Modelling of a fluorescent lamp plasma

Citation for published version (APA):

DOI:
10.6100/IR565333

Document status and date:
Published: 01/01/2003

Document Version:
Publisher’s PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:
• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher’s website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license above, please follow below link for the End User Agreement:
www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:
openaccess@tue.nl
providing details and we will investigate your claim.

Download date: 17. Oct. 2020
Modelling of a Fluorescent Lamp Plasma

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de Technische Universiteit Eindhoven, op gezag van de Rector Magnificus, prof.dr. R.A. van Santen, voor een commissie aangewezen door het College voor Promoties in het openbaar te verdedigen op woensdag 14 mei 2003 om 16.00 uur

door

Albertus Hartgers
geboren te Amsterdam
Dit proefschrift is goedgekeurd door de promotoren:

prof.dr.ir. D.C. Schram
en
prof.dr.ir. G.M.W. Kroesen

Copromotor:
dr. J.J.A.M. van der Mullen

This research is sponsored by the Dutch Technology Foundation STW as project ETN.4441
Contents

1 General introduction ............................................ 7
   1.1 The fluorescent lamp ........................................ 7
   1.2 The model .................................................. 8
   1.3 Outline ................................................... 10

2 An overview of the Plasimo model ......................... 11
   2.1 Introduction .............................................. 11
   2.2 The $\phi$-equation ...................................... 11
   2.3 Outline of the modules ................................. 13
      2.3.1 The grid ........................................... 13
      2.3.2 The flow module .................................. 14
      2.3.3 The electro-magnetic module .................. 14
      2.3.4 The composition .................................. 14
      2.3.5 Discretisation and solver ...................... 16
      2.3.6 Iterations ......................................... 16

3 Using C++ for numerical simulations ...................... 19
   3.1 Introduction .............................................. 19
   3.2 Concept of object oriented programming ............ 19
   3.3 C++ object basics ...................................... 20
   3.4 Recycling: encapsulation and inheritance ........ 23
   3.5 Virtual methods ....................................... 25
   3.6 Comparison to Fortran ................................ 26

4 Implementation ................................................. 29
   4.1 Introduction .............................................. 29
   4.2 Time-dependent discretisation ....................... 29
      4.2.1 Introduction ....................................... 29
      4.2.2 Steady state discretisation ................... 29
      4.2.3 Extension to time-dependent discretisation .. 31
      4.2.4 Implementation ................................... 34
      4.2.5 Numerical Problems ................................ 37
      4.2.6 Conclusions ......................................... 38
   4.3 Runtime configuration ................................... 38
   4.4 The valueref system .................................... 38
      4.4.1 Introduction ....................................... 38
## Contents

4.4.2 The plValueRef .......................... 40
4.4.3 The accessors .......................... 42
4.4.4 plLocalValueMap and plFieldValueMap .. 43
4.4.5 plParticleValue .......................... 43
4.4.6 Const-ness and references ................. 43
4.4.7 Efficiency .......................... 44
4.4.8 Conclusions .......................... 46

4.5 Abstracting dimensionality .................. 46
4.5.1 Introduction .......................... 46
4.5.2 Implementation .......................... 47
4.5.3 Conclusions .......................... 47

5 CRModel: A general collisional radiative modelling code 49
5.1 Introduction .......................... 49
5.2 Theory .......................... 50
5.2.1 Introduction ......................... 50
5.2.2 Collisional processes .................. 51
5.2.3 Solutions .......................... 54
5.3 Rates .......................... 56
5.3.1 Hydrogen approximations ............... 56
5.3.2 Cross section fits ..................... 58
5.4 Program structure ....................... 59
5.4.1 Overview .......................... 60
5.5 Conclusions .......................... 60

6 Modelling an Ar-Hg fluorescent lamp plasma using a 3 electron-temperature approximation 63
6.1 Introduction .......................... 63
6.2 Theory .......................... 64
6.2.1 Introduction ......................... 64
6.2.2 Multi group EEDF .................... 65
6.2.3 Processes .......................... 67
6.2.4 Tail temperature ..................... 70
6.3 Results .......................... 72
6.3.1 Comparison to other calculations ........ 72
6.3.2 Results for an Ar-Hg discharge ........... 73
6.4 Conclusion .......................... 76

7 Implementation of a friction-based self-consistent multi-component diffusion model 77
7.1 Introduction .......................... 77
7.2 Theory .......................... 79
7.2.1 Introduction ......................... 79
7.2.2 Self-consistent diffusion model ........ 81
7.2.3 Ambipolar condition for a single ion .... 83
7.3 Solving the multi-component diffusion problem ....... 84
Contents

7.3.1 Introduction ........................................... 84
7.3.2 Boundary conditions ................................. 84
7.3.3 Implementation in our model ....................... 85
7.4 Results ...................................................... 87
7.4.1 Description of the models ......................... 87
7.4.2 Medium pressure argon discharge ................. 88
7.4.3 A high pressure argon discharge ................. 89
7.4.4 Argon mercury discharge ......................... 89
7.5 Discussion .................................................... 93
7.A A multiple ion Fick-like diffusion description .... 94
7.A.1 Introduction ............................................ 94
7.A.2 Theory ..................................................... 94
7.A.3 Comparison ............................................ 95
7.A.4 Discussion ............................................. 98
8 The influence of de-Maxwellisation on a fluorescent lamp simulation 99
8.1 Introduction ............................................... 99
8.1.1 The fluorescent lamp ............................... 100
8.2 Overview of the model ................................. 101
8.2.1 Introduction ............................................ 101
8.2.2 Fluid transport ....................................... 101
8.2.3 Transport coefficients ......................... 104
8.2.4 Collisional radiative models .................... 104
8.2.5 Transport of radiation .................. 105
8.2.6 Mixture ................................................. 106
8.3 Results ...................................................... 108
8.4 Conclusion ................................................... 112
9 A time-dependent fluorescent lamp simulation 113
9.1 Introduction ............................................... 113
9.2 Setup of the model ................................... 114
9.2.1 Time-dependent modifications ................. 114
9.2.2 Simplifications to the model ................. 115
9.3 Results ...................................................... 118
9.4 Conclusions ............................................... 122
10 General conclusions ......................................... 123
Bibliography ..................................................... 125
Summary ........................................................ 129
Samenvatting ................................................... 131
Dankwoord ....................................................... 133
Chapter 1

General introduction

With the advent of the digital computer, it has become feasible to do numerical simulations of physical phenomena. Such simulations are particularly useful in cases where the mathematical problem is too complex to solve in an analytical manner, or when performing a physical experiment is either too difficult, too expensive or too dangerous. In addition, comparison of the results of simulations with experimental data can help in understanding the physical processes that play a role in the experiment.

This thesis deals with the design and construction of, and the results obtained with, a model of a fluorescent lamp. The decision to model such a lamp was made because of the importance of these discharges (fluorescent lamps account for a significant part of the world’s light production), in combination with their interesting properties.

This model was implemented using the PLASIMO plasma modelling package [1]. Implementing a model of this discharge provides many opportunities for adding improvements and new features to our modelling toolkit, that have turned out to be useful for other applications as well. In this manner, our construction of a fluorescent lamp model contributed to more than only that specific research subject. In addition, the existing PLASIMO infra-structure allows a researcher to easily “play” with various parameters and mechanisms. Finally, the framework allows one to easily extend the fluorescent lamp model with new physical effects, both ones that were implemented specifically for it, and ones that were added for other reasons.

1.1 The fluorescent lamp

Low pressure electric discharges in gasses have been known since at least 1750. Only much later, in the 1920s, it was discovered that a mixture of mercury vapour and a noble gas was able to convert 60% of the electrical energy into ultra violet (UV) light from the single mercury 253.7 nm line. However, it was not yet possible to base a practical visible light-source on this. Two things were still missing: a suitable phosphor that could be put on the inside of the tube walls, to convert the UV into visible light, and electrodes that were both efficient
Chapter 1: General introduction

Fluorescent lamps in their present form have been commercially available since the early 1940s. It is the most energy-efficient lamp in the world that is used on a large scale, and as such it is the most often used man-made plasma. From a physics point of view it has many interesting properties, such as thermal non-equilibrium, a non-Maxwellian electron energy distribution function (EEDF), and (obviously) significant radiation transport. Additionally, many important phenomena occur at the electrodes, were deposition and etching influence the decay of the electrodes. However, this thesis limits itself to the positive column, a part of the plasma that is located between the electrodes.

The traditional standard fluorescent lamp has a cylindrical geometry with a diameter of about 38 mm. It contains 400 Pa of argon buffer gas, and mercury with a vapour pressure of 6 mTorr. The typical current through such a lamp is 400 mA. This leads to an absorbed electrical power of approximately 0.3 W/cm of which about 60% is converted into UV radiation. In turn, this radiation is absorbed and re-emitted as visible light by phosphors on the inside wall of the tube.

Because of their use inside fluorescent lamps, low-pressure Hg-rare gas discharges have been studied for more than fifty years. In this period, the development and optimisation of lamps based on such discharges has benefited enormously from these experimental and numerical investigations. Important experimental achievements in this field are the experiments of Kenty [2], Verweij [3] and Koedam et al [4,5].

One of the first self-consistent numerical models was presented by Waymouth and Bitter [6], who made a self-consistent 0-dimensional numerical implementation of Schottky’s positive column model [7]. This model was followed by many others, of which the more noteworthy contributions are mentioned here. One of these is the model of Winkler et al [8] who included a non-Maxwellian electron energy distribution and multi-step excitation and ionisation. Maya and Lagushenko [9] subsequently presented a model that added radial dependency and radiation transport, while Lister and Coe [10] made the source code of their positive column simulation program GLOMAC available to the scientific community, allowing other researchers to experiment with the physics included in it. Finally the model of Zissis et al [11] uses a self-consistent collisional-radiative model to consider both mercury and argon ionisation and excited states, with a non-Maxwellian EEDF. Additionally, it considers radiation trapping of the mercury lines.

1.2 The model

As mentioned, the simulations presented in this thesis were constructed using the PLASIMO toolkit. This general plasma modelling package has been under development since 1991 by numerous people under the supervision of van der Mullen. After initial modelling work by Beulens et al [12], the first PLASIMO version that was a true plasma simulation model was implemented in the language Pascal.
and later in C by Benoy and de Jong in 1991 [13] to model inductively coupled spectrochemical argon plasmas. After that, Janssen [14] extended the code to study expanding cascaded arc plasmas with more complex mixtures that are not in thermal equilibrium.

The creation of the current PLASIMO-code that is written in C++ and is very modular in nature was started by van Dijk [15], who based his work on the older C version of Janssen and Benoy. This new implementation was then used and extended by others. A radiative transport model was implemented by van der Heijden [16], while Johnston [17] contributed calculations for transport coefficients and Garlof [18] added code to model extreme ultra-violet (EUV) light sources. Finally, code to describe fluorescent lamp-like discharges was added in the course of this thesis work.

The construction of a fluorescent lamp simulation has many different building blocks. Some of these have been implemented specifically in the frame of this work. Examples of these are the construction of a general collisional-radiative model to describe the chemistry, an approximation to take deviations from a Maxwellian electron energy distributions into account, a self-consistent diffusion model, and a time-dependent discretisation method allowing time-dependent behaviour to be modelled. Other parts have been added to the PLASIMO framework by others. For example the basic model and electro-magnetic power incoupling (by van Dijk), radiation transport (implemented by van der Heijden) and transport coefficients (included by Johnston).

Conversely, the additions made for the fluorescent lamp model are being used to simulate other discharges, such as expanding cascaded arcs and capillary discharges. By working in this manner, more advanced models can be constructed more easily, since one builds upon the work of others. The adoption of this strategy for our model has lead to an impressive amount of code, currently well over 150,000 lines distributed over more than 1,000 files. In addition, the recent dramatic increase in performance of personal computers allows simulations to be run that are far more computationally intensive than before.

Combining all the various pieces of code into one simulation results in an extensive fluorescent lamp model. Firstly, it is fully one-dimensional. In principle, a two-dimensional model could easily be constructed, but we have limited ourselves to the one-dimensional case so far. In addition, it contains an accurate description of the various ionisation and excitation processes due to the use of collisional-radiative models, combined with a non-Maxwellian EEDF. It uses an extensive self-consistent diffusion description and a very advanced radiation transport module. Finally, this simulation can be done time-dependently. Almost all parts mentioned here can optionally be omitted or replaced by simpler versions, resulting in a less accurate but faster simulation. Unfortunately, on current PC hardware, this is still necessary in some cases, in order to keep the total running time of the model acceptable.
1.3 Outline

This thesis starts with a broad overview of what is included in the PLASIMO modelling toolkit in chapter 2. The two chapters that follow provide a small tour of the world of numerical modelling. Chapter 3 gives a brief introduction of the programming language C++, in which most of the model has been written, and its applicability to numerical modelling. Chapter 4 then discusses some of the specific additions to the model that have been made for this work. The most important two being a time-dependent discretisation method and a general mechanism for passing plasma data to specific parts of the code. While these chapters provide insight into what is happening “under the hood”, they are not necessary for understanding the physics and models that are presented in the rest of this thesis.

After these introductory chapters follows a series of chapters that build upon each other to form a full model of a fluorescent lamp. Chapter 5, which first appeared as a paper [19], deals with the collisional-radiative modelling code used to describe the chemistry\textsuperscript{1}, while chapter 6, published previously as [20], presents our implementation of a three-electron group model, describing deviations from a Maxwellian electron energy distribution. Chapter 7, originally a paper published as [21], introduces a self-consistent diffusion model based on friction forces. Finally, results for a steady-state and time-dependent fluorescent lamp model are presented in chapters 8 (submitted for publication as [22]) and 9.

\textsuperscript{1}With chemistry we mean any change of particles in general, so excitation and de-excitation in addition to the more usual ionisation, recombination, dissociation and association.
Chapter 2

An overview of the Plasimo model

2.1 Introduction

In order to provide a frame of reference for the chapters that follow, this chapter contains a brief introduction to the various parts of the Plasimo [14–16] modelling platform. Most of the material discussed here is dealt with in more detail either in other publications, or in other chapters of this thesis.

Simulating a physical process generally means finding the solution to a coupled set of partial differential equations (PDEs). These equations describe the rate of change of various quantities (such as densities, energy, momentum) as determined by transport coefficients (for example particle and thermal diffusion coefficients, specific heat), flow, and source terms (for example elastic energy transfer, and production/destruction due to chemical\footnote{Again, chemical is meant here in a broader than usual sense. See footnote 1 on the facing page.} interactions). The PDEs are typically coupled through these transport coefficients and source terms. For example, densities influence the (thermal) diffusion coefficients and specific heat, and thus the temperature (energy), while temperatures in turn affect the species densities through particle diffusion coefficients and chemical interactions. Because these dependencies can be strongly non-linear, it is normally necessary to solve the set of PDEs iteratively, as explained in section 2.3.6 on page 16.

2.2 The $\phi$-equation

The so-called $\phi$-equation is a partial derivative equation (PDE) of a general form, in which all balances for particles, momentum and energy, as well as the mass continuity equation are written. This method of focusing on a single kind of PDE and casting all balances in its form is quite common in the field of computational fluid dynamics, since it allows one to concentrate on an efficient solution procedure for a single equation.

The general PDE solved in our model is similar to the $\phi$-equation used by
a.o. Patankar [23], and is of the form:

\[
\frac{\partial (c\phi)}{\partial t} + \nabla \cdot \left( \vec{U}\phi \right) - \nabla \cdot \left( \Gamma \nabla \phi \right) = S_c + S_p\phi.
\]  

(2.1)

In contrast with Patankar and others, we do not require that \( \vec{U} \) is divergence-free, allowing us to apply this equation to far more problems. Normally, the requirement that \( \nabla \cdot \left( \vec{U} \right) = 0 \) limits \( \vec{U} \) to \( \vec{u} \), where \( \rho \) is the mass density and \( \vec{u} \) is the bulk velocity, which in turn limits the quantities that \( \phi \) can represent. In our case, we have much more freedom in our choice of \( \phi \). As will be explained in section 4.2 on page 29, this is possible because we use a slightly modified discretisation algorithm.

The first term on the left-hand side is the time-dependent term, which will be central to the discussion of time-dependent discretisation in section 4.2. The second term is the convection term, and the third term is the diffusion term. The right-hand side represents a source term. This source term has a linear part \( S_p\phi \), for (parts of) source terms that have a linear dependency on \( \phi \) and a constant part \( S_c \) for terms that do not have this linear dependency. As stated before, all the balance equations in our model are put in the form of equation (2.1), of which we will now introduce some examples.

The first example is the specific particle mass balance:

\[
\frac{\partial n_i}{\partial t} + \nabla \cdot \left( n_i \vec{u}_i \right) - \nabla \cdot \left( D_i \nabla n_i \right) = S_i,
\]  

(2.2)

where \( n_i \) is the density of species \( i \), \( D_i \) its diffusion coefficient, and \( S_i \) is the local particle source term for species \( i \).

The second example is the specific momentum conservation equation for species \( i \):

\[
\frac{\partial (\rho_i \vec{u}_i)}{\partial t} + \nabla \cdot (\rho_i \vec{u}_i \vec{u}_i) = -\nabla \cdot \overline{\tau}_i + \vec{u}_i m_i S_i + \vec{F}_i + \vec{R}_i.
\]  

(2.3)

Here \( \rho_i = n_i m_i \) is the mass density of species \( i \), \( \vec{u}_i \) the velocity of the species, and \( m_i \) its mass. Furthermore, \( \overline{\tau}_i \) is the pressure tensor, \( \vec{F}_i \) is the body force acting on species \( i \), and \( \vec{R}_i \) is the force (per unit volume) caused by friction with other diffusing species [21] (chapter 7).

And the third example is the energy balance

\[
\frac{\partial (n_i \epsilon_i)}{\partial t} + \nabla \cdot (n_i \epsilon_i \vec{u}_i) + \overline{\tau}_i \nabla \cdot \vec{u}_i + \overline{\eta}_i : \nabla \vec{u}_i + \nabla \cdot \vec{q}_i = Q_i,
\]  

(2.4)

where \( \epsilon_i \) is the internal energy of species \( i \), \( \overline{\eta}_i \) is the viscosity tensor, and \( Q_i \) is the heat source caused by elastic and inelastic interactions, chemical interaction with other species, and/or electro-magnetic power incoupling, and \( \vec{q}_i \) the heat flux vector. This heat flux is given by Fourier’s law, \( \vec{q}_i = -\lambda_i \nabla T_i \), were \( \lambda_i \) is the thermal conductivity.

These three equations will be dealt with in more detail in chapters 7 and 8.
2.3 Outline of the modules

In this section, a schematic overview of the various parts that make our modelling platform will be given (see figure 2.1). At this point, it should be mentioned that the current code is very modular, and allows models to be constructed by combining various modules at runtime. This can even be done using an easy to use graphical user interface. In addition, not all modules have to be included at all times. For example, in order to model a simple flow, one would merely need a grid and a flow module, while non-flowing plasma configurations can be simulated without an actual flow-module.

The modules typically generate data fields defined on one or more grids. In other words, they calculate and store values on a discrete number of points distributed over a volume. The modules are interconnected (coupled) because transport coefficients or source-terms required by one module as input are output fields of another. As mentioned before, an iterative solution procedure is used to find a solution for this coupled system.

2.3.1 The grid

The $\phi$-equations introduced in section 2.2 are written in a generalised two-dimensional orthogonal curvilinear coordinate system. This coordinate system provides a mapping between the physical coordinates and an equidistant computational grid. This makes it possible to model a wide range of physical geometries, provided that there exists an axis of symmetry. The grid is generated by map-
ping the physical \((x, y)\) points on their computational \((\xi_1, \xi_2)\) counterparts. In the general case, this can be done by solving a two-dimensional differential Poisson equation, where the boundary conditions determine the actual shape of the grid, as described by Benoy [13]. For the simple cases of Cartesian, cylindrical, and spherical grids, the mapping is generated in a more straightforward manner. It is possible to locally refine the grid, i.e. increase the number of computational points in a certain physical location, if steep gradients are expected.

2.3.2 The flow module

On the grid, a flow module provides a flow field by calculating three velocity components and a pressure. While the calculations presented in this thesis deal with non-flowing configurations (i.e. all velocity components are zero), it is also possible to use real flow solvers based on the Semi-Implicit Method for Pressure-Linked Equations (SIMPLE) and Semi-Implicit Method for Pressure-Linked Equations Revised (SIMPLER) [23] methods. Since this thesis does not deal with flowing configurations, we will refer to [18] for a description.

2.3.3 The electro-magnetic module

If required, the electro magnetic (EM) module calculates electric and magnetic fields on the grid. A number of different modules are provided for this, ranging from a simple electrical current, via a fixed power dissipation in the total volume, to a complex calculation involving combinations of plasmas, coils and ferrites [24]. In addition, an EM module for microwave-based power in-coupling based on the Yee’s algorithm [25] is under development. In our case of a fluorescent lamp model, we have limited ourselves to either a simple current, or a given power dissipation per volume.

2.3.4 The composition

Naturally, it is possible to model a combination of different species, called a composition or mixture. This is done by calculating density-fields on a grid. The two modules currently available allow for either a local thermal equilibrium (LTE) mixture, for which all densities are determined by a single temperature field [16], or a more complex non-LTE mixture, for which all individual species densities are tracked separately. However, in terms of energies, only two temperatures are used, one for the electrons, and one for the other (heavy) species. Figure 2.2 on the facing page gives an overview of what is involved in calculating a composition.

Since the densities of all species are considered individually in the case of non-LTE, the production and destruction due to chemical interactions are explicitly taken into account through source terms of the specific particle balances (equation (2.2)). If we consider the example of an ionisation reaction of argon:

\[ \text{Ar} + e \xrightarrow{K} \text{Ar}^+ + e + e, \]
2.3 Outline of the modules

![Figure 2.2: A schematic overview of the composition module. Examples of the various types of mixtures, chemical interaction calculations, and transport coefficients are shown.](image)

with reaction rate coefficient $K$, we see that this would make a contribution of $n_{Ar}n_eK$ (with $n_{Ar}$ and $n_e$ the argon and electron densities, respectively) to the source terms of the $Ar^+$ and electron density fields. Conversely there would be a contribution of $-n_{Ar}n_eK$ to source terms of the argon density field. Note that the conservation of mass and charge criteria are still fulfilled in this manner. The various reaction rates can be described by simple expressions (a.o. an Arrhenius-type rate coefficient), or the results of collisional radiative models (CRMs, see chapter 5) can be used.

Transport properties, such as diffusion coefficients, and thermal and electrical conductivity, are calculated by a variety of calculation objects that can be used according to need. In addition, radiation transport calculators are used to take the effects of radiation transport through the volume into account. Different modules are provided for different geometries and symmetries, as well as different types of emission and absorption (such as optically thin, optically thick, Doppler broadening and Stark broadening). This is all explained in far greater detail in [16]. For LTE plasmas, diffusion can either be ignored for simple compositions, or an elemental diffusion model [16] can be used to describe more complex cases.

In case of a non-LTE plasma, there are separate diffusion models, such as one describing a simple Fick-type diffusion, an ambipolar diffusion model for a single or multiple ions, or even a self-consistent diffusion model that also takes friction, thermophoretic force and the electric field into account. This sub-model is presented in chapter 7.
2.3.5 Discretisation and solver

The \( \phi \)-equations are discretised using the control volume (CV) method. In this method, the computational domain is divided into control volumes. The nodal grid points, on which the field variables are defined, are centred in each of these control volumes. The fluxes from one control volume to a neighbouring one are defined on the CV boundaries, also called “faces” or “walls”.

