltzE;
    K=D/np/na/lambda/lambda+recrate*necenter*necenter/np;
    BoltzE=K/kion/pow(Te_ini,ArrheniusPower);
    Te=-ionEnergy*eVtoK/log(BoltzE);
}

void DBR::calculatene()//This calculates ne from the energy
//balance. Both processes are linear in ne
//so this is not too hard, fortunately.
{   double K;
    K=exp(-ionEnergy*eVtoK/Te);
    necenter=eps/(D/lambda/lambda*ionEnergy/
    np*kB*eVtoK+kheat*(Te-Th)*np);
    //this works if there is recombination.
    //It is incorrect to use
}
//with a good way of describing ne and Te, we at least
//have a chance of getting a good description of Th.

void DBR::calculatekheat()
//The amount of energy lost by these processes is given by
//the amount of heat lost in one process time the collision
//frequency. The amount of heat lost in one collision is given
//by 2me/mh*3/2*KB(Te-Th). (Te-Th) gets separated. The collision
//time is tau=C/Te*(ne*na) Again, the last part gets seperated
{   double K,tau,crs;
    crs=getcrosssectea(Te);
    K=me/mass*3.0*kB;
    tau=1.0/crs/sqrt(8.0*kB*Te/pi/me);
    kheat=K/tau;
}

void DBR::calculateP()
{   pressure=np*kB*Th+necenter*kB*Te;
}
//can't do much wrong with that, altough things might get
//interesting if we try to use profiles. Note the temperature
//profile. Not implemented yet though. You have to make an
//educated guess yourself :(. Implementation will be very difficult,
//because we would have to know the profile. This is somewhat
//antithetical to the 1-control volume approach to say the least.


DBR::DBR(const double Boltzmann,const double pii,
    const double BoltzmannA, const double mproton,
    const double melectron, const double h)
{   kB=Boltzmann;
    pi=pii;
    eVtoK=BoltzmannA;
    mp=mproton;
    me=melectron;
    Planck=h;
    //This implements the fundamental constants of nature.
    //This way, we may overload them with our own values.
    //This has no practical use whatsoever
    conduction=0.00000005;
    chemical=0.99999995;
    nameloaded=false;
    //an initial guess. Too much conduction has a tendency of
    //screwing the calculation, so we start at the safe side.
}
void DBR::getPowerDensity(const double Pow,const double radius,
                const double lenght)
```

```
{    int error=0;
     power=Pow;
     r=radius;
     l=lenght;
     double vol=pi*r*r*l;
     if (vol >0.0000001) eps=power/vol;
     else
     {    eps=1;
          error++;
     }
     //The whole programs uses power densities rather than powers.
}

void DBR::calculateHPcond(double Theavy)
{    double crs,ls,vtherm;
     crs=getcrosssectaa(Theavy);
     vtherm=sqrt(8.0*kB*Theavy/pi/mass);
     ls=0.70716/(np*crs);
     HPcond=(15.0/8.0)*kB*np*ls*vtherm;
}
//This represents the heat conduction of a gas. Of course, we may
//need to use electrons too. This will be very inaccurate, difficult
//and empirical. I'd rather stay clear of it.

void DBR::calculateThHP()
{    double ca,cb;
     ca=necenter*np*kheat;
     cb=HPcond/lambda/lambda;
     Th=(ca*Te+cb*Twall)/(ca+cb);
}
//Th calculated from the heavy particle balance, which seems like
//a sensible place to get it from. The size of both parts of the
//equation is given by the electron power distribution

void DBR::calculateThfromPower(int mode)//mode; 1=errorcheck,0 no check.
{    int i;
     i=0;
     Th=Twall+lambda*lambda*eps*conduction/HPcond;
     while (mode &&(Th>Te)&&(i<100000)) //obviously, we have
     //channeled too much energy in conduction...
     {    conduction=conduction*0.999;
          chemical=1-conduction;
          Th=Twall+lambda*lambda*eps*conduction/HPcond;
          i++;
     }
}

void DBR::calculatePowerDistribution()
//calculates the distribution of the imput power over diffusion losses
//and consuction losses
```