The discretised equations are obtained by integrating the \( \phi \)-equations over each control volume. Depending on the Peclet number (\( Pe \)), which is the ratio of the convection and diffusion terms, either a first order upwind scheme (\( |Pe| > 2 \), convection dominance) or a second order central difference scheme (\( |Pe| \leq 2 \), diffusion dominance) is chosen. This discretisation method is further described in [23].

In the case of a time-dependent model, the time-dependent term is discretised as well. Currently, it is possible to choose between three time-dependent discretisation methods, Explicit Euler, Implicit Euler and Crank-Nicolson [26], although it is relatively straightforward to add others. This will be explained in greater detail in section 4.2.

The discretised equations can be solved with a variety of solvers. Apart from the robust Modified five-point Strongly Implicit Procedure (MSIP) [27, 28] that is appropriate for systems that are sensitive to numerical instabilities, there are a number of Krylov subspace based solvers, such as (Bi)Conjugate Gradient (CG, BiCG) and Quasi-Minimal Residual (QMR) solvers [29], that can optionally use a preconditioner.

2.3.6 Iterations

As stated before, all modules are coupled to others, making an iterative solution method necessary. Figure 2.3 on the next page gives a schematic overview of the various interactions. For instance, the flow module is influenced by the mixture, since the production and destruction of species influences densities and temperatures, and thus pressures. The same is true for the EM module, since it determines power dissipation, based on the electrical conductivity resulting from the mixture composition. Finally, the mixture is influenced by both the EM module through the temperature, and the flow module through particle transport. In order to find a solution that satisfies all conditions, the model iteratively calculates the fields for each module in turn, and repeats this process until convergence is reached.
Figure 2.3: A schematic overview of the interactions between the various parts of the model. These interactions make an iterative solution procedure for the whole model necessary. Note that the composition itself typically contains additional couplings between the various species densities through chemical interactions and transport of radiation.
Chapter 3
Using C++ for numerical simulations

3.1 Introduction
Traditionally, many codes in the field of scientific computing have been written in a Fortran dialect. Examples of these are PHOENICS CVD [30], ELENDIF [31] and GLOMAC [10]. Recently however, there has been a tendency to move to the object oriented C++ language [32], for example with OOPIC [33] and NUMLAB [34]. It is interesting to notice that this shift is also happening in most other areas of software development. In this section we will describe some of the benefits and disadvantages of using C++ for numerical simulations. Naturally, many of these arguments are also valid for using C++ or even other object oriented languages to develop non-scientific software.

Especially in the field of plasma physics, where there exists a great diversity in chemical composition and interaction, power incoupling and geometry, it is very advantageous to use object oriented programming (OOP) techniques. In this chapter, we will show that OOP allows us great flexibility in the description of chemical interactions and makes it easy to separate the geometry from the chemistry.

3.2 Concept of object oriented programming
The main advantage of using an object oriented language such as C++ is the strong coupling of data and the functions (called methods in C++) that work on that data. This combination of data and corresponding functions forms an object or class as it’s called in C++. In a way, objects are an extension of the modular programming concept as seen in Pascal [35] and Modula-2 [36]. These languages stimulate a strict distinction between an interface and internal data and/or functions on a module level (i.e. typically per source file). The module contains a few public functions (the interface) that can be called from other modules, typically to modify data in some manner. It also contains a number of other functions, internal to the module, that are guaranteed to only be used
inside the module. In this manner, the module could be completely rewritten, without affecting any other modules, as long as the interface (public functions) does not change. In other words, there exists a separation between the interface that should change as little as possible, since it affects other modules, and the implementation that can be changed without consequence to other parts. The benefit of these languages is not so much that they allow modular programs to be written, since this can be done in almost any language. The great advantage is that these languages limit the manner in which one part can interact with another one, thus making complex systems more easy to understand. At this point, it should perhaps be mentioned that virtually no language denies the possibility to ignore its benefits and write bad programs. Conversely, it is possible to use almost any kind of language to write object oriented code.

As said, object oriented languages can be seen as an extension of this concept, where the separation between interface and implementation is not done at the module level, but at an object level instead. An object is any self-contained piece of data. Examples of these can be a string (a sequence of letters), a stream (an object to access files on a disk) or an object representing an electron density field. By again separating the interface from the implementation, the manner in which one part of a program interacts with another part is much more clear.

While we do not aim at writing a full-fledged C++ course, we will now introduce some basic C++ features. This is necessary in order to make the following chapter more easy to understand without the need of an extensive background in C++ programming.

### 3.3 C++ object basics

Some very basic concepts about the C++ syntax are shown in listing 3.1 on the facing page. It shows two functions, both taking an `int` (integer) as an argument, that declare their own local variables. Variables declared in `function_a` are different from those in `function_b`, although they may be labelled by the same name. Though this example is trivial, most of the concepts that apply to `integer` variables also apply to C++ objects.

Listing 3.2 on page 22 shows a small example of a class (as an object is called in C++) that represents a temperature, demonstrating some of the ideas behind the object oriented approach. In essence, a temperature is a real number that has to be positive. In this way a new variable type is introduced; a `double` which may not be negative.

While a rather simple example, it demonstrates three important features. The first is the separation between interface and implementation. Anything after the `private` keyword cannot be used outside the `Temperature` class. Only `methods` (the functions of a class) defined after the `public` keyword can be used by other parts of the program. Obviously, these public methods may call private methods and use private variables.

Secondly, the temperature is guaranteed to be above zero, since access to the actual variable storing the temperature is limited through the `Set` and `Get`
```cpp
int function_a ( int y )
{
    int x = 1; // statements which declare that x and z are
    int z = 2; // variables of type "int" initialised with
        // values "1" and "2"

    x = 3; // x now contains "3"

    return x + y * z; // this "return" generates the output of
        // function_a
}

// x and z cease to exist here; we say that
// their scope is limited to the internal
// function_a code.

int function_b ( int y )
{
    int x = 4; // function_b 's private x
    int my_y = y; // private my_y becomes a copy of argument y

    return x + my_y * z; // error, z not defined
}
```

Listing 3.1: A simple example of the declaration, assignment and scoping rules for variables in C++. Note that "//" precedes comments in the code (that are not seen by the compiler).
class Temperature
{
    private: // implementation starts here
        double temp;
    public: // interface starts here
        void Set(double t)
        {
            if (t<=0) Error("Non-positive temperature");
            temp=t;
        }
        double Get()
        {
            return temp;
        }
    Temperature(double t) // constructor
    {
        Set(t);
    }
    ~Temperature() // destructor
    
};

Temperature t = 273.16; // compare with listing 3.1, line 18;
// t is a variable of the (new) type Temperature.
Temperature t2 = t; // compare with listing 3.1, line 19.
t.Set(321.14);
t = 321.14; // actually becomes t = Temperature(312.14);
double Celsius = t.Get() - 273.16;
Temperature t1; // error ! A value must be set.
t = -34; // error ! Non-positive value.

Listing 3.2: A simple example of a class, promoting the temperature to a distinguished category with its own characteristics.
methods. Any attempt to set the temperature to a non-positive value will result in an error.\footnote{While temperatures obviously cannot be anything but positive, a calculation yielding a negative temperature is not impossible in the world of numerical simulations.} This means that one can use the temperature without ever worrying about negative temperatures, without studying each and every line of code where a temperature might be modified.

The third noticeable feature is the constructor/destructor. The method \texttt{Temperature(double t)} is called whenever a \texttt{Temperature} object is created, such as on line 24. It makes certain that the temperature is positive by using the \texttt{Set} method. Since the constructor takes one argument (\texttt{double} \texttt{t}), it is an error to create a temperature without directly assigning a value to it. For this reason, the code on line 30 is erroneous. Conversely, the method \texttt{~Temperature()} is called whenever a \texttt{Temperature} object goes out of scope.\footnote{Is no longer accessible, for example outside the function where it was created.} While not doing much in this example, the \texttt{constructor} and \texttt{destructor} are very important in keeping things consistent for more complex classes. Since an object cannot exist, and thus cannot be used, unless its constructor has been called, this mechanism guarantees a consistent state of the object, if the constructor is written properly. And because destructors are guaranteed to be called before an object disappears, they can be used to make sure that things are cleaned up correctly. This cleaning up might include such things as freeing any allocated memory, and closing or removing files.

Finally, the reader’s attention should be drawn to line 28. Here the temperature \texttt{t} is assigned a new value. Since anything assigned to a \texttt{Temperature} should in itself be a temperature\footnote{In this sense, the C++ rules for classes are very similar to the physical rules for dimensions.}, the number \texttt{321.14} is first translated into an anonymous \texttt{Temperature}-object (i.e. an object without a name, or rather, an object not associated with an explicit variable), as shown in the comment on that line. This means that the \texttt{Temperature}-constructor will be called to construct this object, and the check for positive values will be made.

\section*{3.4 Recycling: encapsulation and inheritance}

If a well-written C++ program consists of many self-contained pieces, so-called objects, that have clear non-changing interfaces, it is less risky and even advantageous, to base new, and more complex objects on simpler ones. The two mechanisms that C++ provides for this are called encapsulation and inheritance, both of which will be illustrated here.

Listing 3.3 on the following page gives an example of encapsulation. We use a \texttt{Position} object to specify a certain point in three-dimensional space, and then use that object to specify the position of the \texttt{Particle}. In this way, the \texttt{Position} object is encapsulated by the \texttt{Particle}. Like a position, a particle also has coordinates that can be questioned and changed (via the \texttt{GetX} and \texttt{SetX} methods).
class Position
{
    private:
    double x, y, z; // the three local symbols labelling the
    // coordinates of position within the class

    public:
    double GetX() const { return x; }
    void SetX(double xx) { x = xx; }
    // Imagine Get and Set versions for Y and Z going here.
    Position(double xx, double yy, double zz) // constructor
    {
        // Note that xx, yy, zz are the labels with which the
        // coordinates are passed to the constructor,
        // while x, y, z are the coordinates inside the class
        SetX(xx); SetY(yy); SetZ(zz);
    }
    // if not explicitly specified, a default destructor will
    // be generated automatically
};

class Particle
{
    private:
    double mass;
    Position pos;

    public:
    Position GetPos() const { return pos; }
    void SetPos(Position p) { pos = p; }
    double GetX() const { return pos.GetX(); }
    void SetX(double x) { pos.SetX(x); }
    // Again, imagine Get and Set versions for Y and Z going here.
    Particle(double m, Position p)
    {
        SetMass(m); SetPos(p);
    }
};

Listing 3.3: An example of encapsulation. A Position object is encapsulated in the Particle class.
3.5 Virtual methods

An interesting concept of C++ is the virtual method. This is a method that can be overridden in derived classes. This powerful concept allows us to modify behaviour of code after it has been written! Listing 3.5 on page 27 shows an example of this. On line 22, a calculation is done with a reaction rate coefficient that is always zero. Below line 23, a class called Arrhenius is derived from ReactionRateCoeff that provides a rate calculation using the well-known Arrhenius-expression given by

\[ K(T) = \sqrt{T} \exp \left( -\frac{E}{kT} \right), \]  

with \( T \) the temperature, \( E \) the energy difference between the initial and final state, and \( k \) Boltzmann’s constant.
When a *Arrhenius* object is passed to the calculation function on line 42, an entirely different calculation will be done than that suggested by lines 1-19. Note that while this example contains the *ReactionRateCoeff* and *Arrhenius* classes on the same page, the *Arrhenius* class could, in principle, be written centuries after the *calculate_reaction_rate* function has been written, and things would still work as expected! So the use of inheritance and virtual functions allows us to modify the behaviour of existing code in a controlled manner. In addition, it should be pointed out here that it is possible to modify the behaviour of the *calculate_reaction_rate* function at runtime, not just at compile time.

A small extension to the concept of a virtual method is the *pure* virtual method. In the previous example, the *ReactionRateCoeff* class was defined with a *RateCoeff*-method *returning* 0. Since this kind of reaction rate is arguably not very useful, it would be preferable to indicate that descendants of the *ReactionRateCoeff* class should have a *RateCoeff* method, but not to specify an implementation in the *ReactionRateCoeff* class. This can be done by declaring the method as a pure virtual method, with the syntax shown in listing 3.6 on page 28. In contrast with the *RateCoeff* method in listing 3.5 on the facing page, this method has a “zero” implementation instead of an implementation returning 0.

Since there is no implementation in *ReactionRateCoeff* of the *Rate* method, objects of this type cannot be created. The class now only serves to specify an interface, or to be used as a base of which to derive other classes. For this reason, a class with pure virtual methods is also called an *abstract* base class.

### 3.6 Comparison to Fortran

As mentioned before, numerical codes have traditionally been written in Fortran [37]. The main advantages over using Fortran over using C++ are primarily that there are many pre-existing Fortran codes [38], that can be incorporated when writing new code, and, secondly, that code written in Fortran generally runs faster.

The first advantage can be put in perspective by the fact that it is relatively easy to link Fortran code to C++ code, allowing the use of existing Fortran algorithms where it would benefit. If a particular piece of Fortran is found to be very useful, it could even be wrapped in a C++ class, allowing it to be easily reused. While it is true that this might require some adaption of data structures used in the C++ code to the Fortran code, it would most likely also be necessary to change the structure of one’s own Fortran code when combining it with (Fortran) code of others.

Concerning the second point, it is true that current Fortran compilers produce slightly faster numerical code than current C++ compilers. This is mainly due to the fact that there has been an emphasis on using Fortran for numerical applications for many years, leading to the development of highly optimised compilers and even processors and machine architectures for doing numerical calculations in Fortran. However, there is no fundamental reason for Fortran
3.6 Comparison to Fortran

class ReactionRateCoeff
{
    public:
        virtual double RateCoeff( Temperature t ) { return 0; }
        // default constructor and destructor are automatically used
    }

void calculate_reaction_rate ( ReactionRateCoeff * rc )
    // this function has the new type "ReactionRateCoeff" as an argument
{

    // n1, ne and temp are calculated here somewhere
    ...
    double R = n1*ne*rc->RateCoeff( temp )
    // the RateCoeff function of the rc object is called
    ...
    // do something with R
}

ReactionRateCoeff r; // a variable of the new type is declared

calculate_reaction_rate ( &r ); // very simple

class Arrhenius : public ReactionRateCoeff
{
    double energy;

    public:
        virtual double RateCoeff( Temperature t )
        {
            return sqrt(t.Get())*exp(-energy/(kBoltzmann*t.Get()));
            // the C++ equivalent of equation (3.1)
        }

        Arrhenius( double en )
        {
            energy = en;
        }
};

Arrhenius arr ( 3.2e-19 ); // Energy of 3.2e-19 Joule (2 eV)

calculate_reaction_rate ( &arr ); // more complex but looks the same

Listing 3.5: An example of a virtual method. The class Arrhenius is derived from
the base class ReactionRateCoeff. In doing so, the RateCoeff method on line 4 is
replaced (for Arrhenius objects) by the one on line 28.
Chapter 3: Using C++ for numerical simulations

class ReactionRateCoeff
{
    public:
        virtual double RateCoeff( Temperature t ) = 0;
};

Listing 3.6: An example of a pure virtual method. Note the difference with line 4 of listing 3.5.

to be faster than C++, it is simply the fact that C++ compilers have not yet been fine tuned as much as their older and more mature Fortran counterparts. The only exception to this is the use of virtual functions in C++, which incurs a slight overhead as compared to function calls in Fortran.

However, the small cost of using a slightly slower language should be compared to the significant gain in flexibility and reduction in both development time and bug finding time that C++ brings. Certainly in an economic sense, the use of C++ is advantageous. Profiling of our code showed that 50% to 90% of the calculation time is spent in doing matrix inversions, that are already highly optimised. So rewriting the whole model in Fortran could not possibly yield more than a factor 2 in performance (the factor 2 would be in the unlikely case that the runtime of all non-matrix inversion code would become zero). A more realistic difference in performance would probably be of the order of 10%, which is a small price to pay for the added benefits of C++.

As mentioned in the introduction, most of arguments in favour of C++ mentioned here, also apply to other object oriented languages, such as Modula-2, Delphi. However, C++ has significant benefits over these languages. Firstly, C++ is an open ISO standard, not something created by a single author or company. Secondly, perhaps for reasons related to the first point, C++ is used far more commonly than these other languages, and C++ code is portable to a great many architectures and operating systems. And finally, C++ has its roots in C, the language in which the UNIX operating system was written. While C++ promotes nice and object-oriented code, it is still possible to trade in some niceties for dirtier code that gets things done faster.
4.1 Introduction

This chapter will go into more detail on some of the features that have been implemented in the PLASIMO model in the course of this thesis work. The two most important additions discussed here are time-dependent discretisation, and a general method for referring to specific data at grid points. Finally, some preliminary work to make large parts of the modelling code less dependent on the dimensionality of the problem is discussed.

4.2 Time-dependent discretisation

4.2.1 Introduction

In the past, the PLASIMO model could only model steady-state situations. Recently, the ability to do time-dependent discretisation has been added. This allows us to solve a time-dependent version of the aforementioned $\phi$-equation, and thus to use time-dependent PDEs. In turn, this makes it possible to model time-dependent phenomena. In order to do this, some changes have been made to the original structure of the model, which will be described in the next sections.

4.2.2 Steady state discretisation

As shown in section 2.2, all equations used in our model are of the following form:

$$\frac{\partial (c \phi)}{\partial t} + \nabla \cdot (U \phi) - \nabla \cdot \left( \Gamma \nabla \phi \right) = S_c + S_p \phi. \quad (2.1)$$

We will ignore the time-dependent and source terms for now, in order to focus on the spatial discretisation first. Doing this, the only terms left in equation (2.1) are:

$$\nabla \cdot (U \phi) = \nabla \cdot \left( \Gamma \nabla \phi \right). \quad (4.1)$$
Integration of this equation over the control volume shown in figure 4.1 results in

$$A_E (U_x \phi)_e - A_W (U_x \phi)_w + A_N (U_y \phi)_n - A_S (U_y \phi)_s = A_E \left( \Gamma \frac{d\phi}{dx} \right)_e - A_W \left( \Gamma \frac{d\phi}{dx} \right)_w + A_N \left( \Gamma \frac{d\phi}{dy} \right)_n - A_S \left( \Gamma \frac{d\phi}{dy} \right)_s,$$

(4.2)

where the the $e, w, n, s$ subscripts signify the values at the east, west, north or south boundary of the control volume, respectively, while the $A$’s refer to the areas at the control volume boundaries.

As shown in [23], we can expand this further to

$$\frac{1}{2} A_E (U_x)_e (\phi_E + \phi_P) - \frac{1}{2} A_W (U_x)_w (\phi_P + \phi_W) +$$
$$\frac{1}{2} A_N (U_y)_n (\phi_N + \phi_P) - \frac{1}{2} A_S (U_y)_s (\phi_P + \phi_S) =$$
$$A_E \left( \Gamma \frac{d\phi}{dx} \right)_e - A_W \left( \Gamma \frac{d\phi}{dx} \right)_w +$$
$$A_N \left( \Gamma \frac{d\phi}{dy} \right)_n - A_S \left( \Gamma \frac{d\phi}{dy} \right)_s +$$

(4.3)

We can now rewrite this as

$$a_P \phi_P = a_E \phi_E + a_W \phi_W + a_N \phi_N + a_S \phi_S,$$

(4.4)

where

$$a_{X \in \{E,W,N,S\}} = D_X (D_X, C_X),$$

(4.5)

and

$$a_P = \sum_{X \in \{E,W,N,S\}} D_X (D_X, -C_X).$$

(4.6)

Here, $D_X$ refers to a discretisation function which depends on the diffusion and convection terms $D_X$ and $C_X$. These terms are given for all sides by

$$C_X = A_X U_X$$

(4.7)
4.2 Time-dependent discretisation

\[ D_E = \frac{A_E}{\Delta x_E} \Gamma_e, \quad D_W = \frac{A_W}{\Delta x_W} \Gamma_w, \quad D_N = \frac{A_N}{\Delta y_N} \Gamma_n, \quad D_S = \frac{A_S}{\Delta y_S} \Gamma_s. \]  

(4.8)

It should be noted here that the sign of the \( C_X \) term in equation (4.6) is opposite to that of equation (4.5). This is in contrast with a.o. Patankar, where \( a_P = \sum_{X \in [E,W,N,S]} a_X = \sum_{X \in [E,W,N,S]} D_X \), which is only valid in the case of a divergence free convection term, so that \( C_E = C_W \) and \( C_N = C_S \). In our case, we do not assume such a convection term, so we have to correctly consider the sign of the \( C_X \) term.

The discretisation functions \( D_X \) that correspond to equation (4.3) are

\[ D_{X \in [E,N]}(D_X, C_X) = D_X - \frac{C_X}{2}, \]

\[ D_{X \in [W,S]}(D_X, C_X) = D_X + \frac{C_X}{2}. \]  

(4.9)

However, as pointed out in [23], using this description might give rise to problems when the convection term \( |C| \) exceeds \( 2D \), since \( D_X(D_X, C_X) \) will be negative in this case. In our implementation this was resolved by using the so-called Hybrid Scheme [23], given by

\[
\begin{align*}
D_{X \in [N,E]}(D_X, C_X) = & \begin{cases} 
-C_X & P \leq -2 \\
D_X - \frac{C_X}{2} & -2 < P < 2 \\
0 & P \geq 2
\end{cases} \\
D_{X \in [S,W]}(D_X, C_X) = & \begin{cases} 
0 & P \leq -2 \\
D_X + \frac{C_X}{2} & -2 < P < 2 \\
C_X & P \geq 2
\end{cases}
\end{align*}
\]  

(4.10)

where \( P \) is the Peclet number, defined here as

\[ P = \frac{C_X}{D_X}. \]  

(4.11)

4.2.3 Extension to time-dependent discretisation

If we now return to the time-dependent formulation, and fully discretise equation (2.1), we obtain:

\[
\frac{(c^{n+1}_P p^{n+1}_P - c^n p^n_P)}{\Delta t_{n+1}} + A_E (U_x \phi)_e - A_W (U_x \phi)_w + A_N (U_y \phi)_n - A_S (U_y \phi)_s = A_E \left( \Gamma \frac{d \phi}{d x} \right)_e - A_W \left( \Gamma \frac{d \phi}{d x} \right)_w + A_N \left( \Gamma \frac{d \phi}{d y} \right)_n - A_S \left( \Gamma \frac{d \phi}{d y} \right)_s + (S_C + S_P \phi_P) V_P,
\]  

(4.12)

where \( V_P \) is the volume of the control volume surrounding \( P \). Here we discretised the time into small time steps \( \Delta t_i \). The superscript \( n \) in this equation denotes the value of the variable at time

\[ t(n) = \sum_{i=0}^{n} \Delta t_i. \]  

(4.13)
Equation (4.12) expresses that changes in time of $\phi_P$ are caused by transport (either convective or diffusive) from neighbouring volumes and/or local sources. The goal is now to find an expression for this rate of change.

A first step is to rewrite equation (4.12) to a shape similar to that of equation (4.4):

$$a_{P}^{n+1} \phi_{P}^{n+1} = \theta \left( a_{E}^{n+1} \phi_{E}^{n+1} + a_{W}^{n+1} \phi_{W}^{n+1} + a_{N}^{n+1} \phi_{N}^{n+1} + a_{S}^{n+1} \phi_{S}^{n+1} \right) + (1 - \theta) \left( a_{E}^{n} \phi_{E}^{n} + a_{W}^{n} \phi_{W}^{n} + a_{N}^{n} \phi_{N}^{n} + a_{S}^{n} \phi_{S}^{n} \right) + b_{P}.$$ \hspace{1cm} (4.14)

As can be seen, the factor $\theta$ determines the mix of “past” and “present” contributions to $a_{P}^{n+1}$.

For a time-dependent discretisation, $a_{P}$ and $b_{P}$ are given by

$$a_{P}^{n+1} = \sum_{X} D_{X} (D_{X}^{n+1}, -C_{X}^{n+1}) - S_{P}^{n+1} V_{P} + c_{P}^{n+1} \frac{V_{P}}{\Delta t_{n+1}},$$ \hspace{1cm} (4.15)

and

$$b^{n+1} = S_{C}^{n+1} V_{P} + \frac{c_{P}^{n} \phi_{P}^{n}}{\Delta t_{n+1}} V_{P}. \hspace{1cm} (4.16)$$

The coefficients for the neighbouring points, $a_{X \in \{E,W,N,S\}}$ are still given by equation (4.5). As can be seen, this discretisation is identical to the steady-state discretisation for the case of $\Delta t \rightarrow \infty$. We will now present three time-dependent discretisation methods.

**Explicit Euler Method**

The explicit Euler method, or simple explicit method, simply bases the change of $\phi$ on the values at time $n$, or,

$$\frac{c_{P}^{n+1} \phi_{P}^{n+1} - c_{P}^{n} \phi_{P}^{n}}{\Delta t_{n+1}} = F(\phi^{n}).$$ \hspace{1cm} (4.17)

This is equivalent to taking $\theta = 0$ in equation (4.14).

The accuracy of this method decreases for larger values of $\Delta t$. Since we relate the change in $\phi$ made in time-step $n+1$ to the value of $\phi$ at time $t(n)$, we effectively approximate

$$\int_{t(n)}^{t(n+1)} \phi(t) dt \approx \phi(t(n)) \Delta t,$$ \hspace{1cm} (4.18)

as shown in figure 4.2 on the facing page. Because the value of $\phi^{n}$ is known when calculating $\phi^{n+1}$, no iterations inside a time-step are necessary in contrast with the other methods. However, due to numerical instabilities, the value of $\Delta t$ is limited to

$$\Delta t < \frac{1}{2 \max \left( \frac{U_{x}}{\Delta x}, \frac{U_{y}}{\Delta y} \right)^{2} + \max \left( \frac{U_{x}}{\Delta x}, \frac{U_{y}}{\Delta y} \right)}$$ \hspace{1cm} (4.19)

in this case.
4.2 Time-dependent discretisation

Because of the explicit nature of this discretisation, the accuracy of the solution is entirely determined by the magnitude of $\Delta t$. There is no implicit coupling of the $\phi$-equations anymore. It was found that a typical model with multiple coupled $\phi$-equations would quickly become inconsistent for even very small time-steps. For this reason the explicit Euler method in this form was found to be of little use in our modelling. Only for very simple problems, or for fields with small rates of change, this method could be used.

Implicit Euler Method

In case of the simple implicit method or implicit Euler method, we base the change of $\phi$ in time-step $n + 1$ on the diffusive, convective and source terms at time $t = t(n + 1)$, or,

$$\frac{c_p^{n+1} \phi_P^{n+1} - c_p^n \phi_P^n}{\Delta t_{n+1}} = F(\phi^{n+1}).$$  \hspace{1cm} (4.20)

In other words, we take $\theta = 1$ in equation (4.14). While this method is numerically stable for any size of $\Delta t$, the accuracy again decreases for larger values of $\Delta t$. In this case, we relate the change in $\phi$ made in time-step $n + 1$ to the value of $\phi$ at time $t(n + 1)$, approximating

$$\int_{t(n)}^{t(n+1)} \phi(t) dt \approx \phi(t(n+1)) \Delta t,$$  \hspace{1cm} (4.21)

as shown in figure 4.3 on the next page. This method is especially useful in the case of exponential decaying behaviour, since it is a reasonable approximation for this case. In comparison with the steady-state discretisation method, this discretisation introduces an extra source-term, that constitutes the contribution from the past. Finding a solution for time $n + 1$ is essentially the same as finding a steady-state solution where this extra term is taken into account.
Chapter 4: Implementation

Crank-Nicolson Method

An improvement can be made in some cases by basing the change of \( \phi \) on the average of the other terms at time \( n \) and time \( n + 1 \), or,

\[
\frac{c_P^{n+1} \phi_P^{n+1} - c_P^n \phi_P^n}{\Delta t_{n+1}} = F(\phi^n, \phi^{n+1}). \tag{4.22}
\]

This can be done by using \( \theta = \frac{1}{2} \) in equation (4.14). The approximation

\[
\int_t^{t(n+1)} \phi(t) dt \approx \frac{\phi(t(n)) + \phi(t(n + 1))}{2} \Delta t, \tag{4.23}
\]

is more accurate than the one used by the implicit Euler method (see figure 4.4). However, the Crank-Nicolson method may give rise to numerical oscillations for very large time-steps.

4.2.4 Implementation

All objects in PLASIMO that have something to do with time-dependent behaviour are derived from the class \( plTimeAware \), shown in listing 4.1 on the next page.
4.2 Time-dependent discretisation

```cpp
class plTimeAware
{
public:
    /// Set the time to travel for the next timestep */
    virtual void SetNextTimeStep( REAL deltaT );
    /// Undo the current timestep. */
    virtual void UndoCurrentTimeStep( void );
    /// Prepare for the next timestep. */
    virtual void PrepareNextTimeStep( void );
    /// Returns true when time stepping is mandatory. */
    virtual bool NeedTimeStepping( void ) const;
    /// Returns the maximum time step that this particular
    /// field variable will stand.
    */
    virtual REAL MaxTimeStep( void ) const;
    /// d−tor
    virtual ~plTimeAware() {};
};
```

Listing 4.1: Abstract base class for all objects that have an interest in time-dependence.

At the beginning of each time-step, the method `PrepareNextTimeStep` is called for all objects, allowing objects to get ready for the next time-step. For example, all $\phi$ variables will store a copy of themselves, allowing their old value at $t = t(n)$ to be retrieved when $b^{n+1}$, that depends on values at time $n$, needs to be calculated.

To calculate the new value of the $\phi$-variables, the normal iterations will be done until a certain accuracy is reached, similar to the ones done in steady-state situation. When this is complete, all `plTimeAware` objects are questioned about the maximum $\Delta t$ that they allow, through their `MaxTimeStep` method. This maximum value depends on a configurable maximum change in the fields and any limits on the time-step determined by the discretisation algorithm.

If the $\Delta t$ that was used was larger than is acceptable, its value is reduced to the maximum value, all objects are asked to revert to their state before the time-step, through the `UndoCurrentTimeStep` method, and the iteration is re-tried.

In case the $\Delta t$ is smaller than the maximum allowed value for four sequential time-steps, $\Delta t$ is increased. Especially in the case of relaxation-behaviour, this speeds up the model considerably, since the rate of change typically decreases exponentially with time, allowing larger $\Delta t$’s to be used when time progresses. Naturally, the range of allowed $\Delta t$ value is also limited by a user-specified minimum and maximum.

Figure 4.5 on the following page shows the time-dependent iteration process schematically.
Figure 4.5: A flowchart showing the most important steps in the time-dependent solution strategy.
4.2 Time-dependent discretisation

<table>
<thead>
<tr>
<th>Maximum</th>
<th>1.7976931348623157 $\times 10^{308}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum</td>
<td>2.2250738585072014 $\times 10^{-308}$</td>
</tr>
<tr>
<td>$\epsilon$</td>
<td>2.2204460492503131 $\times 10^{-16}$</td>
</tr>
</tbody>
</table>

Table 4.1: Limits for double precision floating point numbers as specified by ANSI/IEEE Std 754-1985.

4.2.5 Numerical Problems

A far too common problem in numerical codes is the occurrence of calculation errors due to the finite precision of the hardware. Typically, real numbers (also called floating point numbers) are internally represented by a sign, a mantissa and an exponent. Where the width of the mantissa determines the precision (“the number of digits after the point”), the width of the exponent determines the range of the number. The difference between precision and range can best be illustrated by an example. Table 4.1 lists three numbers describing a certain floating point type. The Maximum and Minimum specify respectively the largest and smallest numbers that can be represented by the machine, while $\epsilon$ is the smallest number that can be added to 1 yielding a representable result different from 1. In other words, if the hardware would add a number smaller than $\epsilon$ to 1, the result would still be 1.

From this it is clear that attention should be paid to the way calculations are done. Divisions of large numbers by small numbers may produce a result too large to be represented, leading to an overflow condition, while the division of a small number by a large one can underflow and produce a result that is either rounded down to zero, or represented by a special value, named not-a-number or NaN. It is clear that continuing a calculation with the results from these divisions might give unwanted results.

For this reason, modern hardware can be programmed to terminate the program when such a situation occurs. In many cases, this is the sensible thing to do, since the results of any further calculations might not have any significance anymore. However, it is also possible to continue the calculation ignoring the fact that numerical errors did occur.

Since the dynamic adjustment of $\Delta t$ required support for reverting to the beginning of a time-step, it was easy to implement code that checks whether any overflow of underflow occurred after each iteration. If there were any problems, the model is restored to its state at the beginning of the current time-step, and $\Delta t$ is reduced. This is done under the assumption that the numerical problem occurred due to too large changes of fields in the iteration, and that those changes are reduced when $\Delta t$ is made smaller. After that, the calculation for the current time-step is restarted. This mechanism is also shown in figure 4.5.
4.2.6 Conclusions

In order to be able to simulate time-dependent phenomena, it was necessary to implement a time-dependent discretisation of the general \( \phi \)-equation. For this reason, a framework was constructed that, in principle, allows any kind of time-dependent discretisation to be added to the model. While only the three simplest methods were actually implemented, it is straightforward to add more advanced methods [39]. However, due to the loosely coupled and non-linear nature of the many \( \phi \)-equations in use in the PLASIMO model, we do not expect a large improvement in accuracy or performance, and did not try any other methods.

In addition, the presence of the framework made it easy to add code that responds in a useful manner to any numerical over- and underflow problems that may occur. This makes time-dependent calculations very robust, and makes it very unlikely that a model “explodes” due to numerical problems.

4.3 Runtime configuration

A key concept of the PLASIMO model is its ability to configure it at runtime. Since most aspects of this have already been covered in [15], we will only provide a brief overview here. At key points in the model, actual functionality is abstracted by using classes with virtual methods, similar to the example given in section 3.5. Among other things, it is possible to change the mixture type (LTE, non-LTE), the diffusion model, the flow model, the EM model, rate coefficients for reaction rates, transport coefficient calculators, boundary conditions, discretisation methods and matrix solvers. Since it is possible to select these at runtime, it is relatively easy to construct a model using a graphical user interface (GUI).

By using a self-registering object technique [15, 40], and combining this with the dynamic loading of shared libraries, it is possible and even straightforward to add new code and functionality to the model without knowledge of most of its inner workings.

4.4 The valueref system

4.4.1 Introduction

In the PLASIMO model, much (if not all) of the runtime configuration is implemented through pointers to abstract base classes. Derivatives of these classes provide actual functionality to the model through virtual functions. An example of such a construction is shown in listing 4.2 on the facing page. A reaction calculation can be done with any kind of reaction rate. All that needs to be done is to construct a derived class from the \textit{plReactionRateCoeff} class, implement a new \textit{RateCoeff()} method for it, and pass a pointer to such an object to the calculation function.

Although this is an elegant approach, something is lacking here. Typically, reaction rates depend on parameters such as densities and temperatures. So,
4.4 The valueref system

Figure 4.6: A schematic overview of how the \textit{plValueRef} refers to values at a certain point, and the way the various accessors are used to access this data. The two \textit{plPerParticleAccessors}, “density” and “temperature” refer to values that are associated with particles, while \textit{plNonParticleAccessors} (“conductivity” and “pressure”) are used for values that are not directly related to particles.

\begin{verbatim}
class plReactionRateCoeff
{
    public:
        virtual double RateCoeff( double T ) = 0;
};

double CalculateWithRateCoeff( plReactionRateCoeff *rc )
{
    // Te is a temperature
    return rc->RateCoeff(Te)*42; // meaningless calculation ...
}
\end{verbatim}

Listing 4.2: Example of a framework for handling reactions, that assumes that reaction rates are only depending on a temperature. Note that there is a convention in \textit{plasimo} to prefix names of classes with “pl”. 
Chapter 4: Implementation

class plReactionRateCoeff
{
    public:
        virtual double RateCoeff(double ne, double T) = 0;
};

double CalculateWithRateCoeff(plReactionRateCoeff *rc)
{
    ...
    return rc->RateCoeff(ne,Te)*42; // another meaningless calculation ...
}

Listing 4.3: Example of a more advanced base class for handling reactions, that anticipates a density and temperature dependence of reaction rates.

in order to support this, we need to adjust our plReactionRateCoeff class, and anything that either uses it or is derived from it, to accept and pass a temperature and density. Listing 4.3 shows this improved version.

This approach allows for many types of reaction rates, so it might be used in a model. Imagine however, that at a certain point in time, someone wants to calculate a reaction rate coefficient involving something besides a density and a temperature. For instance, an electron energy distribution in the form of an electron tail temperature.

At this point, there are two options. The first one is to add another parameter to the plReactionRateCoeff class, and adjust all descendants of this class, and anything that uses it. In the case where many reaction rate coefficient calculators derived from the previous version of the plReactionRateCoeff class were already programmed, this could involve a lot of work, touch a lot of code, and, potentially, introduce many bugs.

The other option would be to create a new (derived) plReactionRateCoeffWithElectronTailTemperature class and add support for it in the cases where such a reaction rate coefficient is actually needed. This will restrict the use of this rate coefficient, and destroy the simple concept of using a general kind of rate coefficient.

Both solutions complicate the model and make it necessary to change working code, instead of focusing on implementing more physics. Therefore, the following section will discuss an alternative way of passing data to derived classes that does not have this problem.

4.4.2 The plValueRef

In order to solve the difficulties mentioned above, we implemented a C++ class named plValueRef. This is a new type of variable that makes it possible to refer to values on any grid point. The action of this new type is depicted in figure 4.6.

To understand the function of the new variable type, one should realize that the
class plReactionRateCoeff
{
    public:
    virtual double RateCoeff(const plValueRef & point) = 0;
};

// A somewhat strange rate that depends on the square-root of the pressure.

class plPressureRate : public plReactionRateCoeff
{
    public:
    virtual double RateCoeff(const plValueRef & point)
    {
        extern plNonParticleAccessor Pressure; // extern,
        // since this plNonParticleAccessor
        // is global to the model
        return 42 * sqrt(Pressure(point));
    }
};

Listing 4.4: Example of the use of the plValueRef class. In contrast to listings 4.2 and 4.3, there is no a priori dependence of reaction rates on a certain set of values. Through the plValueRef object, any value can be used in the RateCoeff methods of classes derived from plReactionRateCoeff. The plPressureRate is an example of this.

PLASIMO model at this moment consists of a number of two dimensional fields, with values at each grid-point. A plValueRef instance refers to the values of all those fields, at a certain grid-point. All functions or members that need to access one or more of those values can refer to these through the plValueRef.

To access a specific variable, for example the pressure, through the plValueRef object, an accessor-object is necessary. Instances of such objects know how to get at a value of a quantity using a plValueRef. Listing 4.4 shows a trivial example of a plValueRef in action.

As can be seen, the pressure is referenced through an external instance of the plNonParticleAccessor class (explained later on), named “Pressure”. At the construction of the model, the exact same variable “Pressure” is associated with a field that contains the pressure. This association is used to retrieve the correct field from a plValueRef for each accessor. Obviously, for this to work, all parts of the model must agree on a single accessor for the pressure, hence it is referred to as extern, since it is defined globally.

The benefit of this approach now becomes clear. If we rewrite our plReactionRateCoeff class to accept a plValueRef instead of a list of values, any descendant of plReactionRateCoeff can do its calculation with any or all of the fields that the model “knows” about. This is true even for fields that were not yet conceived of when the plReactionRateCoeff class was implemented.

Note that the plReactionRateCoeff class does not rely on any field explicitly, and thus does not have to be changed when new fields are introduced. Effectively,
**double local_pressure ( const plValueRef & point )**

```cpp
extern plPerParticleAccessor Temperature;
extern plPerParticleAccessor Density;
double p=0;
for( int i=0; i < number_of_particles ; ++i)
    p+=Density(point)[i] * kBoltzmann * Temperature(point)[i];
return p;
```

Listing 4.5: Example of the use of the `plPerParticleAccessor` to calculate the pressure.

only the code that directly accesses a certain field, either because it needs to use a value, or because it calculates the field, is dependent on it.

### 4.4.3 The accessors

Apart from the `plNonParticleAccessor`, there are two other accessors, named `plPerParticleAccessor` and `plGeneralAccessor`. As seen before, the `plNonParticleAccessor` is used to get a plain and simple value. However, the `plPerParticleAccessor` refers to values that are potentially unique for each particle-type, such as the densities and temperatures of that species. An example of its use is shown in listing 4.5. As can be seen, an extra integer referring to the species is now necessary to get an actual value.

It is possible to associate the same field with multiple particles for a given `plPerParticleAccessor`. This is used in the case of a plasma in thermal equilibrium, where all particles have the same temperature. Naturally, the resulting code is a little less efficient than code that explicitly uses the fact that there is only one temperature. However, being able to use the same code for all cases clearly has other benefits.

The third kind of accessor, the `plGeneralAccessor`, refers to a single value, just as the `plNonParticleAccessor`. Alternatively, it can refer to a `plPerParticleAccessor`-value for a certain particle. As the name suggests, it may be regarded as more general than the other two accessors.

The `plGeneralAccessor` is typically used in cases where the actual variable to use is unknown at compile time, for example because the user specifies it at runtime. Examples of its use are a mathematical expression typed at runtime by the user, or a general lookup-table where the indices to be used are determined when loading the table. In both cases one might use a pressure (`plNonParticleAccessor`) or an electron density (`plPerParticleAccessor`), or something else altogether.
4.4 The valueref system

Listing 4.6: Example of the use of the `plGeneralAccessor`.

```cpp
int calculate_something ( const plValueRef & point )
{
    extern plNonParticleAccessor Pressure;
    extern plPerParticleAccessor Density;

    plGeneralAccessor Something1=Pressure;
    plGeneralAccessor Something2=Density[0];

    ...

    double s1=Something1(point); // same as s1=Pressure(point);
    double s2=Something2(point); // same as s2=Density(point)[0];

    ...
}
```

4.4.4 plLocalValueMap and plFieldValueMap

The `plLocalValueMap` and `plFieldValueMap` are the glue that keeps everything together. When a model is constructed, fields (for a multi-dimensional model) or values (for a 0-dimensional model) are registered in a `plFieldValueMap` or a `plLocalValueMap`, respectively, with reference to the accessors. Afterwards, the maps can return `plValueRef` objects. The value that an accessor returns from a `plValueRef` is the value that was registered using the same accessor in the map that produced the `plValueRef`. Listings 4.7 on the following page and 4.8 on page 45 show examples of their use.

4.4.5 plParticleValue

The `plParticleValue` is an object that refers to a set of particle-dependent values. The usual examples for this are densities and temperatures. In a way, the `plParticleValue` is what a `plPerParticleAccessor` gets out of a `plValueRef`. This object can be passed to functions that need access to those kinds of values. Listing 4.9 on page 46 shows an example of this.

4.4.6 Const-ness and references

Defining parameters as `const` is a common and useful practice in C++. By doing so, one tells the compiler that one does not intend to change the parameter. This allows the compiler to catch errors, i.e. when the parameter is changed, and perhaps sometimes generate more efficient code based on knowledge of the fixed value of the parameter.

For this reason, applying the const-ness concept to the objects mentioned here is a good idea. However, it proved impossible to reliably handle this case using the `const` keyword. The reason for this is that the `plValueRef` and `plParticleValue`
\texttt{extern} plNonParticleAccessor Pressure;
\texttt{extern} plPerParticleAccessor Density;

\texttt{plLocalValueMap vmap;}

\texttt{// register the variable containing the pressure}
\texttt{double press;}
\texttt{vmap.Register ( Pressure, press );} // register a double
\texttt{// for the pressure}

\texttt{// register a variable to put the density in for each particle}
\texttt{double dens[ number_of_particles ];}
\texttt{for ( int i=0; i < number_of_particles; ++i) }
\texttt{vmap.Register ( Density[i], dens[i] );} // register a double
\texttt{// for each density}

...\texttt{double x=CalculateSomePressureThing( vmap() );} // vmap() returns
\texttt{// a plValueRef}

...\texttt{...}

Listing 4.7: Example of the use of the \texttt{plLocalValueMap}, that contains values for a single point.

classes are in effect pointers or references to other data. Applying the \texttt{const} keyword to them only tells the compiler that it should disallow modifications of the \texttt{plValueRef} and \texttt{plParticleValue} objects themselves, not what they are referring to\textsuperscript{1}. For this reason, the \texttt{plValueRef} and \texttt{plParticleValue} objects have constant counterparts named \texttt{plConstValueRef} and \texttt{plConstParticleValue}, that disallow modification of the values they refer to.

All these objects are normally passed by reference, to allow overloading of the functionality in descendant classes (through virtual methods). Since these objects themselves are not changed, they are usually passed via const references, as in \texttt{const plParticleValue & dens} or \texttt{const plConstValueRef & point}.

4.4.7 \textbf{Efficiency}

This discussion would not be complete without spending some words on the subject of efficiency. Naturally, the benefits of the presented abstractions are somewhat offset by a certain extra runtime-overhead in memory usage and execution time. However, this overhead is small compared to the large amount of time typically spent on modelling-calculation such as matrix inversions.

\textsuperscript{1}At first sight, it seems possible to use the \texttt{const} keyword in this case. However, the necessity to supply a copy-constructor (taking a \texttt{const} reference) would allow the conversion of a const-type object to a non-const-type object.
4.4 The valueref system

```c
extern plNonParticleAccessor Pressure;
extern plPerParticleAccessor Density;

plFieldValueMap vmap;

// register the field containing the pressure
plField press;
vmap.Register ( Pressure, press ); // register a field
// for the pressure.
// (compare with line 9
// of listing 4.7)

// register a field to put the density in for each particle
plField dens[ number_of_particles ];
for( int i=0; i < number_of_particles ; ++i)
    vmap.Register ( Density[i], dens[i] ); // register fields
    // for the densities
    // (compare with line 15
    // of listing 4.7)

// register a field to put the density in for each particle
...
for( int x=0; x<vmap.n1(); ++x) for( int y=0; y<vmap.n2(); ++y)
    double x=CalculateSomePressureThing( vmap(x,y) );
    // vmap(x,y) returns a plValueRef
...
```

Listing 4.8: Example of the use of the plFieldValueMap, that contains fields.
Chapter 4: Implementation

4.4.8 Conclusions

The set of objects described here provide an elegant way to access quantities needed in modelling-applications. Apart from the already mentioned benefit of interfaces that do not have to change when new fields and dependencies are introduced, it also removes knowledge of the dimensionality of the model from large parts of the code. All the code using a plValueRef can remain the same no matter the number of dimensions in the model.

In turn, this greatly enhances the possibility to modularise the code, i.e. to reduce interdependencies between the various program parts. Because of this, many extensions and alterations can be made without changing other code, thus allowing improvements and aiding stability at the same time.

4.5 Abstracting dimensionality

4.5.1 Introduction

As mentioned before, the PLASIMO model uses a two-dimensional grid. Recently however, effort has been made to abstract the dimensionality, with the long-term goal to allow one- or three-dimensional grids. Some configurations that have been modelled using PLASIMO were essentially one-dimensional. Apart from this, it is sometimes advantageous to use simple one-dimensional models for verification or to estimate startup values for more complex models.