```
{    double K;
     K=exp(-ionEnergy*eVtoK/Te);
     chemical=necenter/np*D/lambda/lambda/eps*ionEnergy*kB*eVtoK;
     conduction=1-chemical;

}

void DBR::calculatena() //This is going to be hard, unless...
//In a NLTE plasma, ionization is not likely to rise above 99.9%, right?
//So let's put a ceiling there.
{    const double max=0.0001;
     na=np-necenter;
     if ((na/np)<max)
     {    na=np*max;
          necenter=np-na;
     }
}

int DBR::calculate()
{    const int imax=100;
     const int imin=1;
     int i,e,error;
     //0=Normal nLTE calculation
     //1=error
     double c;
     necenter=sqrt(np);
     if (necenter>np) necenter=np/1.1; //errorcheck.
     error=loader();//check for errors in the loader. The filename
     //check should catch most unless a LUT is missing.
     if (error) return (1);//if we have an error, we quit.
     else
     {    na=np;
          calculateD(guessTe,guessTh);
          calculateThreeParticleRate(guessTe);
          calculateTe(guessTe);
          for (i=imin; i<imax; i++)
          {    calculateD(Te,guessTh);
               calculateThreeParticleRate(Te);
               calculateTe(Te);
          }//iteratively determining Te
          calculatekheat();
          calculateHPcond(guessTh);
          calculateThfromPower(1);
          calculatene(); //At this point,we know a lot.
          //good guess for a second pass!
          calculatena();
          for (i=imin; i<imax; i++)
          {    calculatene();
               calculateTe(Te);
               calculateThfromPower(1);
```

```
            //actually, this is used to determine whether
            //the amount of conduction makes any sense.
            calculateThHP();
            calculatePowerDistribution();
            calculateHPcond(Th);
            calculateD(Te,Th);
            calculatekheat();
            calculatena();
            calculateThreeParticleRate(Te);
        }
        calculateP();
    }
    return error;//either 0 or 1, depending on no error/error
}

double getdouble()// a robust way of reading a character.
{   char input[1000];
    double result;
    int error;
    char *errpointer;
    error=1;
    while (error)
    {   cin>>input;
        result=strtod(input,&errpointer);
        if (*errpointer=='\0') error=0; //Thank you Bart H.
        else cout<<"Error␣in␣input␣string"<<endl;
    }
    return result;
}

int main()
{   DBR b;
    double x,xx,xxx;
    int a;
    char filename[254];
    a=1;
    while (a)
    {   cout<< "What␣particle␣are␣we␣using?"<<endl;
        //gets the filename.
        cin >> filename;
        a=b.getFileName(filename);
        if (a) cout<<"File␣not␣found"<<endl;
    }
    a=1;
    while (a)
    {   cout << "Enter␣the␣wall␣temperature␣in␣K"<<endl;
        x=getdouble();
        if (x>0)
        {   b.getTwall(x);
            a=0;
        }
        else cout<<"Temperatures tend to be positive"<<endl;
    }
    a=1;
    while (a)
    {   cout << "Enter the average heavy particle density in m^-3" <<endl;
        x=getdouble();
        if (x>0)
        {   b.getNp(x);
            a=0;
        }
        else cout<<"Particle densities tend to be positive"<<endl;
    }
    a=1;
    while (a)
    {   cout <<"Enter the total power in W" <<endl;
        x=getdouble();
        if (x>0) a=0;
        else cout<<"We need a positive amount of power"<<endl;
    }
    a=1;
    while (a)
    {   cout <<"Enter the radius in m"<<endl;
        xx=getdouble();
        if (xx>0) a=0;
        else cout<<"We need a positive radius"<<endl;
    }
    a=1;
    while (a)
    {   cout <<"Enter the lenght in m"<<endl;
        xxx=getdouble();
        if (xxx>0) a=0;
        else cout<<"We need a positive lenght"<<endl;
    }
    a=1;
    b.getPowerDensity(x,xx,xxx);
    while (a)
    {   cout <<"Enter the diffusion lenght in m"<<endl;
        x=getdouble();
        if (x>0)
        {   a=0;
            b.getSmallestDimension(x);
        }
        else cout<<"We need a positive diffusion lenght"<<endl;
    }
    x=10000;
    b.getguessTe(x);
    x=1000;
    b.getguessTh(x);
    //I have yet to ancounter a case in which these guesses do not work.
    a=b.calculate();
    if (a)  cout<<"an␣error␣has␣occured␣during␣the␣calcultion";
    else
    {    x=b.putnecenter();
         cout << "An␣estimate␣for␣ne␣at␣the␣center␣is␣"
               << x << "m^-3" << endl;
         x=b.putnacenter();
         cout << "An␣estimate␣for␣na␣at␣the␣center␣is␣"
               << x << "m^-3" << endl;
         x=b.putTecenter();
         cout << "An␣estimate␣for␣Te␣at␣the␣center␣is␣"
               << x << "K" << endl;
         x=b.putThcenter();
         cout << "An␣estimate␣for␣Th␣at␣the␣center␣is␣"
               << x << "K" << endl;
         x=b.putPressure();
         cout << "An␣estimate␣for␣p␣at␣the␣center␣is␣"
               << x << "Pa" <<endl;
         x=b.putChemical();
         xx=b.putConduction();
         cout<<"The␣portion␣of␣the␣energy␣used␣for␣heat␣conduction␣is␣"
              << xx << endl;
         cout<<"The␣portion␣of␣the␣energy␣used␣for␣particle␣generation␣is␣"
              << x << endl;
    }
    return(0);
}
```

Appendix C

The code of the pinching module

C.1 Header file