It is possible to describe such systems in the current model by demanding

```cpp
double local_pressure ( const plParticleValue & dens,
                        const plParticleValue & temp )
{
    double p=0;
    // iterate over all particles
    for( int i=0; i<dens.size(); ++i)
       p+=dens[i]*kBoltzmann*temp[i];
    return p;
}

double twice_pressure ( const plValueRef & point )
{
    extern plPerParticleAccessor Temperature;
    extern plPerParticleAccessor Density;
    // Note that here there is no number for the species
    // after Density or Temperature.
    return 2* local_pressure ( Density(point ),
                               Temperature(point ));
}
```

Listing 4.9: Example of the use of the plParticleValue.
4.5 Abstracting dimensionality

\begin{verbatim}
for( int i=0; i<density.n1(); ++i)
  for( int j=0; j<density.n2(); ++j)
    pressure (i,j) = density (i,j) * kBoltzmann * temp(i,j);

Listing 4.10: Explicitly two-dimensional code.
\end{verbatim}

\begin{verbatim}
for( pos_iter pos = pressure.begin (); pos!= pressure.end (); ++pos)
  pressure (*pos) = density (*pos) * kBoltzmann * temp(*pos);

Listing 4.11: Dimension-independent code.
\end{verbatim}

zero gradients along one essentially unused axis, through appropriate boundary conditions. Unfortunately, it is still necessary to use at least three grid volumes along the unused axis, resulting in three times more grid points than necessary. In addition, it was found that some gradients would still appear along the unused axis, and slowly disappear while the model reached convergence, needlessly slowing it down.

For these reasons it was decided that it would be beneficial to make the modelling code directly support one and, if possible, three-dimensional configurations.

4.5.2 Implementation

Most of the abstracting has been done by replacing code similar to that of listing 4.10 by that of listing 4.11. Where the first loop explicitly considers a \(N_1\) by \(N_2\) structure for the fields, the second example introduces a type called a \texttt{pos_iter}, short for position iterator.\(^2\) It is easy to grasp that the variable \texttt{pos} goes over the pressure field from the \texttt{begin()} -point to the \texttt{end()} -point, calculating the pressure at each position.

The second syntax has a number of benefits over the first one. The actual meaning of the code is more clearly expressed. Errors, such as exchanging \(i\) and \(j\), are less likely to occur. And finally, the code is independent of the actual structure of the fields and grid.

4.5.3 Conclusions

Changing to an one-dimensional or three-dimensional configuration will not require any changes to code (re)written to use position iterators. It is even possible to abandon the concept of a structured grid and replace it by an unstructured one without having to change the pressure calculation. Obviously, changes of dimensionality will require changes to the boundary condition code, if the number of boundaries change. Though it is not yet possible to change the number of dimensions, an important step in this direction has been made.

\(^2\)In C++, an iterator is a well-known concept for an object that iterates over a collection of data, selecting each element in turn [41].
Chapter 4: Implementation

Whether the switch to different dimensionality should be a compile-time or a runtime option remains to be seen. Clearly, it would be a very useful and convenient option to be able to adapt the dimensionality of the modelling code to the dimensionality of the problem at runtime. However, since iterating over fields is a very common operation, the usual way of introducing abstracting through virtual methods is most likely not sufficiently efficient. On the other hand, it might be possible to exploit the fact that all fields, whether they are one-, two- or three-dimensional are all stored linearly in memory. Obviously, some kind of provision will have to be made for the boundary conditions, since the number of boundaries depends on the number of dimensions.
Chapter 5

CRModel: A general collisional radiative modelling code

Abstract

This paper describes CRModel, a general collisional radiative modelling code written in C++. Collisional radiative models are 0-dimensional plasma models, used to calculate atomic state distribution functions for one or more species, as a function of particle densities and temperatures. They normally work under the assumption that the two main types of processes that cause an atom to change its excited state are radiative processes and electron collisions.

The model uses a flexible plain text input file to describe all relevant aspects of the considered plasma region, including collisional cross sections and radiative decay probabilities. This allows the model to be used for a wide range of atomic or ionic plasma species. The types of results that the model generates, also specified in the input file, include atomic state densities and ionisation and recombination coefficients. CRModel was developed on Linux/UNIX workstations, but can also be used on the PC (i.e. Windows) platform. This chapter is a slightly modified form of [19], via which the source code of the collisional radiative model presented here can be obtained.

5.1 Introduction

Collisional radiative models can be an useful tool in understanding experimental observations, for instance to relate an observed spectrum to an atom state distribution function, and thus to an electron density and temperature. In addition, a collisional radiative model can be used to calculate effective ionisation and recombination rates, and the effective emissivity, as a function of electron temperature and density. These results can then be used in a plasma transport model to calculate local ionisation, recombination and radiation effects.

In this work, we present our C++ collisional radiative model code. At the moment, this code can model a large variety of plasmas that consist of a single atomic species with any number of excited states in between a ground state and
the next ionised ground state. The only restriction is that the density of the excited states is small as compared to the ground state or ion state density.

The modelling program uses a flexible input file to define the various atomic states, radiative probabilities and collisional cross sections, thus specifying the actual model. Because of this flexibility, it is possible to describe a variety of atomic plasmas and to investigate the effects of different cross section expressions. In addition, plasma parameters such as the electron density and temperature, and the atom density can be specified in this file. Finally, the desired type of output can be selected.

5.2 Theory

5.2.1 Introduction

In a collisional radiative model, results are obtained by solving a particle-balance for all considered states. To reduce the complexity of the model, several atomic states are usually combined into a single level where it is assumed that all states in the same level are equally distributed. The state density $\eta(p)$ is related to the level density $n(p)$ through $\eta(p) = n(p)/g(p)$, where $g(p)$ is the statistical weight of level $p$.

The relation between temporal and spatial relaxation of the density $n(p)$ of level $p$ and its collisional/radiative destruction and production is given by

$$\frac{\partial n(p)}{\partial t} + \nabla \cdot (n(p) u(p)) = \left( \frac{\partial n(p)}{\partial t} \right)_\text{CR}. \quad (5.1)$$

For most of the considered levels, it can be assumed that the change in density is entirely dominated by radiative and collisional excitation to and de-excitation from other levels, so that the left part of (5.1) can be ignored.

$$0 = \left( \frac{\partial n(p)}{\partial t} \right)_\text{CR}. \quad (5.2)$$

This assumption is valid if the time constants for the radiative or collisional processes are small in comparison to the time constants of other, transport related, source and sink terms. Under these conditions, the densities of these levels are directly linked to the densities of other levels through the radiative and collisional processes. Because of this, these levels are named dependent levels. In the physical sense, they are dominated by local chemistry, and for that reason also known as LC levels.

For most atomic plasmas, this condition holds for all excited levels. For the lower levels, radiation lifetimes are typically much smaller than transport times ($10^{-7}$ s vs. $10^{-4}$ s), and for the higher excited levels, where radiation lifetimes are larger, the more efficient collisional (de-)excitation by electrons usually satisfies the small lifetime requirement.
For some levels, the simplification of equation (5.2) is not valid. In the model, their densities should be considered as given. These levels are called the independent levels. Since they are dominated by transport, they are also known as transport sensitive (TS) levels. Typical examples of these are the ground states of the atom and ion. In this article, the independent level densities will be denoted by \( n_p \) and the dependent level densities by \( n(p) \).

When the total number of atoms in excited levels is much smaller than the number of ground level atoms or ions, and all excited levels are dependent (on the independent atom and ion ground level), we can apply the quasi steady state solution (QSSS) \([42, 43]\). It is assumed that the distribution of excited levels is instantaneously coupled to the densities of the atom and ion ground levels. The densities of these two independent levels can be considered constant. Note that in situations where certain excited levels cannot be considered dependent, it is still possible to use the QSSS, if the densities of these excited levels are known and used as inputs in the model.

In the remaining of this article will assume that the QSSS conditions apply and that electron-atom collisions are the dominating collision type. These conditions apply to most low temperature plasmas.

### 5.2.2 Collisional processes

**Excitation and de-excitation**

A collision between an atom \((X)\) and an electron \((e)\) may cause the atom to change its state. In this process, the two particle system \(X-e\) gains or loses an amount of kinetic energy equal to the difference in energy between the two states. In schematic form:

\[
X_p + e + (E_{pq}) \leftrightarrow X_q + e.
\]

The probability of an electron collision causing an atom to excite from \(p\) to \(q\) is determined by a cross section \(\sigma_{qp}(E)\), that usually is a function of the electron energy \(E\). The reaction coefficient \(K(p, q)\) for this transition can be obtained by integrating the cross section with the product of the electron energy distribution function \(f(E)\) and the electron velocity \(v_e(E) = \sqrt{2E/m_e}\), where \(m_e\) is the electron mass:

\[
K(p, q) = \int_{E_{pq}}^{\infty} \sigma_{pq}(E)f(E)v_e(E)\,dE. \tag{5.3}
\]

Here the threshold \(E(p, q)\) is the energy difference between level \(p\) and level \(q\).

The coefficient for the corresponding de-excitation reaction, \(K(q, p)\) can be obtained through the principle of detailed balancing. In case of a thermodynamic equilibrium and electron-atom collisions, the Boltzmann equation applies:

\[
\frac{n^B(q)}{g(q)} = \frac{n^B(p)}{g(p)} \exp\left(-\frac{E_{pq}}{kT_e}\right), \tag{5.4}
\]

with \(k\) the Boltzmann constant and \(T_e\) the electron temperature. Because of the equilibrium, the number of transitions from \(p\) to \(q\) must be equal to the number
of reverse transitions:

\[ n_e n(p) K(p, q) = n_e n(q) K(q, p). \]  

\[ (5.5) \]

Combining this with (5.4), we obtain the following relation between \( K(p, q) \) and \( K(q, p) \):

\[ K(q, p) = \frac{g(p)}{g(q)} K(p, q) \exp \left( \frac{E_{pq}}{kT_e} \right). \]  

\[ (5.6) \]

A more general relation between \( K(q, p) \) and \( K(p, q) \) can be found by relating \( \sigma_{qp} \) to \( \sigma_{pq} \). For thermal equilibrium, the electron energy distribution function is the Maxwell distribution function \( f^m \):

\[ f^m(E) = \frac{2}{(\pi kT_e)^{3/2}} \sqrt{E} \exp \left( -\frac{E}{kT_e} \right). \]  

\[ (5.7) \]

Reapplying the principle of detailed balancing as in (5.5), but only for electrons with an energy between \( E \) and \( E + dE \), yields

\[ n_e n(p) \sigma_{pq}(E) v_e(E) f^m(E) dE = n_e n(q) \sigma_{qp}(E - E_{pq}) v_e(E - E_{pq}) f^m(E - E_{pq}). \]  

\[ (5.8) \]

Combining this with (5.4) and (5.7) results in

\[ \sigma_{qp}(E) = \frac{g(p)}{g(q)} \frac{E + E_{pq}}{E} \sigma_{pq}(E + E_{pq}). \]  

\[ (5.9) \]

Since the cross sections do not depend on the presence of thermodynamic equilibrium, equation (5.9) is valid under all conditions. By combining this with equation (5.3), it is possible to calculate \( K(q, p) \) for any electron energy distribution function, provided that \( \sigma_{pq} \) is known.

**Ionisation and recombination**

If enough energy is available, an electron-atom collision can cause ionisation of the atom. On the other hand, the reverse process, a collision between an atom and two electrons, can cause recombination of the ion and an electron.

\[ X_p + e + (E_{p+}) \leftrightarrow X_+ + e + e. \]

When there is equilibrium between the ionisation and the three particle recombination, the Saha relation holds:

\[ \frac{n(p)}{g_p} = \frac{n^S(p)}{g(p)} = \frac{n_e n_+}{g_e g_+} \left( \frac{\hbar^2}{2n_e m kT_e} \right)^{3/2} \exp \left( \frac{E_{p+}}{kT_e} \right), \]  

\[ (5.10) \]

where \( n^S(p) \) denotes the Saha density of level \( p \), and \( \hbar \) is Planck’s constant.

Another recombination mechanism is the so-called two particle recombination, where an electron is captured and a photon emitted,

\[ X_+ + e \rightarrow X_p + h\nu. \]
Although this mechanism only requires two particles, it is less important than the three-particle recombination at not too low densities for reasons of energy and momentum conservation. In [42] an expression can be found for the two particle recombination rate from the ion level to a bound level $p$:

$$n_e n_+ \alpha_{+p} = n(p)^{S \gamma} Z^4 p_p^{-5} \int_{\epsilon_p}^{\infty} \frac{\exp{-\epsilon}}{\epsilon} h(p, \epsilon/e_p) d\epsilon,$$

(5.11)

$$\epsilon_p = \frac{E_{p+}}{k T_e},$$

(5.12)

where $\alpha_{+p}$ is the recombination coefficient,

$$p_p = Z \sqrt{R / |E_{1p}|},$$

(5.13)

the principle quantum number of level $p$, $Z$ the number of independent electrons (1 for hydrogen-like atoms), $R$ the Rydberg energy and $h(p, y)$ the Gaunt Factor, equal to 1 in reasonable approximation.

**Emission and absorption**

Emission and absorption of radiation also contribute to the production and destruction of levels. The three processes involved are spontaneous emission, absorption and stimulated emission,

$$X_p \xrightarrow{A(p,q)} X_q + h \nu_{pq}$$

$$X_p \xrightarrow{\rho_v B(q,p)} X_q + h \nu_{pq}$$

$$X_p + h \nu_{pq} \xrightarrow{\rho_v B(p,q)} X_q + h \nu_{pq} + h \nu_{pq}$$

where $\rho_v$ is the spectral energy density, and $A(p, q)$, $B(q, p)$ and $B(p, q)$, the spontaneous emission probability, the absorption probability coefficient and the stimulated emission probability coefficient, respectively.

A problem with the treatment of radiation absorption and stimulated emission is the non-local nature of these phenomena. Because of radiation, one part of the plasma can be influenced by another part. Through the introduction of an escape factor $\Lambda(p, q)$, and the replacement of the transition probability $A(p, q)$ by $A(p, q) \Lambda(p, q)$, these effects can be partially accounted for.

The escape factor $\Lambda(p, q)$ is obtained by integrating over the line profile:

$$\Lambda(p, q) = 1 - (n(p) A(p, q))^{-1} \int (n(q) B(q, p) - n(p) B(p, q)) \rho_v d\nu,$$

(5.14)

Table 5.1 on the following page shows the relation between $\Lambda$ and the dominating radiative processes. The escape factor approximation is only correct in case the mean free path of the photon is small relative to the gradient lengths in the plasma, or, trivially, when there is no absorption.
Absorption and stimulated emission can be neglected.  
No net radiative transfer.  
Absorption supersedes local emission.  
Stimulated emission dominates.

Table 5.1: Relation between $\Lambda$ and the dominating radiative process.

5.2.3 Solutions

The right hand term of equations (5.1) and (5.2) can be divided in a production part and a destruction part:

$$\left( \frac{\partial n(p)}{\partial t} \right)_{CR} = P(p) - n(p)D(p). \quad (5.15)$$

where $P(p)$ is the collisional and radiative production rate, and $D(p)$ the destructive factor.

The production $P(p)$ is given by:

$$P(p) = n_e \sum_{q \neq p} n(q)K(q,p) + \sum_{q > p} n(q)A(q,p)\Lambda(q,p) + n_e n_+ (n_e K(+,p) + A(+,p)), \quad (5.16)$$

and the destruction $D_p$ by:

$$D(p) = n_e \sum_{q \neq p} K(p,q) + \sum_{q < p} A(p,q)\Lambda(p,q) + n_e K(p,+) \quad (5.17)$$

Here $K(+,p)$ and $A(+,p)$ are the rates for three and two particle recombination and $K(p, +)$ is the ionisation rate.

Combining equations (5.2, 5.15, 5.16, 5.17) results in the following expression for the dependent levels:

$$n_e \sum_{q \neq p} n(q)K(q,p) + \sum_{q > p} n(q)A(q,p)\Lambda(q,p) + n_e n_+ (n_e K(+,p) + A(+,p))$$

$$-n(p)n_e \sum_{q \neq p} K(p,q) - n(p) \sum_{q < p} A(p,q)\Lambda(p,q) - n(p)n_e K(p,+) = 0. \quad (5.18)$$

In principle it is possible to solve this set of linear equations for $n(p)$. However, since, in principle, an unlimited number of excited levels exist, a limit should be imposed. In order to reduce the amount of levels that have to be considered explicitly and still obtain accurate results, an analytical cut-off procedure, as introduced by Van der Mullen [42] is used for the higher excited levels. According to this author, the highest excited level (cut-off level) should be chosen so that it, and all excited levels above it, are both collisionally dominated and in the so-called hot region, where the average electron energy is larger than the ionisation energy of the level. Under these circumstances, collisional excitation
5.2 Theory

rates \( K(p,q) \) from a level below to a level above the cut off level, can be taken into account as an extra ionisation contribution from level \( p \) using:

\[ K^*_p = K(p,q) \left( 1 - \frac{p^2 p^6}{p^6} \right). \] (5.19)

For a system where the only two independent levels are the ion and atom ground states, the set of linear equations (5.18) can be rewritten as

\[ \sum_{q>1} C_{qp} n(q) = n_+ C_{+p} + n_1 C_{1p}, \] (5.20)

where \( C_{qp} \) is a function of \( K(p,q), A(p,q), \Lambda(p,q), T_e \) and \( n_e \). Since \( n_1 \) (the ground level density) and \( n_+ \) (the ion ground level density) are independent variables, the density of a level \( p \) can be split in atom and ion ground level contributions:

\[ n(p) = n^+(p) + n^1(p), \] (5.21)

\[ n^+(p) = r^+(p)n^S(p), \] (5.22)

\[ n^1(p) = r^1(p)n^B(p), \] (5.23)

introducing \( r^+(p) \) and \( r^1(p) \), the so-called relative population coefficients. The contributions \( n^+(p) \) and \( n^1(p) \) can be found by solving (5.20) for \( n_1 = 0 \) and \( n_+ = 0 \), respectively. In this way, \( r^+(p) \) and \( r^1(p) \) can also be obtained.

It is now possible to write (5.1) for the atom ground level as

\[ \frac{\partial n(p)}{\partial t} + \nabla \cdot (n(p) \vec{u}(p)) = n_e n_+ J_e(+,1) - n_e n_1 J_e(1,+), \] (5.24)

in which the recombination coefficient \( J_e(+,1) \) from the ion state to the ground state is given by

\[ J_e(+,1) = n_e K(+,1) + A(+,1) + \sum_{q>1} \frac{n_q^S r^+_q}{n_e n_+} (n_e K(q,1) + A(q,1) \Lambda(q,1)), \] (5.25)

and the ionisation coefficient \( J_e(1,+) \) from the ground state to the ion state by

\[ J_e(1,+) = K(1,+) + \sum_{q>1} \frac{n_q^B r^1_q}{n_e n_1} (n_e K(q,+) - A(+,q) \Lambda(+,q)). \] (5.26)

Note that many other publications use \( \alpha_{CR} \) and \( S_{CR} \) to denote the recombination and ionisation coefficients. However, our notation is more general, since it allows for more independent levels than just the ground and ion state.

From equations (5.25) and (5.26), it is clear that the recombination is coupled with the \( r^+ \) coefficient and the ionisation with the \( r^1 \) coefficient. It can also be seen that there is no direct dependence of either \( J_e(+,1) \) or \( J_e(1,+) \) on \( n_1 \) or \( n_+ \). This makes it convenient to use these coefficients in a plasma transport model where only ground level atoms, electrons and ions are considered. The ionisation
and recombination coefficients can then be used to calculate the transformation flow from ground level atoms to ions and vice-versa.

For systems which contain more independent levels besides the atom and ion ground level, the previous can be generalised by introducing effective transformation rate coefficients \( J_c(p, q) \) between the independent levels, just as \( J_c(+, 1) \) is an effective rate coefficient from the ion ground level to the atom ground level and \( J_c(1, +) \) is a rate coefficient in the other direction.

Using the \( n^p(q) \) contributions, it is also possible to introduce the specific effective emissivity terms \( L_p \), describing the contribution of each independent state \( p \) to the total radiation energy flux out of the plasma:

\[
 n_p L_p = \sum_q n^p(q) \sum_{r < q} A(q, r) \Lambda(q, r) E_{pq} \tag{5.27}
\]

### 5.3 Rates

In order to solve the set of linear equations (5.18), knowledge of the radiation probabilities \( A(p, q) \), the escape factor \( \Lambda(p, q) \) and the transition probabilities \( K(p, q) \) (or \( \sigma_{pq} \)) is necessary. Values for \( A(p, q) \) can be easily found in literature whereas \( \Lambda(p, q) \) depends on the dimensions of the modelled plasma. For \( \sigma_{pq} \), either semi-theoretical expressions based on a hydrogen approximation or experimental data can be used. This section describes the different types of cross section approximations that can be used in the presented code.

The modelling code numerically integrates the product of these cross sections with a Maxwellian energy distribution function, using a 34-point Gauss-Legendre method. The numerical integration, as opposed to a faster analytical evaluation, is used because it allows greater flexibility in both the cross section expressions and, perhaps in the future, the energy distribution functions. Since the typical runtime of the model is about 100 ms as it stands, the extra processing due to the numerical integration is not a big concern. For all cross section expressions used in the model, the accuracy of this method is sufficient. Other, not so smooth, cross section expressions might require a different integration method, but this is relatively easy to implement through the C++ mechanism of overloading, which makes it convenient to use a more specific integration method in certain cases.

#### 5.3.1 Hydrogen approximations

Vriens and Smeets suggest using the following cross section with \( \alpha = 3.25 \) for the ionisation of excited atoms and ground-state alkali atoms [44]:

\[
 \sigma_{p+}(E) = \frac{\pi e^4}{E + \alpha E_{p+}} \left( \frac{5}{3 E_{p+}} - \frac{1}{E} - \frac{2 E_{p+}}{3 E^2} \right), \tag{5.28}
\]

where \( e \) is the electron charge, and \( E \) the electron energy.
In the same article, they present an expression for excitation cross sections:

\[
\sigma_{pq}(E) = \frac{2\pi a_0^2 R}{E + \gamma_{pq}} \left[ A_{pq} \ln \left( \frac{E}{2R} + \delta_{pq} \right) + B_{pq} \right],
\]

where \(a_0\) is the Bohr radius,

\[
A_{pq} = \frac{2R}{E_{pq}} f_{pq},
\]

with \(f_{pq}\) the absorption oscillator strength,

\[
B_{pq} = \frac{g_q}{g_{sub}} \frac{4R^2}{p_q^3} \left( \frac{1}{E_{pq}^2} + \frac{4E_{pq}^2}{3E_{pq}^3} + b_p E_{pq}^2 \right),
\]

with

\[
b_p = \frac{1.4 \ln p_p}{p_p} - 0.7 \frac{0.51}{p_p^2} + \frac{1.16}{p_p^3} - \frac{0.55}{p_p^4},
\]

\[
\delta_{pq} = \exp \left( -\frac{B_{pq}}{A_{pq}} \right) - 0.4 \frac{E_{pq}}{R},
\]

and

\[
\gamma_{pq} = R \left[ 8 + 23 \left( \frac{p_q - p_p}{p_p} \right)^2 \right] \left( 8 + 1.1 \frac{p_q - p_p}{p_p} \right) + \frac{0.8}{(p_q - p_p)^2} + \frac{0.4}{\sqrt{p_q - p_p}} \left| p_q - p_p - 1 \right|^{-1}.
\]

The factor \(g_q/g_{sub}\) is an extension of the original formula. It is used to correct for the fact that the (principal) quantum levels of hydrogen are split into multiple levels in the case of non-hydrogen atoms. For this reason, the cross sections must be weighted in such a way that the sum of all \(g_q/g_{sub}\) factors for transitions from a given start level \(p\) to a certain principal quantum level equals unity.