```
#ifndef H_EM_UNIFORM1D_H
#define H_EM_UNIFORM1D_H

#include "plem/em.h"
#include "plem/base_em.h"

class plBaseEMUniform1dData : public plBaseEMData
{
protected:
    void CalculateLorentzForce(void);
    plBaseEMUniform1dData( plModelRegion * reg, plNode & emnode );
    bool m_calc_lorentz;
    REAL m_lorentz_urf;
    CGridVar<REAL> m_B3;
};


#endif
```

C.2 Body file

```
#include "plconfig.h"
#include "plem/em_uniform1d.h"
#include "plconfig.h"
#include "plem/base_em.h"
#include "plgeneric/forcelnk.h"
#include "plmath/consts.h"
#include "plgrid/grid.h"
#include "plgrid/gridvar.h"

/*
```

```
 * Implementation of the Lorentz force calculation.
 * - Logic by Bart Broks
 * - Beautification by Bart Hartgers
 */

plBaseEMUniform1dData::plBaseEMUniform1dData(
    plModelRegion *reg, const plNode & emNode )
    : plBaseEMData(reg,emNode),
    m_B3(reg,"B3",plGrid::loc_nodal)
{
    m_B3.FName()="B3";
    plNode::const_iterator i=emNode.onlyN("LorentzForce");
    if (i) {
        if (reg->Grid()->coord_type != cylindrical)
            throw plParserException("LorentzForce␣only␣supported␣"
                        "on␣cylindrical␣grids");
                        ///Works only on cylindrical grids
                        ///Take care if you implement a new grid type
                        ///that should work with a Lorentz force!!
        m_calc_lorentz = true;
        m_lorentz_urf = (**i)("URF");
    } else {
        m_calc_lorentz = false;
        m_lorentz_urf = 0;
    }
}

void plBaseEMUniform1dData::CalculateLorentzForce(void)
{
    /// This piece of code calculates the Lorentz force.
    /// The Lorentz force is a force that a magnetic field exerts on
    /// a moving charged particle.
    /// In this calculation, the following is assumed:
    ///
    /// - The magnetic field is caused by the current itself
    ///   (pinching, no external magnetic fields)
    /// - The current loop is closed at infinity
    /// - There is a cylindrical symmetry
    /// - The Lorentz force in the x1-direction is negligible
    /// - The Lorentz force is implemented as a bulk force,
    ///   meaning that the charged and neutral particles travel
    ///   together.
    ///
    /// The magnetic field is caused by the current that is surrounded
    /// by the control volume.
    /// èAmpre's law is used to calculate its magnitude. Now, this
    /// magnetic field interacts with the current in the control volume,
    /// resulting in a Lorentz force. The force density is
    /// calculated and returned.
    unsigned i, j;
```