The weight \(g_{sub}\) is equal to \(g_q\) if \(p\) and \(q\) have the same principal quantum number. If \(p\) and \(q\) do not have the same principal quantum number, then \(g_{sub}\) should be equal to the sum of the statistical weights of all the levels with the same principal quantum number as level \(q\), to which an excitation (i.e. gain of energy) from level \(p\) is possible.

If the transition probability \(A(q,p)\) is known, the absorption oscillator strength \(f_{pq}\) can be obtained using

\[
A(q,p) = \frac{2\pi e^2 g_p}{m_e c \hbar^2 \lambda_{pq}^3 g_q} f_{pq},
\]

where \(c\) is the speed of light in vacuum, \(\lambda_{pq}\) the wavelength between states \(p\) and \(q\), and \(\epsilon_0\) the vacuum permittivity.

Drawin [45, 46] proposes the following expressions for three types of cross sections (with \(\tilde{E}_{pq} = E/E_{pq}\)). For allowed transitions:

\[
\sigma_{pq}^a(E) = 4\pi a_0^2 \left( \frac{R}{E_{pq}} \right)^2 f_{pq} g(\tilde{E}_{pq}),
\]
Figure 5.1: The collision cross section described by equation (5.42).

where

\[ g(\hat{E}_{pq}) = \alpha_{pq} (1 - \hat{E}_{pq}^{-1}) \hat{E}_{pq}^{-1} \ln(1.25\beta_{pq} \hat{E}_{pq}), \]  

(5.38)

for parity forbidden transitions ($\Delta l \neq \pm 1, \Delta s = 0$):

\[ \sigma_{pq}^p(E) = 4\pi a_0^2 \left( \frac{R}{E_{pq}} \right)^2 Q_{pq}^p (1 - \hat{E}_{pq}^{-1}) \hat{E}_{pq}^{-1}, \]  

(5.39)

and spin forbidden transitions ($\Delta l = \pm 1, \Delta s \neq 0$):

\[ \sigma_{pq}^s(E) = 4\pi a_0^2 \left( \frac{R}{E_{pq}} \right)^2 Q_{pq}^s (1 - \hat{E}_{pq}^{-2}) \hat{E}_{pq}^{-3}. \]  

(5.40)

Here $\alpha_{pq}$, $\beta_{pq}$, $Q_{pq}^p$ and $Q_{pq}^s$ are constants obtained from experiments or the literature ($\alpha_{pq}$ and $\beta_{pq}$ are approximately equal to 1).

5.3.2 Cross section fits

Experimentally determined integrated cross sections are often presented in the literature. In order to use these data, the following expressions for $\sigma_{pq}(E)$ can be fitted to the experimental data.

Step fit

The simplest approximation of a cross section is a step-function:

\[ \sigma_{pq}(E) = \begin{cases} 0 & \text{for } E < E_{pq}, \\ \sigma & \text{for } E \geq E_{pq}. \end{cases} \]  

(5.41)

Pots fit

This fit is an extension of a simple step function (see figure 5.1).

\[ \sigma_{pq}(E) = \begin{cases} 0 & \text{for } E < E_{pq}, \\ \sigma_1 + s_1 (E - E_{pq}) & \text{for } E_{pq} \leq E \leq E_1, \\ \sigma_2 + s_2 (E - E_1) & \text{for } E_1 < E. \end{cases} \]  

(5.42)

Note that $s_1 = (\sigma_2 - \sigma_1)/(E_1 - E_{pq})$ and $s_2$ are the slopes of the lines in figure 5.1.
5.4 Program structure

To conserve space, only a general overview of the various objects/classes in the model, and the source files it consists of, shall be presented here. For more information, the reader is directed to the generated documentation included with
Chapter 5: CRModel: A general collisional radiative modelling code

The program, or to the comments in the source code. Both can be found through [19].

5.4.1 Overview

Since the collisional radiative model is described by a set of linear equations, the software simply has to construct a matrix representing these equations, and then invert this matrix. For this reason, the CRMatrixBuilder is a key object in the model. It is the abstract base class (as explained in section 3.5) of seven “builder” classes (see figure 5.4) that construct the matrix using data specified in an input file. By letting the various builders process the matrix consecutively, the full set of linear equations is created.

The CRIonRateBuilder uses equation (5.28) to calculate ionisation rates for the excited levels. The CRMatTransBuilder takes the various types of collision induced transitions into account, using a list of objects describing the cross sections. Since the CRMatTransBuilder overwrites matrix elements, it is possible to let the CRIonRateBuilder calculate default ionisation rates, and override these later through the CRMatTransBuilder when specific ionisation cross sections are known. The three particle recombination of equation (5.10) is taken into account by the CRThreePartBuilder, and radiative (two particle) recombination through the CRRadCombBuilder, using equation (5.11). Finally, the CRLinearBuilder adds the radiation probabilities and creates the actual linear system.

All these builders can be added to a linked list managed by a CRModel object, that is also a derivative of CRMatrixBuilder itself, through the CRMatrixBuilderGroup. The CRModel uses the “builder” objects to construct the linear equation matrix, solves it and calculates the atomic state distribution function, ionisation and recombination rates and radiative energy loses.

5.5 Conclusions

The program presented here allows us to calculate excited state densities, effective transfer rates and radiative energy losses as a function of electron density, and temperature. These values can be calculated for a large number of atomic
systems, as long as a suitable input file describing the system is provided. This is a relatively simple task due to the fact that the program allows the use of hydrogen approximations. These approximations are reasonable for the transitions between higher states in many systems.

In turn, the results of the collisional radiative model can be used to take local collisional and radiative effects into account in more elaborate simulations such as the PLASIMO model. This allows an accurate treatment of local effects, while limiting the number of states and species that need to be considered in the “large” simulation.
Chapter 6

Modelling an Ar-Hg fluorescent lamp plasma using a 3 electron-temperature approximation

Abstract

By using a 3 electron-group model to describe the deviation from a Maxwellian electron energy distribution, a collisional radiative model describing a low temperature Ar-Hg plasma is greatly improved. Previously, the ionisation mechanisms of such plasmas, commonly used in fluorescent lamps, could not be modelled satisfactorily. Where using a Maxwellian electron energy distribution showed the production of argon ions to be dominating over the production of mercury ions, the 3 temperature approximation yields a mercury ionisation rate which is 30 times larger than the argon ionisation rate. This work has been previously published in a slightly different form [20].

6.1 Introduction

Collisional radiative models (CRM) are used to calculate the atomic state distribution function of species in a plasma for given densities and temperatures. In addition, they can be used to determine effective production and destruction rates of particles, and radiative losses. Even though these models are 0-dimensional, they are still useful either in combination with a more complete multi-dimensional plasma model, or by themselves, to gain understanding in underlying mechanisms, and to help in the interpretation of diagnostics.

The modelling code described in [19] offers a framework for constructing collisional radiative models of atomic systems. One of the applications of the general code is given in [47] where the excitation, ionisation and radiation creation in mercury plasmas of relatively high ionisation degree are described. The original version could only handle systems with a Maxwellian electron energy distribution. In this article we will discuss extensions made to the code to handle simple cases of non-Maxwellian electron energy distributions, using a multi electron group
model, similar to the ones presented in [48–50]. This extension was motivated by the fact that non-Maxwellian electron energy distributions play an important role in low-temperature Hg-rare gas plasmas, as used in fluorescent lighting, due to a low ratio of electron-electron collisions to inelastic electron-atom collisions.

The most rigorous approach to this problem would be to solve the full Boltzmann equation in combination with the collisional radiative model, and to consider the effect of inelastic collisions on the electron energy distribution [8]. However, this type of calculation is relatively time consuming. Since a common use of collisional radiative models is to include local ionisation, recombination and radiation effects in a more general multi-dimensional plasma model, where the collisional radiative model in principle needs to be evaluated at each grid point, this limitation is not always acceptable.

To circumvent the non-linearity introduced by Coulomb electron-electron collisions, we decided to apply a 3 group model in which electrons in the tail being capable of exciting Hg or Ar are treated as separate groups.

As stated above, this is similar to the approach taken by Vriens [48]. However, our method considers particle flows between the electron groups, while Vriens focuses on energy flows. The advantage of our particle-based approach is that it gives more insight in the contributions of the various mechanisms involved, and makes refinement to a 3 group model rather straightforward. In addition, interaction with other mechanisms that affect densities, such as radiation transport, can be more easily implemented. The introduction of multiple groups allows us to take processes such as the influence of inelastic (de)excitation and external electric fields into account in a relatively simple way.

In this article, we will apply our multi electron group CRM to an Ar-Hg plasma typical of fluorescent lamps. It combines the argon model as described in [51] with the mercury model of [47]. The calculations presented here are meant as an introduction to a full 2-dimensional fluorescent lamp calculation using a general plasma fluid model [52], that will be presented in a subsequent article. In the 2-dimensional calculation, the collisional radiative model will be used to calculate the local ionisation, recombination and radiation effects using a local electron energy distribution function. Therefore, it should use an acceptable trade-off between accuracy and speed of the 0-dimensional local calculation, to allow it to be integrated into a multi-dimensional model.

6.2 Theory

6.2.1 Introduction

Collisional radiative models are usually based on the assumption that collisional and radiation effects are the dominating mechanisms in the production and destruction of excited atoms. Knowledge of the excitation energies, radiative emission probabilities and (de)excitation rates induced by electron-atom collisions are necessary to apply the model. The rate coefficient $K_{pq}$ for the electronic (de)excitation from state $p$ to state $q$ can be calculated from the electron-atom
6.2 Theory

collision cross section $\sigma_{pq}$ using the following expression:

$$K_{pq} = \int_{\epsilon_{pq}}^{\infty} \sigma_{pq}(\epsilon) f(\epsilon) v(\epsilon) d\epsilon,$$

(6.1)

where $\epsilon_{pq}$ is the energy difference between states $p$ and $q$, $v(\epsilon)$ the electron velocity, and $f(\epsilon)$ the electron energy distribution function, commonly abbreviated as EEDF.

When electron-electron collisions are sufficiently frequent, the electron energy distribution can be described by the Maxwellian distribution function $f^m(\epsilon)$:

$$f^m(\epsilon) = 2 \sqrt{\frac{m}{\pi}} (kT_e)^{-3/2} \exp \left[ -\frac{\epsilon}{kT_e} \right],$$

(6.2)

where $k$ is the Boltzmann constant, $T_e$ the electron temperature, and $f(\epsilon)d\epsilon$ the fraction of electrons with an energy between $\epsilon$ and $\epsilon + d\epsilon$.

6.2.2 Multi group EEDF

A common phenomenon in low temperature atomic plasmas of low ionisation degree is the depletion of the high-energy part of the EEDF. Collisional excitation of an atom by an electron reduces the kinetic energy of the electron, thus migrating this electron from the higher energy range to the lower energy range. When the frequency of electron-electron collisions is not large enough to restore this loss of high energetic electrons, the plasma will not be able to maintain a Maxwellian electron energy distribution.

For low temperature atomic plasmas, where the electron temperature $T_e$ is small compared to the excitation energy $\epsilon_{12}$ of the first excited level ($kT_e \ll \epsilon_{12}$), the excitation of atoms to the first excited level is only determined by a very small fraction of the electrons (typically less than 1%). It is clear that (fractions of) these so-called tail electrons also determine excitations from the ground level to higher excited states. Rates of excitations starting from excited levels are usually smaller due to the lower density of excited levels, and, when the excitation energy is small, these processes are mainly performed by electrons with low energies. This means that the electrons with energies below $\epsilon_{12}$ are hardly affected. The stepwise ionisation flow through the excitation space is mainly determined by the few tail electrons, since these trigger the stepwise process by means of the $1 \rightarrow 2$ transition.

Because of the significance of the tail electrons, we divide the electron energy distribution into two parts: a bulk part with electron energies below $\epsilon_{12}$ and a tail part for energies above $\epsilon_{12}$. For simplicity we will initially follow the two electron group model [49], and assume that both parts essentially have a Maxwellian distribution, but with two distinct temperatures (see figure 6.1 on the next page). For the bulk this seems reasonable, since it was determined that it contains the majority of the electrons, and that those electrons are not significantly affected by the inelastic processes. For the tail, this assumption is less intuitive, but other calculations [8] suggest that the tail can be acceptably approximated by a Maxwell function.
The energy distribution function for the electrons is now given by

$$f(\epsilon) = 2\sqrt{\frac{\epsilon}{\pi}} \times \begin{cases} (kT_e)^{-3/2} \exp\left[-\frac{\epsilon}{kT_e}\right] & \epsilon < \epsilon_{12}, \\ S \times (kT_t)^{-3/2} \exp\left[-\frac{\epsilon}{kT_t}\right] & \epsilon > \epsilon_{12}, \end{cases}$$

(6.3)

where $T_e$ is the electron bulk temperature, $T_t$ the tail temperature, and $S$ is a factor used to make $f(\epsilon)$ a continuous function:

$$S = \left(\frac{T_e}{T_t}\right)^{-3/2} \exp\left[-\epsilon_{12} \left(\frac{1}{kT_e} - \frac{1}{kT_t}\right)\right].$$

(6.4)

Note that this expression for the electron energy distribution function violates the normalisation condition, since $\int_0^\infty f(\epsilon)d\epsilon \neq 1$. In addition, the derivative of $f(\epsilon)$ is not continuous, violating the continuity of the electron flux in energy space. However, due to the small number of electrons in the tail, these deviations are negligible.

We will now treat the collection of tail electrons as one group, determine the population of this group, and investigate how this population is affected by the various production and destruction processes.

The population of the electron energy tail is given by

$$F = \int_{\epsilon_{12}}^\infty f(\epsilon)d\epsilon.$$  

(6.5)

Omitting non-local effects, we now consider the steady-state population balance for these electrons,

$$F \cdot (D_c + D_{el} + D_{in}) = P_c + P_E,$$

(6.6)

showing that the production of tail electrons due to Coulomb heating ($P_c$) and field acceleration ($P_E$) is balanced by the destruction by Coulomb cooling ($D_c$) and elastic ($D_{el}$) and inelastic ($D_{in}$) energy transfer to the heavy particles.
In case of a Maxwell equilibrium, where the Coulomb processes are dominant, and the other processes may be considered as mere perturbations [42], this balance can be reduced to

\[ F_{eq} \cdot D_c = P_c, \]  

(6.7)

where \( F_{eq} \) can be obtained by using \( f = f^m \) in equation (6.5).

Combining eqs. (6.6) and (6.7) gives an expression for the departure from equilibrium:

\[ y_t = \frac{F}{F_{eq}} = \frac{1 + P_E/(F_{eq} D_c)}{1 + D_{el}/D_c + D_{in}/D_c}. \]  

(6.8)

6.2.3 Processes

General

In the treatment of the destructive processes we can distinguish between stepwise processes, in which crossing the transition between tail and bulk is done in a relatively smooth energy changing manner, and jump processes in which this transition happens suddenly from any position (energy) in the tail towards the bulk.

For the step processes we may define a flux at the intersection energy \( \epsilon \):

\[ \Phi_{proc.}(\epsilon) = f(\epsilon) \left( \frac{\partial \epsilon}{\partial t} \right)_{proc.}, \]  

(6.9)

where \( (\partial \epsilon/\partial t)_{proc} \) represents the change in energy caused by a certain process. We can now define the corresponding destruction frequency as:

\[ \nu_{proc.} = \left( \frac{\partial \epsilon}{\partial t} \right)_{proc.} \epsilon^{-1}, \]  

(6.10)

so that

\[ \Phi_{proc.}(\epsilon) = f(\epsilon) \epsilon \nu_{proc.}. \]  

(6.11)

In order to obtain expressions for the tail population and the various destruction factors and production terms, we will employ some useful algebraic manipulations, based on the fact that in our case \( \exp \left[ -\frac{\epsilon}{kT} \right] \) is strongly decreasing as function of \( \epsilon \).

If we consider the integral

\[ \int_{\epsilon_{12}}^{\infty} a(\epsilon) \exp \left[ -\frac{\epsilon}{kT} \right] d\epsilon \]

(6.12)

\[ = -kT a(\epsilon) \exp \left[ -\frac{\epsilon}{kT} \right]_{\epsilon_{12}}^{\infty} + kT \int_{\epsilon_{12}}^{\infty} \frac{\partial a(\epsilon)}{\partial \epsilon} \exp \left[ -\frac{\epsilon}{kT} \right] d\epsilon, \]

and assume that the exponential factor dominates the integrated expression, we can reduce it to

\[ \int_{\epsilon_{12}}^{\infty} a(\epsilon) \exp \left[ -\frac{\epsilon}{kT} \right] d\epsilon = kT \exp \left[ -\frac{\epsilon_{12}}{kT} \right] \left( a(\epsilon_{12}) + kT \frac{\partial a(\epsilon)}{\partial \epsilon} + \ldots \right), \]  

(6.13)
since $\exp\left[-\frac{\epsilon}{kT}\right] \to 0$ for $\epsilon \to \infty$.

Cross sections of step wise processes, i.e. without a threshold energy, can be described by

$$a(\epsilon) = c \epsilon^\alpha.$$  \hspace{1cm} (6.14)

If $\alpha kT \ll \epsilon_{12}$, it follows that

$$\int_{\epsilon_{12}}^{\infty} a(\epsilon) \exp\left[-\frac{\epsilon}{kT}\right] d\epsilon = kT a(\epsilon_{12}) \exp\left[-\frac{\epsilon_{12}}{kT}\right] \left(1 + \alpha \frac{kT}{\epsilon_{12}} + \ldots\right)$$

$$\approx kT a(\epsilon_{12}) \exp\left[-\frac{\epsilon_{12}}{kT}\right].$$  \hspace{1cm} (6.15)

Cross sections of jump processes, with threshold energy $\epsilon_{12}$ may be described by

$$a(\epsilon) = c \left(\frac{\epsilon}{\epsilon_{12}}\right)^\alpha - 1 \epsilon^\beta,$$  \hspace{1cm} (6.16)

in which case we find ($a(\epsilon_{12}) = 0$) that:

$$\int_{\epsilon_{12}}^{\infty} a(\epsilon) \exp\left[-\frac{\epsilon}{kT}\right] d\epsilon$$

$$= (kT)^2 \frac{\partial a(\epsilon)}{\partial \epsilon} \bigg|_{\epsilon=\epsilon_{12}} \left(1 + (\alpha - 1 + 2\beta) \frac{kT}{\epsilon_{12}} + \ldots\right)$$

$$\approx (kT)^2 \frac{\partial a(\epsilon)}{\partial \epsilon} \bigg|_{\epsilon=\epsilon_{12}},$$  \hspace{1cm} (6.17)

provided that $(\alpha - 1 + 2\beta) \frac{kT}{\epsilon_{12}} \ll 1$. In the following sections, the approximations of eqs. (6.15) and (6.17) will be used to derive expressions for the aforementioned destruction factors and production terms.

Applying eq. (6.15) to the population of the tail given by eq. (6.5) shows this population of electrons with an energy above $\epsilon_{12}$ can be approximated by

$$\int_{\epsilon_{12}}^{\infty} f(\epsilon) d\epsilon \approx f(\epsilon_{12}) kT_t.$$  \hspace{1cm} (6.18)

**Field production**

An expression for the contribution due to the electric field can be found by realising that it creates a diffusion and a corresponding flux in velocity space [53]:

$$\Gamma_v(\epsilon) = -D_E \frac{df(v)}{dv}, \quad \text{with} \quad D_E = \frac{e^2 E^2}{3m_e^2 \nu_m},$$  \hspace{1cm} (6.19)

where $f(v)$ is the velocity distribution function, $e$ and $m_e$ are the electron charge and mass, $E$ is the electric field strength, and $\nu_m = N \sigma_e v$ is the momentum transfer frequency, with $\sigma_e$ the elastic collision cross section. By using

$$\frac{df(v)}{dv} = \frac{df(v)}{d\epsilon} \frac{d\epsilon}{dv} = -\frac{f(v)}{kT} m_e v,$$  \hspace{1cm} (6.20)
and transforming the flux to energy space through

\[ \Phi_E(\epsilon) = 4\pi v^2 \Gamma_v(E), \quad f(\epsilon)d\epsilon = 4\pi v^2 f(v)dv, \quad (6.21) \]

we obtain

\[ \Phi_E(\epsilon) = \frac{e^2 E^2}{3N\sigma_{el}} \frac{f(\epsilon)}{kT} \sqrt{\frac{2\epsilon}{m_e}}. \quad (6.22) \]

The flux due to the electric field at the threshold \( \epsilon_{12} \), which effectively the production of tail electrons, is now given by

\[ P_E = \Phi_E(\epsilon_{12}) = f(\epsilon_{12}) \frac{e^2 E^2}{3N\sigma_{el}} \frac{v_{12}}{kT}. \quad (6.23) \]

### Coulomb decay

Due to collisions with bulk electrons, the tail electrons will decay to low energy values with a decay frequency

\[ \nu_c = 2n_e \pi \hat{e}^4 \sqrt{\frac{2\epsilon}{m_e}} \epsilon^{-2} \ln \Lambda, \quad (6.24) \]

where \( \ln \Lambda \) is the Coulomb logarithm (approximately 10 for the type of plasma under study), and \( \hat{e} = e/\sqrt{4\pi\epsilon_0} \). Since the decay rate of the tail is equal to the flux, given by eq. (6.11), divided by the tail contents, given by eq. (6.18), we get

\[ D = \frac{\Phi(\epsilon_{12})}{f(\epsilon_{12})kT} = \frac{\epsilon_{12}}{kT_t} \nu(\epsilon_{12}), \quad (6.25) \]

so that the following expression for the Coulomb destruction rate is obtained:

\[ D_c = \frac{\epsilon_{12}}{kT_t} 2n_e \pi \hat{e}^4 \sqrt{\frac{2\epsilon_{12}}{m_e}} \epsilon_{12}^{-2} \ln \Lambda. \quad (6.26) \]

### Elastic decay rate

The decay frequency for elastic energy transfer from the electrons to the heavy particles is given by

\[ \nu_{el} = 2\frac{m_e}{M} N\sigma_{el} \sqrt{\frac{2\epsilon_{12}}{m_e}}, \quad (6.27) \]

where \( N \) is the atom density and \( M \) the heavy particle mass. Using eq. (6.25), this leads to the follow elastic decay rate:

\[ D_{el} = 2\frac{E_{12} m_e}{kT_t M} N\sigma_{el} \sqrt{\frac{2\epsilon_{12}}{m_e}}. \quad (6.28) \]
Inelastic decay rate

When a tail electron excites an atom, the electron will normally be transferred to the low energy part of the bulk. For the inelastic decay rate of the jump-wise process, we may write

\[ D_{\text{in}} = \langle \nu_{\text{in}} \rangle = N \langle \sigma_{\text{in}} v \rangle _t, \quad (6.29) \]

where \( \sigma_{\text{in}} \) is the inelastic collision cross section, and \( \langle x \rangle _t \) means \( x \) averaged over the tail distribution. Note that averaging is necessary since tail electrons may leave the tail from any position. They will not gradually pass the threshold, as do step processes.