```
  unsigned int N1 = sig.Grid()->n1();
  unsigned int N2 = sig.Grid()->n2();
 if (m_calc_lorentz==true)
 {
  REAL sumI, I, r, F, l, J, pr, pl, area, radius, length, PartCovered;


  for (i=0; i<N1; i++)
  {
      sumI = 0;
      // this gives the total length and radius of the plasma. Seems
      // we have to do this the hard way.
      for (j=0; j<(N2-1); j++)
      {   //This part will get the lenghts and grid elements and the
          //position of the nodal points.
          length = sig.Grid()->L1(Pos(i,j));
          radius = sig.Grid()->L2(Pos(i,j));
          r = sig.Grid()->dx2(Pos(i,j)) * radius;
          l = sig.Grid()->dx1(Pos(i,j)) * length;
          sig.Grid()->get_phys_location(plGrid::loc_nodal,i,j,pl,pr);

          // Let's calculate I.
          J = m_Eimposed1(i,j)*sig(i,j);

          // now, we need to know the volume...
          if ((j==0) || (j==N2-1))
              area = 0;
          else if (i==0)
              area = sig.Grid()->areaCV(East,i,j);
          else if (i==N1-1)
              area = sig.Grid()->areaCV(West,i,j);
          else
              area = (sig.Grid()->areaCV(West,i,j) +
                  sig.Grid()->areaCV(East,i,j))/2;

          I = J*area;

          // This is exact... Area calculation has been checked and looks fine.
          if (!(area==0)) PartCovered=(Constant::pi)*((pr*r)-(r*r/4))/area;
          else PartCovered=1;
          if (PartCovered<0.1) PartCovered=0.1;
          if (PartCovered>0.9) PartCovered=0.9;
          sumI += (I*PartCovered);

          //This calculates the part of the current
          //that flows trought the control surface
          //It is exact if the nodal point is in the
          //middle of the CV; it is highly accurate
          //if it lies between 0.35 and 0.7 of the lenght of the CV.
          // This is Ampere's Law: 2*pi*r*B=mu0*Iencl
```

```
                if (!j)
                    m_B3(i,j) = 0;
                else
                    m_B3(i,j) = sumI*(Constant::mu0)/2/(Constant::pi)/pr;
                // This should provide us with a nice magnetic field!
                // It is in the phi-direction!!
                // Now that we have a B-field,
                // we can use it to calculate
                // the Lorentz force.
                F = - m_B3(i,j)*J;
                sumI += (I*(1-PartCovered));
                // Note that the l's disappear
                // Sign? This is inside, I suppose.
                m_lor1(i,j) = 0;
                m_lor2(i,j) = (m_lor2(i,j)*(1.0-m_lorentz_urf) +
                           m_lorentz_urf*F);

            }
            //This part gives a nice value of the Lorentz
            //force at the boundary.
            m_B3(i,(N2-1))=m_B3(i,(N2-2));
            m_lor1(i,(N2-1)) = 0;
            m_lor2(i,(N2-1)) = m_lor2(i,(N2-2));
            //12-08-02 BB
            // Hack: a nonhomogeneous Lorentz force
            // in the center is NOT going to work.
            // All terms in your momentum balance
            // are homogeneous in the center.
            // So there is "no way" this will ever
            // satisfy the momentum balance.
            // So, we give the Lorentz force a
            // homogeneous Dirichlet condition
            // because we don't allow for an
            // inward velocity in the center
            // and the Ampere's law surface
            // has surface 0. AND an homogeneous
            // Neumann, for reasons outlined above.
            m_lor2(i,1)=0;
        }
    }
    else
    {       for (i=0; i<N1; i++)
        {   for (j=0; j<N2; j++)
            {   m_B3(i,j)=0.0;
                m_lor1(i,j) = 0.0;
                m_lor2(i,j) = 0.0;
            }
        }
    }
}
```

Appendix D

The analytical model of the decay in hollow cathode discharge

In this appendix, the diffusion time constant of a plasma in a cylindrical geometry will be determined. This is done by solving equation D.1:

$$\frac{\partial n_e}{\partial t} = (D_f \nabla^2 n_e) \tag{D.1}$$

For the solution, the existence of nontrivial solutions of $n_e(t, r, z)$ in the form

$$n_e(t, r, z) = T(t)R(r)Z(z) \tag{D.2}$$

with $T(t)$ a function that depends only on t, $R(r)$ a function that depends only on r, and $Z(z)$ function that depends only on z, is assumed. This yields a set of solutions, which can be used to reconstruct the actual solution, that is also determined by the boundary conditions:

$$\frac{\partial n_e(0, z, t)}{\partial r} = n_e(b_r, z, t) = n_e(r, 0, t) = n_e(r, b_z, t) = 0 \tag{D.3}$$

Substituting equation (D.2) in equation (D.1) yields gives:

$$\frac{\partial T}{\partial t} RZ = -D_f \left(T \frac{1}{r} \frac{\partial}{\partial r} \left(r \frac{\partial R}{\partial r} \right) Z + TR \frac{\partial^2 Z}{\partial z^2} \right) \tag{D.4}$$

This can be rewritten to

$$\frac{\partial T}{\partial t} \frac{1}{T} = -D_f \left(\frac{1}{r} \frac{\partial}{\partial r} \left(r \frac{\partial R}{\partial r} \right) \frac{1}{R} + \frac{\partial^2 Z}{\partial z^2} \frac{1}{Z} \right) \tag{D.5}$$

This equation must hold for all values of t,r and z. Because there are three parts in the equation, each only depending on t, r and z, all of these parts have to be constant for the equation to hold for all t,r and z. This can be made explicit by separating the equation in:

$$\frac{\partial T}{\partial t} \frac{1}{T} = -D_f s \tag{D.6}$$

$$\frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial R}{\partial r}\right)\frac{1}{R} = -s_r \tag{D.7}$$

$$\frac{\partial^2 Z}{\partial z^2}\frac{1}{Z} = -s_z \tag{D.8}$$

$$s = s_r + s_z \tag{D.9}$$

First, equation (D.7) will be solved. This equation can be rewritten to:

$$r^2\frac{\partial^2 R}{\partial r^2} + r\frac{\partial R}{\partial r} + s_r r^2 R = 0 \tag{D.10}$$

It can be shown that for these boundary conditions, s_r can be written as $s_r{=}\alpha_r$, with α_r a positive number. This will be used to scale r to obtain a dimensionless equation:

$$\rho = \frac{\alpha_r r}{b_r} \tag{D.11}$$

Expressing equation (D.10) in terms of ρ:

$$\rho^2\frac{\partial^2 R}{\partial \rho^2} + \rho\frac{\partial R}{\partial \rho} + \rho^2 R = 0 \tag{D.12}$$

This is a special form of the Bessel equation

$$x^2\frac{\partial^2 y}{\partial x^2} + \rho\frac{\partial y}{\partial x} + (x^2 - n^2)y = 0 \tag{D.13}$$

with y a function of x. The solution of this equation is

$$y = A_r J_n(x) + B_r Y_n(x) \tag{D.14}$$

with A_r and B_r constants, J is the Bessel function and Y is the Weber function. This means that the solution of equation (D.12) can be expressed as

$$R = AJ_0(\alpha_r x) + BY_0(\alpha_r x) \tag{D.15}$$

Because our density must be finite, and Y_0 is infinite in 0, B_r must be equal to 0. The value of $\alpha_r x$ is determined by the boundary condition at $r = b_r$. The Bessel function $J(x)$ has its first zero at $x \approx 2.40$ [19], thus, α_r is approximatly equal to $2.40 b_r^{-1}$. A_r is not determined; in practice, it is determined by the starting value of the problem. Furthermore, the Bessel function has an infinite amount of zeros, thus, there are other solutions, for which α_r is larger than $2.40 b_r^{-1}$. These other solutions will be discussed later.

Next, the equation for Z will be solved. This is similiar but somewhat easier than solving the equation for R. Rewriting equation (D.8) gives:

$$\frac{\partial^2 Z}{\partial z^2} + \alpha_z^2 Z = 0 \tag{D.16}$$

where $\alpha_z^2 = s_z$. This is a standard differential equation, which has the following answer:

$$Z = A_z \sin(\alpha_z z) + B_z \cos(\alpha_z z) \tag{D.17}$$

Here, A_z and B_z are arbitrary constants. The boundary condition at $z = 0$ gives that B_z equals 0. The boundary condition at $z = b_z$ gives $\alpha_z = \pi b_z^{-1}$. There are also solutions for larger values of α_z; however these are not very relevant, as will be shown later.

By solving equation (D.7) and (D.8), we can now obtain a value for s. It is equal to

$$s = \left(\frac{2.40}{b_r}\right)^2 + \left(\frac{\pi}{b_z}\right)^2 \tag{D.18}$$

This value can be substituted in equation D.6 to give:

$$\frac{\partial T}{\partial t} = -D_f\left(\left(\frac{2.40}{b_r}\right)^2 + \left(\frac{\pi}{b_z}\right)^2\right)T \tag{D.19}$$

This equation is easily solved and has the solution

$$T = A_t e^{\frac{t}{\tau}} \tag{D.20}$$

with A_t an arbitrary constant and the decay time constant τ equal to

$$\tau = \frac{1}{-D_f\left(\left(\frac{2.40}{b_r}\right)^2 + \left(\frac{\pi}{b_z}\right)^2\right)} \tag{D.21}$$

Earlier, it was mentioned that the higher values of α_r and α_z were less important. Equation (7.23) shows that these modes decay much faster. Therefore, they will rapidly disappear, leaving only the lowest-order mode:

$$T = Ae^{\left(\left(\frac{2.40}{b_r}\right)^2+\left(\frac{\pi}{b_z}\right)^2\right)} J_0\left(\frac{2.40r}{b_r}\right)\sin\left(\frac{\pi z}{b_z}\right) \tag{D.22}$$

Appendix E

The code of the `plKnudsenDiffusion` class