In the binary encounter approximation, where collisions between a bound electron and a free electron are treated as collisions between two free electrons, the following relation holds for the differential cross section:

\[ \frac{d\sigma}{d\Delta \epsilon} = \frac{\pi \delta^4}{\epsilon} \frac{1}{\Delta \epsilon^2}. \quad (6.30) \]

From this, we find an expression for \( \sigma(\epsilon) \) through integration:

\[ \sigma(\epsilon) - \sigma(\epsilon_{12}) = \int_{\epsilon_{12}}^{\epsilon} \frac{\pi \delta^4}{\epsilon} \frac{1}{\Delta \epsilon^2} = -\frac{\pi \delta^4}{\epsilon} \frac{1}{\Delta \epsilon} \bigg|_{\epsilon_{12}}^{\epsilon}. \quad (6.31) \]

If \( \sigma(\epsilon_{12}) = 0 \), this equation leads to

\[ \sigma(\epsilon) = c_A \frac{\pi \delta^4}{\epsilon^2} \left( \frac{\epsilon}{\epsilon_{12}} - 1 \right). \quad (6.32) \]

Here the factor \( c_A \) is introduced to account for deviations from the binary encounter approximation. Using eq. (6.17), it is found that

\[ \langle \sigma v \rangle _t = \left\langle \sigma \sqrt{\frac{2\epsilon}{m_e}} \right\rangle _t = c_A \frac{\pi \delta^4}{\epsilon_{12}} \sqrt{\frac{kT_e}{m_e}} \sqrt{\frac{2\epsilon_{12}}{m_e}}. \quad (6.33) \]

### 6.2.4 Tail temperature

In order to derive an expression for the tail temperature, we reconsider eq. (6.8):

\[ y_t = \frac{1 + P_E/(F_{\text{eq}} D_e)}{1 + D_{\text{el}}/D_e + D_{\text{in}}/D_e} \quad (6.8) \]

Using eqs. (6.18,6.23, 6.26), we see that

\[ \frac{P_E}{F_{\text{eq}} D_e} = \frac{e^2 E^2}{6 N n_e \pi \delta^4 \sigma_{\text{el}} \ln A kT_e} \frac{\epsilon_{12}}{m_e}. \quad (6.34) \]

Since this is much less than unity for a typical fluorescent lamp, we will neglect this term in the numerator.

If we consider the elastic to inelastic decay rate in the denominator of eq. (6.8), we obtain

\[ \frac{D_{\text{el}}}{D_{\text{in}}} = \frac{2 m_e \sigma_{\text{el}} N_{\text{el}}}{M \langle \sigma_{\text{in}} \rangle N_{\text{in}}}. \quad (6.35) \]
6.2 Theory

The ratio \( \frac{N_{el}}{N_{in}} \) is the ratio of particles that are available for elastic and inelastic collisions. For a mixture of a buffer gas and a seed gas, this ratio typically is \( 10^2 \). Since the ratio of the cross sections is usually not larger than 10, and \( 2m_e/M \) is at most \( 10^{-3} \), we can assume \( D_{el}/D_{in} \ll 1 \), so that we may neglect \( D_{el}/D_{in} \) compared to \( D_{in}/D_{c} \).

The ratio of inelastic and coulomb processes, as found in the denominator of eq. (6.8), is given by

\[
\frac{D_{in}}{D_{c}} = \frac{c_A}{2 \ln \Lambda} \frac{N}{n_e} \left( \frac{kT_e}{\epsilon_{12}} \right)^2
\]

(6.36)

It is interesting to note that this equation gives the ratio of a free electron colliding with a bound electron in the binary encounter approximation (density \( N \)) to the collision of a free electron with another free electron (density \( n_e \)).

The latter benefit from the cumulative effect of multiple small collisions, as is expressed by the factor of \( 2 \ln \).

Since eq. (6.36) is the only significant term in eq. (6.8), we find the following expression for the temperature \( T_t \) of the tail:

\[
y_t = \frac{F}{F_{eq}} = \frac{f(\epsilon_{12})kT_e}{f(\epsilon_{12})kT_i} \approx \frac{T_i}{T_e} \approx \frac{1}{1 + D_{in}/D_{c}} = \left( 1 + \frac{c_A}{2 \ln \Lambda} \frac{N}{n_e} \left( \frac{kT_e}{\epsilon_{12}} \right)^2 \right)^{-1}
\]

\[
= \left( 1 + \frac{c_A}{2 \ln \Lambda} \frac{N}{n_e} \left( \frac{kT_e}{\epsilon_{12}} \right)^2 \left( \frac{T_i}{T_e} \right)^2 \right)^{-1}.
\]

(6.37)

This expression relates the tail temperature to the bulk temperature and the ionisation factor \( n_e/N \).

At this point it should be noted that the above expression could easily be modified to take the influence of an \( E \)-field or de-excitation of atoms on the tail temperature into account. This would lead to different values for the \( T_t \), which can then be used to retrieve (de)excitation rates from look-up tables generated by the CRM.

A further enhancement to the scheme described here is the extension to a 3 electron group model. For an Ar-Hg mixture, the EEDF shows a clear bend at 4.67 eV, due to Hg excitation from the ground state, and another bend at 10.55 eV, caused by Ar excitation from the ground state. In this case, the EEDF can be described by (see figure 6.2 on the following page).

\[
f(\epsilon) = 2 \sqrt{\frac{2}{\pi}} \times \left\{ \begin{array}{ll}
(\epsilon_{12})^{-3/2} \exp \left[ -\frac{\epsilon}{\epsilon_{12}} \right] & \epsilon < \epsilon_{12,Hg} \\
S_1 \times (\epsilon_{12,Hg})^{-3/2} \exp \left[ -\frac{\epsilon}{\epsilon_{12,Hg}} \right] & \epsilon_{12,Hg} \leq \epsilon < \epsilon_{12,Ar} \\
S_2 \times (\epsilon_{12,Ar})^{-3/2} \exp \left[ -\frac{\epsilon}{\epsilon_{12,Ar}} \right] & \epsilon_{12,Ar} \leq \epsilon.
\end{array} \right.
\]

(6.38)

Here \( T_{l,Hg} \) and \( T_{l,Ar} \) are the temperatures of the first and second tail region, respectively, \( \epsilon_{12,Hg} = 4.67 \text{ eV} \), \( \epsilon_{12,Ar} = 10.55 \text{ eV} \), \( S_1 \) is the same as \( S \) of eq. (6.4), and \( S_2 \) is equal to

\[
S_2 = S_1 \left( \frac{T_{l,Hg}}{T_{l,Ar}} \right)^{-3/2} \exp \left[ -\epsilon_{12,Ar} \left( \frac{1}{kT_{l,Hg}} - \frac{1}{kT_{l,Ar}} \right) \right].
\]

(6.39)
Chapter 6: Modelling an Ar-Hg fluorescent lamp plasma using a 3 electron-...

Figure 6.2: An electron energy distribution divided into three groups, each with its own temperature. The locations of the groups are determined by the energies of the first excitations of mercury and argon.

\[ T_{t,\text{Ar}} \]

\[ T_{t,\text{Hg}} \]

\[ \epsilon_{12,\text{Hg}} \]

\[ \epsilon_{12,\text{Ar}} \]

Table 6.1: Comparison between tail temperatures presented in [54] for both a 2 electron group model \((T_{tG})\) and a Boltzmann calculation \((T_{tB})\) with results of our 2 electron group calculation. The calculation is for an Ar-discharge with \(n_{Ar} = 1.65 \times 10^{22} \text{ m}^{-3}\).

<table>
<thead>
<tr>
<th>(I) (mA)</th>
<th>25</th>
<th>50</th>
<th>100</th>
<th>200</th>
<th>400</th>
</tr>
</thead>
<tbody>
<tr>
<td>(T_e) (K)</td>
<td>25000</td>
<td>22000</td>
<td>18000</td>
<td>16000</td>
<td>13000</td>
</tr>
<tr>
<td>(n_e) ((10^{17} \text{ m}^{-3}))</td>
<td>2.7</td>
<td>6.0</td>
<td>12</td>
<td>22</td>
<td>46</td>
</tr>
<tr>
<td>(T_{tG}) (K) ([54])</td>
<td>4070</td>
<td>5160</td>
<td>6240</td>
<td>7050</td>
<td>8230</td>
</tr>
<tr>
<td>(T_{tB}) (K) ([54])</td>
<td>4860</td>
<td>5830</td>
<td>6790</td>
<td>7240</td>
<td>8260</td>
</tr>
<tr>
<td>(T_t) (K) (\text{our calc.})</td>
<td>5100</td>
<td>6160</td>
<td>6890</td>
<td>7660</td>
<td>8170</td>
</tr>
</tbody>
</table>

6.3 Results

6.3.1 Comparison to other calculations

To gain some insight in the validity of our approximation, we compare the temperatures presented by Morgan and Vriens [54] for an Ar discharge described with both a 2 electron group model and a full Boltzmann equation, to the results of our calculation. We used eq. (6.37) to calculate \(T_t\) for the values of \(n_e\), \(T_e\) and \(N = n_{Ar}\) as used in [54]. The factor \(c_A = c_A(Ar) = 0.32\) is obtained by fitting eq. (6.32) to the first excitation cross section of argon, and \(\epsilon_{12} = \epsilon_{12,Ar} = 10.55 \text{ eV}\) is the energy of this excitation. As can be seen in table 6.1, the agreement is quite satisfactory.

Table 6.2 on the facing page shows a comparison of tail temperatures calculated using our method with temperatures derived from EEDF’s obtained by solving the full Boltzmann equation as done by Winkler et al. [8]. Here, we use the three group approximation of eq. (6.38), where \(T_{t,Hg}\) is calculated through eq. (6.37), with \(N = n_{Hg}\). Since the first four energy levels of mercury have similar
Table 6.2: Comparison between tail temperatures estimated from [8] and our calculation. For this calculation, $n_{Ar} = 9.9 \times 10^{22}$ m$^{-3}$ and $n_{Hg} = 2.3 \times 10^{20}$ m$^{-3}$.

<table>
<thead>
<tr>
<th>$I$ (mA)</th>
<th>100</th>
<th>400</th>
<th>800</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_e$ (K) ([8])</td>
<td>13200</td>
<td>11400</td>
<td>11000</td>
</tr>
<tr>
<td>$n_e$ (10$^{17}$ m$^{-3}$) ([8])</td>
<td>1</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>$T_{t,Hg}$ (K) ([8])</td>
<td>6800</td>
<td>8100</td>
<td>8800</td>
</tr>
<tr>
<td>$T_{t,Hg}$ (K) (our calc.)</td>
<td>6300</td>
<td>7500</td>
<td>8500</td>
</tr>
<tr>
<td>$T_{t,Ar}$ (K) ([8])</td>
<td>3000</td>
<td>4800</td>
<td>4400</td>
</tr>
<tr>
<td>$T_{t,Ar}$ (K) (our calc.)</td>
<td>2300</td>
<td>3000</td>
<td>4000</td>
</tr>
</tbody>
</table>

Figure 6.3: The argon (thin lines) and mercury (thick lines) ion production assuming a Maxwellian EEDF, calculated for three different electron densities: $10^{17}$ m$^{-3}$ (solid line), $5 \times 10^{17}$ m$^{-3}$ (short dashed line), and $10^{18}$ m$^{-3}$ (long dashed line).

excitation energies (4.67, 4.90, 5.46 and 6.7 eV, respectively), using a cross section that is a summation of the first four excitation cross sections yields a more realistic result. We found that this combined cross section can be described satisfactorily by eq. (6.32) with $c_A = c_A(Hg) = 0.79$, and $\epsilon_{12,Hg} = 4.67$ eV. $T_{t,Ar}$ is calculated as in the previous comparison. Again, the agreement for $T_{t,Hg}$ is quite good. Our calculations give values for $T_{t,Ar}$ which are lower than those of Winkler.

### 6.3.2 Results for an Ar-Hg discharge

Figure 6.3 shows the effective ion production $n_1 n_e J_{1+}$ [47] of both argon and mercury, calculated under the assumption of a Maxwellian electron energy distribution function ($T_i = T_e$). As in all other results, an argon density $n_{Ar} = 9 \times 10^{22}$ m$^{-3}$ and a mercury density $n_{Hg} = 2 \times 10^{20}$ m$^{-3}$, corresponding to an argon filling pressure of 360 Pa and a mercury vapour pressure of 0.8 Pa, both at 293 K, were used.
Chapter 6: Modelling an Ar-Hg fluorescent lamp plasma using a 3 electron-...
Figure 6.5: The effective ion production for the two electron group model. Lines as in figure 6.3 on page 73.

Figure 6.6: The total ion production for argon and mercury as calculated by the three electron group model. Lines as in figure 6.3 on page 73.
its ion production remains more or less unchanged.

6.4 Conclusion

Under fluorescent lamp conditions, the 3 temperature model predicts a mercury ion production that is about 30 times higher than the argon ion production. This is a significant improvement over the case of a Maxwellian EEDF, where argon was the dominating ion. Using the results of the 3 temperature model in a multi-dimensional plasma fluid model should improve the realism, while limiting the amount of extra calculation effort necessary.

Due to the lack of experimental information on ionisation rates in Ar-Hg mixtures, we cannot directly compare our results to experiments. But by using the results of our approximation in a full plasma model [52] that also considers important non-local effects, such as transport, and by comparing the densities calculated in this manner with experimental results, we might have an indirect method for confirmation.

The approximation presented in this paper, allows a multitude of effects that influence the tail of the EEDF to be taken into account. For typical fluorescent lamp conditions, it predicts EEDF temperatures that are comparable to those of previous multi electron-group models and Boltzmann calculations. By using the (few) tail temperatures as extra indices into tables of effective (de)excitation rates calculated by a CRM, the extra calculations necessary to consider the influence of these effects on those rates is relatively small, while the realism of the transport model should greatly improve.

Acknowledgement

This research is supported by the Technology Foundation STW, Applied Science Division of NWO. The authors wish to acknowledge the helpful comments of W.J. Goedheer.
Chapter 7

Implementation of a friction-based self-consistent multi-component diffusion model

Abstract

In order to improve the realism of our PLASIMO computer model [1, 52] in modelling multi-ion mixtures, we developed and implemented a self-consistent multi-component diffusion model based on frictions. As presented, this diffusion model includes effects of the ambipolar electric field as well as any external electric fields. Moreover, it is comparatively easy to include other diffusion contributions. This model was shown to produce good and consistent results for both single and multi-ion mixtures. This work has been previously published in a slightly different form [21].

7.1 Introduction

A very important process in the description of multi-component flows and plasmas is diffusion. With this, we mean the movement of individual species relative to a bulk flow. A naive model of this is the Fick model which states that the particle flux density $\Gamma_i$ of species $i$ is driven by a gradient in the number density $n_i$ of the species:

$$\Gamma_i = -D_i \vec{\nabla} n_i. \quad (7.1)$$

The $D_i$ in this equation is called the diffusion coefficient. While this description is strictly only valid in the case of the diffusion of a single type of particle (i.e. species) through an object or flow with constant density and temperature, it can be applied with reasonable accuracy to describe diffusion of multiple minority species through a dominating “background” gas, providing the temperature gradients are small.

In the case of diffusion of a mixture of electrons and ions, it is common to adapt equation (7.1) to account for ambipolar diffusion. In this case the lighter and faster electrons enhance the diffusion of the heavier and slower ions through
the Coulomb force. It can be shown that this results in an electron flux \( \Gamma_e \) and an ion flux \( \Gamma_+ \) according to the following relation \([53]\):

\[
\Gamma_e = \Gamma_+ = -D_+ \left( 1 + \frac{T_e}{T_+} \right) \nabla n_+ ,
\]

(7.2)

where \( T_e \) is the temperature of the electrons and \( T_+ \) the temperature of the ions. Note that this equation is only valid in the case of a plasma with only one singly charged ion species.

In order to describe more complex mixtures, with multiple ions, without a dominating background species, or with significant temperature gradients, a more complete treatment of diffusion is necessary. Examples of this are the multicomponent diffusion models presented in \([56]\) and \([57]\), or the diffusion model used in the Phoenics model \([30]\), that use mass fraction or concentration based descriptions.

This article is devoted to the theoretical background and the implementation of the self-consistent diffusion (SCD) model in the new C++ based PLASIMO code for modelling low-temperature plasmas \([1, 15]\). We stay as close as possible to the Boltzmann Transport Equations (BTE), which implies that our description is more based on densities rather than concentrations. It was felt that this description is more transparent than pre-existing ones and, in our case, allowed a more robust implementation.

In this paper we will use the term diffusion for any kind of transport of species that leads to a mean velocity with respect to the bulk flow. This is in contrast with the field of gas discharge physics where a distinction is made between drift and diffusion. Our nomenclature is closer to that of the field of fluid dynamics where e.g. the drift of electrons due to an electric field is called forced diffusion instead of drift. Thus in addition to transport driven by gradients in the density, we will also denote transport due to temperature gradients and electric fields as diffusion.

To test the code we present three case studies for plasmas without any bulk flow. It was found that in a low-pressure case of a simple ion-electron-atom mixture the SCD treatment leads to electron densities which can be 20% higher than in the case of a Fick-like ambipolar diffusion treatment and that this even increases towards 30% for a high pressure case. This is remarkable since the high pressure plasma was found to be close to equilibrium so that diffusion is not expected to play an important role; volume recombination might be important as well. In a mixture of two ions (Hg+ and Ar+) we found that the SCD model gives a electron density which is a factor of 3 larger than what the Fick model predicts. In general we may conclude that SCD gives a better plasma confinement and thus larger values of the electron densities.
7.2 Theory

7.2.1 Introduction

In order to accurately describe the transport processes occurring in a multi-component plasma, we need a systematic treatment of the driving forces which are responsible for the inequality between the systematic velocities \( \bar{u}_i \) of the various species and the velocity \( \bar{u} \) of the plasma as a whole, the so-called barycentric or bulk velocity.

In this study, this treatment will be based on the first two moments of the Boltzmann transport equations yielding for each species the specific particle balance and specific momentum balance (force balance).

The first is the specific particle mass balance:

\[
\frac{\partial \rho_i}{\partial t} + \nabla \cdot (\rho_i \bar{u}_i) = m_i S_i, \tag{7.3}
\]

where \( S_i \), the local particle source term for species \( i \), gives the production of particle \( i \) per unit of time and volume, \( \rho_i \equiv n_i m_i \) is the partial mass density, \( m_i \) the mass, and \( \bar{u}_i \) is the average velocity of species \( i \).

The second equation is the specific momentum balance:

\[
\frac{\partial (\rho_i \bar{u}_i)}{\partial t} + \nabla \cdot (\rho_i \bar{u}_i \bar{u}_i) = -\nabla \cdot \overline{\mathbf{F}_i} + \bar{u}_i m_i S_i + \bar{F}_i + \bar{R}_t^T + \bar{R}_f^F, \tag{7.4}
\]

where the pressure tensor \( \overline{\mathbf{F}_i} \) in \( \nabla \cdot \overline{\mathbf{F}_i} = \nabla p_i - \nabla \cdot \mathbf{\Pi}_i \) (\( p_i \) is the partial pressure and \( \mathbf{\Pi}_i \) the viscosity tensor), is developed with respect to the system in which species \( i \) is at rest. \( \bar{F}_i \), the body force per unit volume acting on species \( i \), contains, among others, contributions from the gravitational force and the electric fields, \( \bar{R}_t^T \) is the thermophoretic force per unit volume [58], and \( \bar{R}_f^F \) is the force caused by friction with other diffusing species.

The friction force \( \bar{R}_f^F \) is given by

\[
\bar{R}_f^F = \sum_j f_{ij} (\bar{u}_j - \bar{u}_i), \tag{7.5}
\]

and depends on the velocity differences. Here we introduced the volumetric friction force \( f_{ij} \) per unit of velocity difference:

\[
f_{ij} = n_i n_j m_{ij} \Omega_{ij}, \quad f_{ii} \equiv 0, \tag{7.6}
\]

where \( \Omega_{ij} \) is the rate coefficient for momentum transfer from species \( i \) to \( j \) and \( m_{ij} \) the reduced mass of the \( i, j \)-system. Note that the friction is related to the binary diffusion coefficient \( D_{ij} \) through

\[
D_{ij} = \frac{p_i p_j}{p f_{ij}}, \tag{7.7}
\]

where \( p_i \) is the partial pressure of species \( i \) and \( p \) is the total pressure.
The thermophoretic force \( \vec{R}_i^T \) is described by \cite{56}

\[
\vec{R}_i^T = \sum_j \left( \beta_{ij} \frac{\nabla T_j}{T_j} - \beta_{ji} \frac{\nabla T_i}{T_i} \right),
\tag{7.8}
\]

where \( T_i \) and \( T_j \) refer to the temperatures of species \( i \) and \( j \), while \( \beta_{ij} \) is the thermal diffusion factor. Note that typically \( \beta_{ij} \neq \beta_{ji} \), and that, to simplify notation, we define \( \beta_{ii} \equiv 0 \). For neutral-neutral and charged-neutral interactions, we use the approximation given by \cite{59} and \cite{60}, respectively. We neglect \( \beta_{ij} \) in case of charged-neutral interactions, since it is much smaller than that of a charged-charged interaction.

The transport of mass, or bulk transport, is described by two equations obtained by summing equations (7.3) and (7.4) over all the particles. Summing equation (7.3) yields the mass continuity equation

\[
\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{u}) = 0,
\tag{7.9}
\]

where \( \rho \equiv \sum_i \rho_i \) is the mass density and

\[
\vec{u} \equiv \sum_i \frac{\rho_i}{\rho} \vec{u}_i,
\tag{7.10}
\]

the barycentric or bulk velocity. Note that \( \sum_i m_i S_i = 0 \), due to the fact that production of one species implies the destruction of others (mass conservation).

Summing equation (7.4) over all the particles gives us the well-known Navier-Stokes equation

\[
\frac{\partial \rho \vec{u}}{\partial t} + \nabla \cdot (\rho \vec{u} \vec{u}) = -\nabla \cdot \vec{P} + \sum_i \rho_i \vec{F}_i,
\tag{7.11}
\]

a derivation which is less simple. Here we will sketch the derivation and refer to \cite{61} for more detail.

Firstly, in adding the time derivatives we may use the definition of the barycentric velocity as given by equation (7.10). Secondly, the summation of the momentum production gives \( \sum_i u_i m_i S_i = 0 \) since in each fragmentation or association process momentum is conserved. Thirdly, the \( \vec{R}_i^X \) terms disappear (\( \sum_i \vec{R}_i^X = 0 \)) because of Newton’s third law of motion (action=reaction).

Finally, we will deal with the two divergence terms by combining them to \( \nabla \cdot (\rho \vec{u} \vec{u} + \vec{P}_i) \). This is justified since both originate from the second term of the Boltzmann transport equation \( \nabla \vec{F}_i \vec{v}_i \), where \( F_i \) is the probability distribution function. Multiplication of this BTE term with \( m_i \vec{v}_i \) and integration over velocity space gives \( \nabla \cdot \rho_i \langle \vec{v}_i \vec{v}_i \rangle \), where \( \langle \rangle \) stands for averaging over velocity space. For species \( i \) this can be split in a systematic and a stochastic part

\[
\rho_i \langle \vec{v}_i \vec{v}_i \rangle = \rho_i \vec{u}_i \vec{u}_i + \rho_i \langle \vec{c}_i \vec{c}_i \rangle.
\tag{7.12}
\]

with \( \vec{c}_i \) and \( \vec{P}_i = \rho_i \langle \vec{c}_i \vec{c}_i \rangle \) the stochastic velocity and pressure in system \( i \), respectively. If we now introduce the diffusion velocity

\[
\vec{u}_i = \vec{u} + \vec{v}_i,
\tag{7.13}
\]
we find that the summation of the systematic terms over all the species gives
\[ \sum_i \rho_i \bar{u}_i \bar{u}_i = \rho \bar{u} \bar{u} + \sum_i \rho_i \bar{v}_i \bar{v}_i + 2 \sum_i \rho_i \bar{v}_i \bar{u}_i, \] (7.14)
and since only the bulk velocity \( \bar{u} \) causes a net mass flux, i.e.
\[ \sum_i \rho_i \bar{u}_i = 0, \] (7.15)
the last summation of equation (7.14) disappears.