```
class plKnudsenDiffusion : public plAmbipolarDiffusion
{
    public:
    plKnudsenDiffusion(const plNode& node,
                const plParticleMap& pmap,
                const CBaseRelationMap& rmap)
                :plAmbipolarDiffusion(node,pmap,rmap)
                , m_ParticleMasses (pmap.size() )
                , m_DDiffusion( pmap.size() )
                , m_DKnudsen (pmap.size() )

    {
    //parses geometry factor and diffusion lenght.
    m_K=REAL(node["Calculator"]("KnudsenGeometryFactor"))
        *REAL(node["Calculator"]("DiffusionLength"));
    for (unsigned ndx = 0; ndx < pmap.size(); ndx++)
    {
        m_ParticleMasses[ndx]=pmap[ndx]->Mass();
    }

    //The following section, which undoubtly could have been
    //written more compactly, parses the type of diffusion
    //and Knudsen flow we are using.
    string Sd = node["Calculator"]("DiffusionMode");
    string S1 ="Disable";
    string S2 ="Ambipolar";
    string S3 ="Standard";
    if (Sd==S1) m_DiffusionMode=0;
    else if (Sd==S2) m_DiffusionMode=1;
    else if (Sd==S3) m_DiffusionMode=2;
    else throw plParserException("Illegal␣diffusion␣type␣specified");
    string S4 ="Bohm";
    string S5 ="Thermal";
    string Sk=node["Calculator"]("KnudsenMode");
    if (Sk==S1) m_KnudsenMode=0;
    else if (Sk==S4) m_KnudsenMode=1;
    else if (Sk==S5) m_KnudsenMode=2;
    else throw plParserException("Illegal␣Knudsen␣velocitytype␣specified")
    if (!(m_DiffusionMode || m_KnudsenMode))
        throw plParserException("No␣diffusioncoefficient␣defined");
    //No diffusion at all will mean doom for the calcualtion.
    }


    virtual void DoCalculate( const plParticleValue<REAL> & res,
                const plCrossSectionMatrix & cs,
                const plConstValueRef<REAL> & point );

    private:
    std::vector<REAL> m_ParticleMasses;
    //The masses of the various species.
    std::vector<REAL> m_DDiffusion; // The diffusion coefficient
    std::vector<REAL> m_DKnudsen; // The effusion coefficient
    REAL m_K; //The Knudsen parameter.
    unsigned m_DiffusionMode,m_KnudsenMode;
    //Diffusion modes:
    //0: Disabled. Knudsen only
    //1: Ambipolar (default)
    //2: Standard
    //Knudsen modes:
    //0: Disabled. Diffusion only
    //1: With Bohm velocity (default)
    //2: With thermal velocity
    void CalculateDDiffusion(const plParticleValue<REAL> & res,
                const plCrossSectionMatrix & cs,
                const plConstValueRef<REAL> & point );
                //Calulates the diffusion coefficient.
    void CalculateDKnudsen (const plParticleValue<REAL> & res,
               const plConstValueRef<REAL> & point);
               //Calculates the Knudsen diffusion coefficient.
};

void plKnudsenDiffusion::DoCalculate( const plParticleValue<REAL> & res,
                    const plCrossSectionMatrix & cs,
                    const plConstValueRef<REAL> & point )

{
    if (m_DiffusionMode && m_KnudsenMode) //Both Knudsen and Diffusion
    {
        CalculateDDiffusion(res,cs,point);
        CalculateDKnudsen(res,point);
        for (unsigned ndx = 0; ndx < res.size(); ndx++)
        {
            res[ndx] =(m_DDiffusion[ndx]*m_DKnudsen[ndx])/
                  (m_DDiffusion[ndx]+m_DKnudsen[ndx]);
            //reciprocal addition. If the denominator would be 0,
            //so would the temperature.
            //The calculation is messed up anyway,
            //in that case, so no error checking.
        }
    }
    else if (!m_DiffusionMode) //Only Knudsen
    {
        CalculateDKnudsen(res,point);
        for (unsigned ndx = 0; ndx < res.size(); ndx++)
        {
            res[ndx] = m_DKnudsen[ndx];
        }
    }
    else  //Only diffusion
    {
        CalculateDDiffusion(res,cs,point);
        for (unsigned ndx = 0; ndx < res.size(); ndx++)
        {
            res[ndx] =m_DDiffusion[ndx];
        }
    }

}

void plKnudsenDiffusion::CalculateDDiffusion(
             const plParticleValue<REAL> & res,
             const plCrossSectionMatrix & cs,
             const plConstValueRef<REAL> & point )
{
    if (m_DiffusionMode==1) //ambipolar diffusion
    {   plAmbipolarDiffusion::DoCalculate(res,cs,point);
        for (unsigned ndx = 0; ndx < res.size(); ndx++)
        {
            m_DDiffusion[ndx] = res[ndx];
        }
    }
    else // standard diffusion.
    {   plDiffusion::DoCalculate(res,cs,point);
        for (unsigned ndx = 0; ndx < res.size(); ndx++)
        {
            m_DDiffusion[ndx] = res[ndx];
        }
    }
}

void plKnudsenDiffusion::CalculateDKnudsen(
            const plParticleValue<REAL> & res,
            const plConstValueRef<REAL> & point)
{
    REAL vth,M,T;
    unsigned elNdx = Particles().GetElectronNdx();
    for (unsigned ndx = 0; ndx < res.size(); ndx++)
    {
        T=plAccessor::Temperature(point)[ndx]; //Thermal velocity
        if (m_KnudsenMode==1) //Bohm velocity
        {
            if ( Particles()[ndx]->ChargeNr() > 0)
                T=plAccessor::Temperature(point)[elNdx];
                //only apply for ions.
                //electron diffusion isn't treated, so this
                //is safe-for now. One could have a lenghty
                //discussion about the fact that the ion
                //temperature may also play a role.
                //Liebermann and Lichtenberg discuss
                //this for a sheath. It is probably safe to ignore,
                //because plasmas in the Knudsen regime
                // are very likely to be far from LTE.
        }
        M=m_ParticleMasses[ndx];
        vth=MeanVelocity(T,M);
        m_DKnudsen[ndx]=1.3333333*vth*m_K;