The second term can be combined with the stochastic term since \[61\]
\[ \overline{\mathbf{p}} = \sum_i p_i + \sum_i \rho_i \bar{v}_i \bar{v}_i, \] (7.16)
meaning that the pressure of an ensemble \( i \) with velocities \( \bar{v}_i \) with respect to a certain system will exert in that system a pressure \( \rho_i (\bar{v}_i \bar{v}_i) + \rho_i \bar{u}_i \bar{u}_i \). This completes our sketch which proves that the summation of the specific momentum equations over the species gives the momentum equation of the bulk.

In general we will proceed in the following manner. First, the balance equations for mass (7.9) and momentum (7.11) are solved in conjunction with the boundary conditions of the bulk velocity. After that, the diffusion velocities of the various species are solved by multiplying equation (7.3) by \( \bar{u}_i \) and subtracting the result from equation (7.4). By expanding some terms, we obtain:
\[ \rho_i \frac{\partial \bar{u}_i}{\partial t} + \rho_i \bar{u}_i \mathbf{\bar{v}} \cdot \mathbf{\bar{u}}_i = -\mathbf{\bar{V}} p_i + \mathbf{\bar{V}} \cdot \overline{\mathbf{p}} + \overline{\mathbf{F}}_i + \mathbf{\bar{R}}_i + \mathbf{\bar{R}}_i^T \] (7.17)

Finally, we will cast this equation in a form where the friction term is the leading term. This is the main subject of this paper, and will be treated in the next section.

### 7.2.2 Self-consistent diffusion model

To describe the case of diffusive transport, we will now reconsider equation (7.17) to show that a friction force acting on particles of type \( i \) caused by the difference in velocity with respect to the other particles is balanced by all other forces acting on this species
\[ -\mathbf{\bar{R}}_i^F = -\sum_j f_{ij} (\bar{u}_j - \bar{u}_i) = \sum_j f_{ij} (\bar{v}_i - \bar{v}_j) = \bar{G}_i, \] (7.18)
where \( \bar{G}_i \) contains all the terms from equation (7.17) that are not explicitly shown.

From (7.18), we see that \( \bar{v}_i \) is given by
\[ f_i^{\text{eff}} \bar{v}_i = \bar{G}_i + \sum_j f_{ij} \bar{v}_j, \] (7.19)
where \( f_i^{eff} \equiv \sum_j f_{ij} \). From this equation it is clear that the velocity of one species depends on the velocities of the others. For this reason we will have to solve a coupled set of equations for all species, of the following form:

\[
F \cdot \vec{V} = \vec{G} \tag{7.20}
\]

As will be shown further on, matrix \( F \) contains the \( f_{ij} \) and \( f_i^{eff} \) terms, \( \vec{V} \) is a vector containing all components of all diffusive velocities \( \vec{v}_i \), and \( \vec{G} \) is source term, whose main contributions are \( \nabla p_i \), \( \nabla T_i \), and the electric field in the case of charged particles.

At this point we only consider the steady state solution, omitting the \( \rho_i \frac{\partial \vec{u}_i}{\partial t} \) term, ignore the \( \rho_i \vec{u}_i \nabla \vec{u}_i \) term, since, typically, \( |\rho_i \vec{u}_i \nabla \vec{u}_i| \ll |\nabla p_i| \), and omit the viscosity term, since our plasmas are normally sufficiently dilute for it not to be important. However, it will be relatively straightforward to include these terms later-on in \( \vec{G} \) if they would be found to be of significance.

As stated before, the only terms in \( \vec{G}_i \) we will consider here, are caused by the partial pressure gradient, the thermophoretic force and an electric field \( \vec{E} \):

\[
f_i^{eff} \vec{v}_i = -\nabla p_i + n_i q_i \vec{E} + R_i^T + \sum_j f_{ij} \vec{v}_j. \tag{7.21}
\]

Here \( q_i = Z_i e \) with \( Z \) the charge number of species \( i \), which can be negative and \( e \) the elementary charge. Note that in the case of neutral species \( i \), equation (7.21) only contains the partial pressure, thermophoretic force, and friction terms.

In addition to equation (7.21), we have two constraints on the velocities \( \vec{v}_i \). The first is on the mass flux, given by equation (7.15), and the second is on the electric current \( \vec{J} \).

Introducing the mobility\(^1\)

\[
\mu_i = \frac{n_i q_i}{f_i^{eff}}, \tag{7.22}
\]

we see that

\[
\vec{J} = \sum_i n_i q_i \vec{u}_i = \sum_i n_i q_i \vec{v}_i = \sum_i \mu_i \left(-\nabla p_i + n_i q_i \vec{E} + R_i^T\right) + \sum_i \mu_i \sum_j f_{ij} \vec{v}_j. \tag{7.23}
\]

Combining this with the conductivity \( \sigma \), given by

\[
\sigma = \sum_i n_i q_i \mu_i, \tag{7.24}
\]

we see that

\[
\vec{J} = \sigma \vec{E} - \sum_i \mu_i \left(\nabla p_i - R_i^T\right) + \sum_i \mu_i \sum_j f_{ij} \vec{v}_j, \tag{7.25}
\]

or, alternatively,

\[
\vec{E} = \frac{\vec{J}}{\sigma} + \frac{1}{\sigma} \left(\sum_i \mu_i \left(\nabla p_i - R_i^T\right) - \sum_i \mu_i \sum_j f_{ij} \vec{v}_j\right), \tag{7.26}
\]

\(^1\)One should realize that due to the fact that \( q_i \) can be negative, the mobility can be negative as well.
which is similar to the expression given in [58]. This shows that the electric field has two contributions. One is directly related to the current density forced through the plasma, and will be denoted by

$$\vec{E}_J = \frac{\vec{J}}{\sigma}. \quad (7.27)$$

The other, which can be present in current-less situations, corresponds to the so-called ambipolar field $\vec{E}_a$ [58, 62]:

$$\vec{E}_a = \frac{1}{\sigma} \left( \sum_i \mu_i \left( \nabla p_i - \vec{R}_i^T \right) - \sum_i \mu_i \sum_j f_{ij} \vec{v}_j \right). \quad (7.28)$$

This general expression for the ambipolar electric field yields the well-known expression for the single ion electron plasma if the last two terms are neglected. This transformation to the simple ambipolar case will be given in the next section.

### 7.2.3 Ambipolar condition for a single ion

We will now show that the diffusion model presented here arrives at the well-known ambipolar diffusion condition of equation (7.2) in the case of a single positive ion.

By multiplying equation (7.21) by $\sigma$, we obtain:

$$\sigma \vec{v}_i = \mu_i \sigma \vec{E}_a - \frac{\sigma}{f_{ij}^{eq}} \nabla p_i + \frac{\sigma}{f_{ij}^{eq}} \vec{R}_i^T + \frac{\sigma}{f_{ij}^{eq}} \sum_j f_{ij} \vec{v}_j, \quad (7.29)$$

under the assumption of $\vec{E}_J = 0$. Substituting $\sigma \vec{E}_a$ yields

$$\sigma \vec{v}_i = \mu_i \left( \sum_j \mu_j \left( \nabla p_j - \vec{R}_j^T \right) - \sum_j \mu_j \sum_k f_{jk} \vec{v}_k \right) - \frac{\sigma}{f_{ij}^{eq}} \nabla p_i$$

$$+ \frac{\sigma}{f_{ij}^{eq}} \vec{R}_i^T + \frac{\sigma}{f_{ij}^{eq}} \sum_j f_{ij} \vec{v}_j$$

$$= \sum_j \mu_i \mu_j \left( \nabla p_j - \frac{n_j q_j}{n_i q_i} \nabla p_i + \frac{n_j q_j}{n_i q_i} \vec{R}_i^T - \vec{R}_j^T \right)$$

$$- \sum_j \mu_i \mu_j \left( \sum_k f_{jk} \vec{v}_k - \frac{n_j q_j}{n_i q_i} \sum_k f_{ik} \vec{v}_k \right) \quad (7.30)$$

If we now consider a single ion species, we obtain

$$\sigma \vec{v}_+ = \mu_e \mu_+ \nabla p_+ + \mu_e \mu_+ \nabla p_e - \mu_e \mu_+ \sum_k f_{ek} \vec{v}_k - \mu_e \mu_+ \sum_k f_{+k} \vec{v}_k. \quad (7.31)$$

Note that the $\vec{R}_k^T$ terms cancel, because $\vec{R}_k^T = -\vec{R}_k^T$. Here we ignore the rightmost terms, since drag due to friction is expected to be of little importance compared to the diffusive flux caused by density gradients. This results in

$$\vec{\Gamma}_+ = n_+ \vec{v}_+ = -\frac{|\mu_e||\mu_+|}{|\mu_e| + |\mu_+|e} \left( \nabla p_+ + \nabla p_e \right). \quad (7.32)$$
Since the mobility is related to the diffusion coefficient through the Einstein relation [63],

\[ |\mu_i| = \frac{D_i e}{kT_i}, \tag{7.33} \]

and \( \mu_e \gg \mu_+, n_e = n_+, \) while ignoring temperature gradients, we can rewrite this to

\[ \Gamma^+ = \nabla n_+ = -\frac{e^2 D_e D_+/ (kT_e T_+)}{e^2 D_e/ (kT_e)} (kT_e + kT_+) \nabla n_+ = -D_+ \left( 1 + \frac{T_e}{T_+} \right) \nabla n_+, \tag{7.34} \]

which is equivalent to the aforementioned expression (7.2) for ambipolar diffusion.

### 7.3 Solving the multi-component diffusion problem

#### 7.3.1 Introduction

In the previous sections, the basic equations for handling the multi-component diffusion problem were introduced. It was shown in section 7.2.2 how this leads to a set of coupled equations which can be presented in a matrix-vector equation. In this section, we will show how to solve these equations in a self-consistent manner as is done in our code [1].

#### 7.3.2 Boundary conditions

At vessel walls or boundaries, particles typically have a flux \( \Gamma_i \) from or towards the wall. The exact magnitude of this flux depends on the actual processes taking place at the wall. We now derive a diffusive velocity at the wall by dividing the flux by the density at the wall:

\[ u_{i\perp} = v_{i\perp} = \frac{\Gamma_i}{n_i}. \tag{7.35} \]

Note that \( u_{i\perp} \) and \( v_{i\perp} \) are equivalent at the wall, since \( u_\perp = 0 \) there.

For example, in the case of recombination reactions at the wall,

\[ X^+ + e^- + \text{wall} \rightarrow X + \text{wall}, \]

the ion-flux to the wall would be driven by the modified Bohm velocity [64]:

\[ \Gamma_+ = \theta_+ n_+ \sqrt{\frac{kT_e}{M_+}}. \tag{7.36} \]

Here, \( \theta_+ \) is a correction factor equal to 1 when the ion mean free path [65]

\[ \lambda_i = \left( \sum_j n_j \sigma_{ji} \right)^{-1} \tag{7.37} \]
is larger than the Debye length [65]

\[ \lambda_D = \left( \frac{\epsilon_0 kT_e}{n_e e^2} \right)^{1/2}, \]  

(7.38)

and equal to

\[ \theta_+ = \frac{\lambda_+}{\lambda_D} \]  

(7.39)

otherwise. This factor is used to correct for the fact that ions will not reach the Bohm velocity due to collisions in the case of higher densities.

If we neglect any sticking of atoms on the wall, an equal flux of neutral atoms would come from the wall. This would then lead to the following wall velocities for the ions and neutrals:

\[ v_{+\perp} = \theta_+ \sqrt{\frac{kT_e}{M_+}}, \]  

(7.40)

\[ v_{0\perp} = -\theta_+ \frac{n_+}{n_0} \sqrt{\frac{kT_e}{M_+}}. \]  

(7.41)

In this manner, we obtain a boundary value for the diffusive velocities. We then use these wall velocities and equation (7.21) to calculate the partial pressure gradients \( \nabla p_{i\perp} \) at the wall. From these, we find the density gradients (using the wall temperatures), leaving us with Neumann boundary conditions that we can use to solve the density field equations.

### 7.3.3 Implementation in our model

The multi-component self-consistent diffusion model described here is integrated in the PLASIMO plasma modelling code. At each model iteration, the system of coupled diffusive velocities are calculated at each point. These velocities are then used in the general particle balance equation (see step 4, below).

So at each iteration, the following steps are taken for the multi-component diffusion:

1. Calculate diffusive velocities \( \vec{v}_i \) by solving the following set of equations for \( \vec{v}_i \):

\[
f_i^{eff} \vec{v}_i = \sum_j f_{ij} \vec{v}_j - \vec{\nabla} p_i + \vec{R}_i^T + n_i q_i \vec{E}_j + \frac{n_i q_i}{\sigma} \left( \sum_j \mu_j \left( \vec{\nabla} p_j - \vec{R}_j^T \right) - \sum_j \vec{v}_j \sum_k f_{jk} \mu_k \right) \]  

(7.42)

The equation for \( \vec{v}_0 \), the velocity for the first particle, is replaced by equation (7.15):

\[
\sum_i n_i m_i \vec{v}_i = 0, \]  

(7.43)
and the equation for \( \vec{v}_e \), the electron’s diffusive velocity, is replaced by Ohm’s law:

\[
\sum_i n_i q_i \vec{v}_i = \sigma \vec{E}_j. \tag{7.44}
\]

So, expressed as a matrix, the following system is solved:

\[
\begin{pmatrix}
  n_0 m_0 & n_1 m_1 & \cdots & n_i m_i & \cdots \\
  f_{10} & -f_i^{\text{eff}} & \cdots & f_{1i} & \cdots \\
  \vdots & \vdots & \ddots & \vdots & \vdots \\
  f_{i0} & \cdots & \cdots & -f_i^{\text{eff}} & \cdots \\
  \vdots & \vdots & \vdots & \vdots & \vdots \\
  f_{j0} - \sum_l \frac{n_{0l}}{\sigma} \sum_k f_{0k} \mu_k & \cdots & \cdots & \cdots & \cdots \\
  \vdots & \vdots & \vdots & \vdots & \vdots \\
  0 & 0 & \cdots & 0 & \cdots \\
\end{pmatrix}
\begin{pmatrix}
  \vec{v}_0 \\
  \vec{v}_1 \\
  \vdots \\
  \vec{v}_i \\
  \vdots \\
  \vec{v}_j \\
  \vdots \\
  \vec{v}_N
\end{pmatrix} =
\begin{pmatrix}
  0 \\
  \vec{v}_1 - \vec{R}_i^T \\
  \vdots \\
  \vec{v}_i - \vec{R}_i^T \\
  \vdots \\
  \vec{v}_j - \vec{R}_i^T - \frac{n_i q_i}{\sigma} \sum_k \mu_k \left( \vec{v}_k - \vec{R}_k^T \right) - n_j q_j \vec{E}_j \\
  \vdots \\
  \sigma \vec{E}_j
\end{pmatrix}, \tag{7.45}
\]

where \( i \) is the index of a neutral particle, and \( j \) is the index of a charged particle.

2. For boundary points, the velocities \( \vec{v}_{i\perp} \) perpendicular to the boundary are known. Use these velocities to calculate the density gradients needed at the boundaries, by solving this set of equations for \( \vec{v}_{p_{i\perp}} \):

\[
\nabla p_{i\perp} = \sum_j f_{ij} u_{j\perp} + \vec{R}_{i\perp}^T - f_i^{\text{eff}} u_{i\perp} + n_i q_i E_{i\perp}
\]

\[
+ \frac{n_i q_i}{\sigma} \left( \sum_j \mu_j \left( \nabla p_{j\perp} - \vec{R}_{j\perp}^T \right) - \sum u_{j\perp} \sum_k f_{jk} \mu_k \right), \tag{7.46}
\]
7.4 Results

\[ \nabla n_{i \perp} = \frac{1}{kT_i} \nabla p_{i \perp} - \frac{n_i}{kT_i} \nabla kT_i \]  \hspace{1cm} (7.47)

3. Use the \( \vec{v}_i \) velocities in the individual density equations (7.3). In order to take better advantage of the specific shape of the density equations, the part of \( \vec{v}_i \) that directly depends on the density gradient is taken into account via a diffusion coefficient, and the rest of \( \vec{v}_i \), given by

\[ \vec{v}_i' = \vec{v}_i + \frac{kT_i}{f_i \eta} \nabla n_i, \]  \hspace{1cm} (7.48)

is added to the convective term.

4. Solve the density equations and calculate new density fields.

\[ \nabla \cdot \left( n_i (\vec{u} + \vec{v}_i') \right) - \nabla \cdot \left( \frac{kT_i}{f_i \eta} \nabla n_i \right) = S_i \]  \hspace{1cm} (7.49)

It was found that the system described by equation (7.45) can be difficult to solve numerically under certain conditions. While iterating in our plasma model, local densities may become such that the conventional Gauss-Jordan method [66] cannot produce a solution for the diffusive velocities. At points where this occurred, we switched to the more robust, but slower, Single Value Decomposition method [66] for a few iterations.

7.4 Results

In order to assess whether the new diffusion implementation is working, a comparison was made with the older diffusion code, based on a simple Fick-like model.

7.4.1 Description of the models

The models used to compare the diffusion implementations describe cylindrical positive column discharges and will be presented in more detail in a future publication (chapter 8). We will confine ourselves to the simple case of non-flowing plasmas, i.e. plasmas in which the bulk velocity equals zero. Such conditions can be found in positive column of a fluorescent lamp.

For these models, we solve the energy equations for the electrons and heavy particles together with density equations for the particles, similar to the manner presented in [52]. Since our goal was to compare the diffusion models, we omitted radiation effects and Penning ionisation and assumed a Maxwellian electron energy distribution for the electrons. While this makes the models unrealistic for the case of medium pressure discharges, it should not affect any comparison of the diffusion behaviour.
Figure 7.1: A comparison between the electron densities calculated using the two diffusion models for an argon discharge.

Although our code is made for two-dimensional problems, the boundary conditions were chosen in such a way that the problem becomes one-dimensional, i.e. only dependent on the radius. Each model was run twice, once with the older Fick-based diffusion model with ambipolar diffusion coefficients, and once with the new diffusion model presented here. After that, the results of both calculations were compared.

7.4.2 Medium pressure argon discharge

The first model that was tried was a simple medium pressure argon discharge, with only three types of particles, neutral argon, singly ionised argon and electrons. The radius of the discharge was 2.5 cm, the argon filling pressure was 400 Pa at 273 K, with a wall temperature of 300 K and a dissipated electromagnetic power of 0.35 W/cm. Since the older Fick-based diffusion model used ambipolar diffusion coefficients, it is expected to yield reasonable results for this simple case.

Figures 7.1 and 7.2 on the next page show the electron densities and temperatures calculated for both cases. As can be seen, the differences are small. The central value of the electron density for the SCD case is about 15% larger than for the ambipolar diffusion case. The difference between the electron temperatures is less than 0.5%. That there are some differences can be explained by the fact that the gradient of the partial pressure is the driving force in the new diffusion model, while the Fick model only considers the density gradient. Since there exist small gradients in the temperature, this produces slightly different results for the new diffusion code.
7.4 Results

Figure 7.2: A comparison between the electron temperatures calculated using the two diffusion models for an argon discharge.

7.4.3 A high pressure argon discharge

In order to make the effect of temperature gradients on the diffusion more clear, we also simulated a high pressure argon discharge, which had conditions similar to those of the medium pressure argon discharge, with the exceptions of a higher pressure (argon filling pressure of 20000 Pa at 273 K) and a higher EM power dissipation of 55 W/cm.

As can be seen from figures 7.3 and 7.4 on the following page, the high pressure discharge can considered to be in local thermal equilibrium. In addition, both the Fick approximation and the self-consistent approach render similar results for the temperatures.

Figure 7.5 on page 91 shows a difference of up to 30% between the old Fick model and the new self-consistent calculation. This is due to the fact that temperature gradients are not considered in the first case, and are taken into account in the second case. That the difference is not larger can be explained by the fact that the local thermal equilibrium causes the densities to be mainly determined by the local ionisation vs. recombination balance, as opposed to the ionisation vs. diffusion balance in the medium pressure case. In other words, the diffusion process is of less significance in this high-pressure case.

7.4.4 Argon mercury discharge

A more serious test of our new multi-component diffusion code was done using a more complex discharge model, where a medium pressure, combined argon-mercury plasma was considered. The conditions were equal to those of the me-
Chapter 7: Implementation of a friction-based self-consistent multi-component...

Figure 7.3: A comparison between the heavy particle temperatures calculated using the two diffusion models for a high-pressure argon discharge.

Figure 7.4: A comparison between the electron temperatures calculated using the two diffusion models for a high-pressure argon discharge.
7.4 Results

Figure 7.5: A comparison between the electron densities calculated using the two diffusion models for a high-pressure argon discharge.

dium pressure argon discharge, with the obvious exception that the mercury wall density was set at $10^{20}\text{m}^{-3}$.

For mercury, the ground state, the first ionisation state, and the first four excited states where included in the model as separate particles, in addition to the argon ground state, and first ionisation state, and the electron. Because of the presence of multiple ions, the old diffusion model is not expected to produce realistic results, but it can still serve as a reference to compare the new diffusion model to.

Figures 7.6 and 7.7 on the following page show the densities of the charged particles and of the electron temperature for both models. As can be seen, the differences in the charged particle densities are considerable, the electron density is 3 times higher in case of the self consistent diffusion model. This was to be expected, since the new model describes diffusion of charged particles differently from the Fick model. Since the densities of the argon and mercury ions are comparable, the naive approach of the Fick model to describe ambipolar diffusion is clearly incorrect. As a side line, it should be mentioned here that the argon ion density is expected to drop considerably with respect to the mercury ion density, if Penning ionisation and a non-Maxwellian electron energy distribution function would be included.

Since the main channel of destruction for charged particles is diffusion towards the wall followed by recombination at the wall, the densities of charged particles will be strongly affected by differences in their diffusion descriptions. In the case of the ambipolar model, the ion diffusion rates are overestimated, leading to a larger local loss term, which needs to be compensated by a higher ionisation rate and, hence, a higher electron temperature.
Figure 7.6: A comparison between the charged particle densities calculated using the two diffusion models for an argon-mercury discharge.

Figure 7.7: A comparison between the electron temperatures calculated using the two diffusion models for an argon-mercury discharge.
7.5 Discussion

Figure 7.8: A comparison between the excited mercury densities calculated using the two diffusion models for an argon-mercury discharge.

Figure 7.8 shows the excited mercury densities for both diffusion models. As can be seen, these are not changed significantly by the new model.

7.5 Discussion

As shown, the new diffusion code gives results that look reasonable compared to the older Fick-based model. It is clear that the new model works, and produces a significant differences in the case of multiple ions in our medium pressure discharges, among other things in the confinement of ions.

The new code will allow us to model phenomena that where not possible using the old Fick-based model. Examples of these are cataphoresis-effects, where ions and neutrals are somewhat separated due to an electric field, and demixing due to gravitational effects. Furthermore, it is straightforward to add other diffusive forces to the model or to incorporate a viscosity term. Finally, the new code proved to be much more stable and robust than the multi-component diffusion model incorporated in the older PLASIMO version.

In a future paper, we will present the results of an Ar-Hg positive column model that uses the diffusion model presented here.

Acknowledgement

This research is supported by the Technology Foundation STW, Applied Science Division of NWO. The authors wish to thank D.C. Schram for his helpful comments.
Chapter 7: Implementation of a friction-based self-consistent multi-component...

7.A A multiple ion Fick-like diffusion description

7.A.1 Introduction

So far in this chapter, we presented a friction-based self-consistent multi-component diffusion model. This model was compared to a more conventional Fick-based ambipolar diffusion model. As was shown in figures 7.6 and 7.7 on page 92, the latter only provides reasonable results for a mixture that contains one dominating singly ionised positive ion, and has no significant temperature gradients. In the following we will present an extension of the Fick-based model to describe ambipolar diffusion for a neutral mixture of neutrals, electrons and multiple singly ionised ions.