    }
}

REGISTER_PROVIDER
(plPartValTransCalculator, plKnudsenDiffusion, "KnudsenDiffusion");
```

Bibliography

[1] M. Mitchner and C.H. Kruger, editors. *Partially Ionized Gases.* (New York: Wiley & Sons), 1973. 1, 4.3.1, 4.3.3, 6.4.2

[2] E.Esaray. Overview of plasma-based accelerator concepts. *IEEE Trans. Plasma Sci*, 24(252), 1996. 1.2

[3] S.V. Patankar, editor. *Numerical Heat Transfer and Fluid Flow.* New York: McGraw-Hill, 1980. 1.4, 3.2.1, 3.2.2, 3.2.4, 5.1.3

[4] J.A.M. van der Mullen. *Phys. Rep.*, 191:109, 1990. 1.4, 4.2

[5] J.P.Boeuf and L.C.Pitchford. Pseudospark discharges via computer simulation. *IEEE Transactions on Plasma Science*, 9(2), 1991. 2.3

[6] L.C.Pitchford, N. Ouadoudi, J.P.Boeuf, M.Legentil, V. Puech, J.C. Thomaz Jr., and M.A. Gundersen. Triggered breakdown in hollow cathode (pseudospark) discharges. *J. Appl. Phys.*, 78(1), july 1995. 2.3

[7] Vladimir Arsov. *Copper vapour density during the prebreakdown phases of a pseudospark discharge measured with laser induced fluorescence.* PhD thesis, University of Erlangen-Nürnberg, 2001. 2.3

[8] D.A. Benoy. *Ph.D. thesis, Eindhoven University of Technology*, 1993. 3.1

[9] Ger Janssen. *Design of a General Plasma Simulation Model.* PhD thesis, Eindhoven University of Technology, 2001. 3.1, 3.2.2, 11, 6.4.2

[10] Jan van Dijk. *Modelling of Plasma Light Sources — an object-orented approach.* Ph.D. thesis, Eindhoven University of Technology, 2001. 3.1

[11] K.C.Karki and S.V. Patankar. *AIAA Journal*, 27:1167, 1989. 5

[12] J.A.M. van der Mullen. PhD thesis, 1986. 11, 3.3.2

[13] G. Verkerk, J.B. Broens, W. Kranendonk, J.L. Sikkema F.J. van der Puijl, and C.W. Stam, editors. *BINAS.* Wolters-Noordhoff, 1986. 3.3.1, 3.3.1

[14] M.A. Lieberman and A.J. Lichtenberg, editors. *Principles of Plasma Discharges and Materials Processing.* (New York: Wiley & Sons), 1994. 4.2, 7.2.2, 25, 25, 25, 26

[15] D.C. Schram and M.C.M. van de Sanden. Plasma physics lecture notes, 1997. 6.4.2

[16] E.A. Mason and A.P. Malinauskas. *Gas Transport in Porous Media: The Dusty-Gas Model*. Elsevier, 1983. 7.1.2, 25, 7.2.3

[17] E.H. Holt and R.E. Haskell, editors. *Foundations of Plasma Dynamics*. (New York: The Macmillan Company), 1965. 7.2.1, 7.2.1

[18] Gordon E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8), july 1965. A

[19] I.N.Bronstein and K.A.Semendjaev. *Taschenbuch der Mathematik*. Verlag Nauka, 1991. D

Acknowledgements

The work done here could not have been done without the help of many people. At this opportunity, I would like to thank some of them.

First of all, I would like to thank Joost van de Mullen. Much of the insights I gained during this graduation come from him. The early-morning discussions we had provided me with the directions I needed to come to results. My understanding of plasma physics is largely shaped by him. He is also acknowledged for correcting and structuring this report.

My two direct supervisors Bart Hartgers and Kurt Garloff, are acknowledged for the time they spent supervising me. Especially in the beginning, when PLASIMO, numerical plasma physics, Linux and C++ were still areas of which I knew little, their patient supervision was invaluable.

Joost's other students were great co-workers. Michiel, Xiao-Yan, Erik, Wouter, Jan, Harm, Colin and Mark not only were good discussion partners, but also helped provide the pleasant atmosphere among the modelers.

Jeroen Jonkers is acknowledged for providing the interesting challenge of modeling the decay of the plasma in the hollow cathode discharge. While the problem seemed easy enough at first, it prooved to be much more difficult an rich than originally anticipated.

Another important contribution should not be forgotten: the thousends of creators of the free software which I used to create this report and operate my computer. When one receives such useful tools as these for free, the least one can do is say an earnest "Thank You".