7.A.2 Theory

The basic equation for the diffusive velocity of a species $i$ is:

$$ f_i^{\text{diff}} \vec{v}_i = -\nabla p_i + n_i q_i \vec{E}_j + n_i q_i \vec{E}_a + R^T_i + \sum_j f_{ij} \vec{v}_j. \quad (7.50) $$

Since the thermophoretic force $R^T_i$ is of secondary importance, we will ignore it in the following derivation.

If we now multiply equation (7.50) by $\sigma$ and divide by $f_i^{\text{diff}}$, we obtain:

$$ \sigma \vec{v}_i = \mu_i \sigma \vec{E}_a - \frac{\sigma}{f_i^{\text{diff}}} \nabla p_i + \frac{\sigma}{f_i^{\text{diff}}} \sum_j f_{ij} \vec{v}_j, \quad (7.51) $$

under the assumption of $\vec{E}_j = 0$, i.e. no net electrical current. Substituting

$$ \sigma \vec{E}_a = \sum_i \mu_i \nabla p_i - \sum_i \mu_i \sum_j f_{ij} \vec{v}_j, \quad (7.52) $$

yields

$$ \sigma \vec{v}_i = \mu_i \left( \sum_j \mu_j \nabla p_j - \sum_j \mu_j \sum_k f_{jk} \vec{v}_k \right) - \frac{\sigma}{f_i^{\text{diff}}} \nabla p_i + \frac{\sigma}{f_i^{\text{diff}}} \sum_j f_{ij} \vec{v}_j, $$

$$ = \sum_j \mu_i \mu_j \left( \nabla p_j - \frac{n_j q_j}{n_i q_i} \nabla p_i \right) - \sum_j \mu_i \mu_j \left( \sum_k f_{jk} \vec{v}_k - \frac{n_j q_j}{n_i q_i} \sum_k f_{ik} \vec{v}_k \right) \quad (7.53) $$

If we ignore the rightmost term, under the assumption that drag due to friction is of little importance compared to the flux caused by density gradients, we obtain the following:

$$ \sigma \vec{v}_i \approx \sum_j \mu_i \mu_j \nabla p_j + \sum_j -\frac{n_j q_j}{n_i q_i} \mu_i \mu_j \nabla p_i. \quad (7.54) $$

And if we assume $i$ to be a singly ionised ion, and take into account that $\mu_e \gg \mu_{j \neq e}$, we can rewrite this as:

$$ \sigma \vec{v}_i \approx \mu_i \mu_e \nabla p_e + \frac{n_e}{n_i} \mu_i \mu_e \nabla p_i. \quad (7.55) $$
7.A A multiple ion Fick-like diffusion description

We now consider the diffusive flux $\vec{\Gamma}_i = n_i \vec{u}_i$ and see that

$$\vec{\Gamma}_i = \frac{n_i \mu_i e}{\sigma} \nabla p_e + \frac{n_e \mu_e e}{\sigma} \nabla p_i.$$  \hfill (7.56)

Approximating the conductivity by $\sigma \approx n_e q_e \mu_e$, we obtain

$$\vec{\Gamma}_i = -\frac{n_i \mu_i e}{n_e e} \nabla p_e - \frac{\mu_i}{e} \nabla p_i.$$  \hfill (7.57)

And by ignoring the temperature gradients and substituting the Einstein [63] relation

$$|\mu_i| = \frac{D_i e}{k T_i},$$  \hfill (7.58)

we finally arrive at

$$\vec{\Gamma}_i = -D_i \frac{n_i k T_e}{n_e k T_i} \nabla n_e - D_i \nabla n_i.$$  \hfill (7.59)

If we now consider the particle balance

$$\nabla \cdot \left( n_i \vec{u}_i \right) = \nabla \cdot \left( n_i \vec{u} + \vec{\Gamma}_i \right) = S_i,$$  \hfill (7.60)

we can substitute equation (7.59), obtaining a particle balance that “fits” the $\phi$-equation (2.1):

$$\nabla \cdot \left( n_i \left( \vec{u} - D_i \frac{T_e}{n_e T_i} \nabla n_e \right) \right) - \nabla \cdot \left( D_i \nabla n_i \right) = S_i.$$  \hfill (7.61)

### 7.A.3 Comparison

In order to assess the performance of the multi-ion ambipolar diffusion model, we compare it to our self-consistent diffusion model [21] and a ambipolar diffusion model that is only valid for a single ion species.

#### Description of the model

The models used to compare the diffusion implementations describe cylindrical positive column discharges and will be presented in more detail in a future publication. We will confine ourselves to the simple case of non-flowing plasmas, i.e. plasmas in which the bulk velocity equals zero. Such conditions can be found in positive column of a fluorescent lamp.

For these models, we solve a set of two different energy equations, one for the electrons and one heavy particles, together with density equations for the particles, similar to the manner presented in [52]. Since our goal was to compare the diffusion models, we omitted radiation effects and Penning ionisation and assumed a Maxwellian electron energy distribution for the electrons. While this makes the models unrealistic for the case of low pressure discharges, it should not affect any comparison of the diffusion behaviour.

Although our code is made for two-dimensional problems, the boundary conditions were chosen in such a way that the problem becomes one-dimensional,
only dependent on the radius. Each model was run three times, once with the older Fick-based diffusion model with ambipolar diffusion coefficients, once with the self-consistent diffusion model, and once with the new diffusion model presented here. After that, the results of both calculations were compared.

The model that was used was a simple low pressure argon and mercury discharge, with seven types of particles, neutral argon and mercury, excited argon and mercury, singly ionised argon and mercury, and electrons. The radius of the discharge was 2.5 cm, the argon filling pressure was 400 Pa at 273 K, mercury wall density was set at $10^{20}$ m$^{-3}$ with a wall temperature of 300 K and a dissipated electromagnetic power of 0.35 W/cm.

**Results**

Figure 7.9 shows the radial electron densities resulting from the three distinct diffusion models. As can be expected, the multi-ion ambipolar model is closer to the self-consistent model than the single-ion ambipolar model.

As can be seen in figure 7.10 on the facing page, the electron temperatures for the multi-ion ambipolar and self-consistent model are in good agreement, while the single-ion ambipolar model yields significantly different results.

Figures 7.11 on the next page and 7.12 on page 98 show that the ion density profiles for argon and mercury as calculated by the multi-ion ambipolar model are similar to those calculated by the self-consistent diffusion model, while the single-ion ambipolar model predicts different profiles.
Figure 7.10: The electron temperatures as calculated by the three diffusion models for an Ar-Hg discharge.

Figure 7.11: The argon ion densities as calculated by the three diffusion models for an Ar-Hg discharge.
Figure 7.12: The mercury ion densities as calculated by the three diffusion models for an Ar-Hg discharge.

7.A.4 Discussion

While the presented ambipolar diffusion model for multiple ions is far simpler than a complete self-consistent model, it produces similar results for fluorescent lamp-like positive column discharges. Its results are significantly better than the ones following from the single-ion ambipolar diffusion model.
Chapter 8

The influence of de-Maxwellisation on a fluorescent lamp simulation

Abstract

A simple three electron temperature model was applied to a one-dimensional fully self-consistent model of a positive column discharge as typically found in fluorescent lamps. It was shown that this model yields results that are in good agreement with what is generally considered to be true for these kinds of discharges, and that it constitutes a significant improvement over a calculation based on a Maxwellian EEDF. This work has been submitted for publication in a slightly different form [22].

8.1 Introduction

Fluorescent lamps account for a significant proportion of the world’s electric light consumption. One of their advantages is the relatively high efficiency as compared to conventional incandescent lamps. In order to improve these lamps or adapt them to other uses, a better understanding of the mechanisms involved in such a lamp is necessary. For this reason, many experimental and numerical studies have been devoted to these light sources [2,3,10,11,67,68].

In order to gain insight into the influence of a non-Maxwellian electron energy distribution function (EEDF) on a fluorescent model, and to assess the applicability of the model PLASIMO [1] to such plasmas, we constructed a model of the positive column in a fluorescent lamp discharge using this modelling toolkit.

The model includes a combination of features that makes it rather unique in the field of simulating such plasmas. As will be described further-on, we combined two collisional radiative models (CRMs), one for argon, and one for mercury, both using an analytical cut-off technique. This allowed us to take ladder-like excitation and ionisation into account, in addition to any radiative losses from excited states not explicitly considered. Furthermore, we used a three electron group model to approximate the EEDF, since this is believed to play an important role in such plasmas. Our model is fully one-dimensional,
so that all quantities are calculated as a function of radius. By using a self-consistent diffusion model, we believe to consider all relevant diffusion-related effects. Finally, radiation of the two main mercury lines (253.7 nm and 185.0 nm) was taken into account through ray tracing, explicitly considering any effects of these lines. We did not include any bulk flow since this does not play a role in the plasma under consideration.

8.1.1 The fluorescent lamp

A fluorescent lamp typically contains mercury as a radiator and a noble gas that acts as a buffer. While there exists a large variety of fluorescent lamps, we chose to simulate a classic industry-standard 4 ft 40W F40T12 lamp. This is a tubular discharge with a diameter of 38 mm, and a electrical energy dissipation of 0.3 W/cm. In these lamps, argon is used as a buffer gas, with a filling pressure of 400 Pa at 297 K. The mercury density of $2.5 \times 10^{20} \text{ m}^{-3}$ was fixed on the wall, at a wall temperature of 310 K. This corresponds to a mercury vapour pressure of about 8 mTorr. The time-averaged electrical dissipation of 0.3 W/cm is in agreement with the power loss of 3 to 4 W in the cathode-fall that occurs in such a lamp [69]. Since we did not consider forced diffusion due to the axial electric field, the simulated discharge is homogeneous in this direction. For this reason, we could model a slice of arbitrary length, making the problem effectively one-dimensional. Although the PLASIMO code is intended for two-dimensional models, we chose the boundary conditions such that we obtained an one-dimensional simulation.

The plasma produced in a fluorescent lamp discharge is not in local thermal equilibrium. While the heavy particles typically have a temperature of about 300 K, the electrons are significantly hotter, with a temperature of about 1 eV (11600 K). However, even the notion of an electron-temperature is an approximation in this case, since the EEDF is not Maxwellian.

In [20] (chapter 6), we presented a 3 electron temperature approximation of the EEDF for conditions typically found in fluorescent lamps and showed that the deviation from a Maxwellian electron energy distribution can significantly reduce the effective argon ion production with about a factor of 30. In order to study the effect of the non-Maxwellian EEDF on an Ar-Hg positive column, we constructed a model of such a plasma, and compared the results obtained with both a Maxwellian EEDF and a non-Maxwellian EEDF as approximated by the 3 electron-group model. In addition, we determined the importance of so-called Penning ionisation [69] by comparing the results of a simulation without the Penning effect with the other two simulations.
8.2 Overview of the model

8.2.1 Introduction

The PLASIMO plasma modelling platform can be used to simulate a large variety of flowing and non-flowing plasma configurations. In this publication, we present the results of a fluorescent lamp model constructed with this toolkit. The PLASIMO code currently has two transport modi, one for the fluid aspects, and another to treat the transport of radiation.

8.2.2 Fluid transport

We will first deal with the fluid transport in PLASIMO which is based on a control volume method. Much of its details were already discussed in [52], so we will confine ourselves to a brief overview at this point. In the general case, three types of balance equations are solved; the mass balance, the momentum balance and the energy balance.

The mass and momentum balances

The first balance that needs to be considered is the specific particle mass balance:

\[ \frac{\partial \rho_i}{\partial t} + \nabla \cdot (\rho_i \vec{u}_i) = m_i S_i, \quad (8.1) \]

where \( S_i \), the local particle source term for species \( i \), gives the production of particle \( i \) per unit of time and volume, \( \rho_i \equiv n_i m_i \) is the partial mass density, \( m_i \) the mass, and \( \vec{u}_i \) is the average velocity of species \( i \). Summation of equation (8.1) over all species yields the continuity equation, which states that there is no source or sink for the total mass. The term \( S_i \) is used to include the influence of chemical interactions and radiation transport in the mass balances, as will be explained later-on.

The second type of balance is the specific momentum conservation equation, which for species \( i \) reads

\[ \frac{\partial \rho_i \vec{u}_i}{\partial t} + \nabla \cdot (\rho_i \vec{u}_i \vec{u}_i) = -\nabla \cdot P_i + \vec{u}_i m_i S_i + \vec{F}_i + \vec{R}_i, \quad (8.2) \]

where the pressure tensor \( P_i \) in \( \nabla \cdot P_i = \nabla p_i + \nabla \cdot \pi_i \) (with \( p_i \) the partial pressure, and \( \pi_i \) the viscosity tensor), is developed with respect to the system in which species \( i \) is at rest. \( \vec{F}_i \), the body force acting on species \( i \), contains, among others, contributions from the gravitational force and electric fields, and \( \vec{R}_i \) is the force (per unit volume) caused by friction with other diffusing species [21].

These mass and momentum balances in principle have to be solved for each of the species considered in the model. In the absence of bulk flow, there is no convection and all transport in the mass balance equation (8.1) is caused by diffusion. A common method of considering diffusion is using \( n_i \vec{u}_i = -D_i \nabla n_i \), yielding the well-known Fick diffusion equation

\[ -\nabla \cdot D_i \nabla n_i = S_i, \quad (8.3) \]
in a steady-state situation, a simplification of equation (8.1). Here, $D_i$ is the so-called diffusion coefficient, for ions usually multiplied by $1 + T_e/T_h$ to account for ambipolar diffusion [53].

However, this approach is only valid for the case of a single (ionic) species diffusing through a dominant background gas of uniform temperature. For the case of multiple ions and/or temperature gradients, a more advanced description of particle transport is necessary. In [21], we presented a self-consistent multicomponent diffusion model that calculates the individual diffusive velocities $\vec{v}_i$ of the species present. In this context, diffusive velocities are velocity components that do not give rise to a net mass flow, as opposed to the bulk velocity that does give rise to a net mass flow.

At this point we will provide a brief overview of this model, and refer to [21] for more details. The diffusive velocities of each species are calculated using the following set of equations.

First, by definition, the diffusive velocities do not give rise to a net transport of mass:

$$\sum_i p_i \vec{v}_i = 0. \tag{8.4}$$

Second, the transport of charged particles should yield the electric current density $\vec{J}$:

$$\sum_i n_i q_i \vec{v}_i = \vec{J} = \sigma \vec{E}, \tag{8.5}$$

where $q_i$ is the charge of species $i$, and $\vec{E}$ the applied electric field.

Finally, the forces acting on each species are balanced by the friction caused by the difference in velocity with respect to the other species:

$$\sum_j f_{ij} (\vec{v}_i - \vec{v}_j) = -\nabla p_i + n_i q_i \vec{E} + R^T_i. \tag{8.6}$$

Here, $f_{ij}$ is the friction coefficient between species $i$ and $j$, and $R^T_i$ is the thermophoretic force on species $i$. The set of equations (8.4-8.6) can now be used instead of equation (8.2). By solving this set at each point, diffusive velocities are obtained that can then be used in the mass balance equations. In principle, equations (8.1) and (8.6) will have to be solved for all particles present. However, as will be shown later-on, it is possible to strongly reduce the number of particles that need this explicit treatment.

The energy balance

The third type of balance is the energy balance. For species $i$ it reads:

$$\frac{\partial (n_i \epsilon_i)}{\partial t} + \nabla \cdot (n_i \epsilon_i \vec{u}_i) + \nabla \cdot \vec{P}_i + \nabla \cdot \vec{q}_i + \nabla \cdot \vec{q}^*_i = Q_i, \tag{8.7}$$

where $\epsilon_i$ is the internal energy of species $i$, $\vec{q}^*_i$ the heat flux vector, and $Q_i$ is the heat source caused by elastic and inelastic interactions and/or chemical interaction with other species. For an ideal gas, $\epsilon_i = c_v T_i$, with $c_v$ the specific heat at
constant volume and $T_i$ the temperature of species $i$. The heat flux is given by $q_i = -\lambda_i \nabla T_i$, where $\lambda_i$ is the thermal conductivity.

Due to the large mass ratio between heavy particles and electrons, the energy transfer between the electron gas and all heavy particles is not very effective. On the other hand, the energy exchange between species of the same or comparable mass is rather easy, so that we may assume that the plasma can from the aspect of energy be divided in two groups: the electron $e$ and the heavy particles $h$. This reduces the number of energy balances to two.

The presented model of the fluorescent lamp is not time-dependent, and does not have to deal with a bulk velocity. Hence, the energy balance for the heavy particles can be written as a balance of energy gain from the electrons and loss by heat conduction:

$$\vec{\nabla} \cdot \vec{q}_h = Q_{eh},$$

where $Q_{eh}$ is the elastic energy transfer between the electrons and heavy particle species, given by

$$Q_{eh} = n_e \sum_h \left( \frac{2m_e}{m_h} \right) \frac{3}{2} k_b (T_e - T_h) \langle \nu_{eh} \rangle,$$  (8.9)

with $\langle \nu_{eh} \rangle$ the collision frequency for electron-heavies collisions, averaged over the Maxwellian electron energy distribution.

The electron energy balance is more complicated and expresses how electrons heated by the electric field transfer most of this energy to the heavy particles by means of elastic and inelastic collisions. This brings us to the following form of the electron energy balance:

$$\vec{\nabla} \cdot \vec{q}_e = Q_{Ohm} - Q_{in} - Q_{eh}.$$  (8.10)

with $Q_{in}$ the inelastic source term, that will be dealt with in more detail later-on.

The input power density $Q_{Ohm}$ is given by

$$Q_{Ohm} = \vec{J} \cdot \vec{E} = \sigma |\vec{E}|^2.$$  (8.11)

Here, $\vec{J}$ is the current density, $\vec{E}$ the electric field and $\sigma$ the conductivity.

In our steady-state simulation we used the time-average of the RMS value of $\vec{E}$ and thus $Q_{Ohm}$. At each iteration, an electric field was calculated through the following expression:

$$\bar{E} = \left( \frac{P_{Diss.}}{f_v \sigma dV} \right)^{1/2} \bar{e}_z,$$  (8.12)

where $P_{Diss.}$ is the total power dissipated in the plasma volume, an input parameter set by the modeller. Note that this expression is based on the assumption of a constant electric field throughout the plasma. It does not take the small so-called fringe-fields into account, as we are only concerned with dissipation.
8.2.3 Transport coefficients

In order to accurately model the transport of particles and energy, transport coefficients are needed. The electron thermal conductivity is calculated based on the Frost mixture rules [70], while the electric conductivity is calculated using the expression given in [21]. The heavy particle transport coefficients are those of Devoto [71–73], based on the Chapman-Enskog-Burnett theory [74]. The heavy particle thermal conductivity is used in third order approximation; the electron-heavy particle collision terms are neglected.

The elastic momentum transfer cross sections for electron-argon interaction is the expression determined by Milloy [75], while the electron-mercury cross sections where based on Rockwood [76]. For all other cross sections either a hard sphere approximation (for neutral-neutral interactions), a Coulomb approximation (charged-charged) or a Langevin approximation (charged-neutral) was used. For more information on the transport coefficients, we refer to [17].

8.2.4 Collisional radiative models

While the species in a plasma mixture typically have a large number of excited levels, it would be impractical to include all these levels as explicit particle types in the model. Fortunately, as explained in [42, 77], we can make a distinction between local chemistry levels (LC) and transport sensitive (TS) levels. The production and destruction of the LC levels are only determined by local collisional and radiative reactions. For this reason, the densities of these levels are directly related to the local densities of other TS levels, and can thus be treated locally. In contrast with this, the production and destruction of TS levels are determined by a competition between transport processes (diffusion and/or radiation transport from other parts of the plasma) and local chemical reactions. For this reason, spatial interactions have to be taken into account in this case. It should be realized that equations (8.1) and (8.6) only have to be solved for TS levels.

By using a collisional radiative model, it is possible to calculate the densities of the local levels from the densities of the transport sensitive levels, the electron density and the electron energy distribution. On the other hand the LC levels facilitate the transitions between TS levels. For instance, stepwise ionisation, involving various LC levels, will lead to an enhancement of the excitational flux between the ground level of the atom and the ion. We used two collisional radiative models, one for argon [51] and one for mercury [47]. Both models calculate effective transfer rates matrix \( J \) between the transport sensitive levels and the effective emissivity vector \( L \) (both introduced in [77]) of these levels, as a function of the electron density \( n_e \) and three electron temperatures. The effective transfer rates are then used to derive the particle source terms \( S_i \). Through the use of a hybrid cut-off technique [42], the number of excited levels considered in the CRM for argon and mercury could be limited to 21 and 19, respectively.

The three electron temperature model presented in [20] is used to approximate the EEDF used in the CRMs, providing an electron bulk temperature \( T_e \), and two tail temperatures \( T_{t,Hg} \) and \( T_{t,Ar} \). These tail temperatures are calculated
at each spatial position as a function of the electron density, the electron bulk temperature and the argon and mercury ground state densities. They are then used in combination with $T_e$ and $n_e$ as indices into a table of transfer rates calculated separately by the CRM.

Apart from calculating the particle source terms $S_i$, $J$ and $L$ are also used to calculate the inelastic energy transfer term for the electrons:

$$Q_{in} = (E)^T J n - L^T n.$$  \hfill (8.13)

Here, $E$ is a vector containing the energies of the transport sensitive levels, and $n$ a vector containing the densities of these levels. The first term on the RHS represents the change in energy of the electrons due to electron induced transfer between TS levels, while the second term represents the energy loss due to radiation.

### 8.2.5 Transport of radiation

As mentioned before, radiation transport is the other transport mode included in the PLASIMO modelling toolkit. Since the model presented is concerned with a plasma that is used as a lightsource, it is important to correctly consider this transport.

Previously, this has been attempted by using so-called escape factors $\Lambda$ to modify the radiative transition probabilities. In this approximation, the transition probability $A_{pq}$ for a transition from state $p$ to state $q$ is replaced by $\Lambda_{pq}A_{pq}$. As shown in equation (5.14), the escape factor $\Lambda_{pq}$ is calculated by integrating over the line profile. This equation is repeated here with a slightly different notation:

$$\Lambda_{pq} = 1 - (n_p A_{pq})^{-1} \int (n_q B_{qp} - n_p B_{pq}) \rho_\nu d\nu,$$  \hfill (8.14)

where $\rho_\nu$ is the spectral energy density, and $A_{pq}$, $B_{qp}$, and $B_{pq}$, are the probability coefficients for spontaneous emission, absorption, and stimulated emission, respectively. In this formulation, $\Lambda = 1$ for cases where absorption and stimulated emission can be neglected, and $\Lambda < 0$ when local absorption exceeds local emission. Equation (8.14) shows that the escape factor depends on $\rho_\nu$ and thus on the radiation produced in other parts of the plasma. If a constant escape factor is assumed, as is done by most researchers, there is no need to calculate $\rho_\nu$ as a function of plasma location.

A more advanced method of dealing with radiation transport is presented in [16]. We will provide a brief outline of this method here. It is based on the use of a discrete number of probe lines to sample the radiation coming from different directions. This is done by solving the equation for radiative transfer along these probe lines:

$$\frac{dI_\nu}{ds} = j_\nu - \kappa(\nu) I_\nu,$$  \hfill (8.15)

where the LHS represents the change in radiation intensity $I_\nu$ [Wm$^{-2}$Hz$^{-1}$sr$^{-1}$] along a path $s$, $j_\nu$ [Wm$^{-3}$Hz$^{-1}$sr$^{-1}$] is the local emission coefficient, and $\kappa(\nu)$ [m$^{-1}$] the local coefficient for absorption. These local plasma properties are
delivered by the fluid model so that $I_\nu$ can be determined for various directions at all plasma positions by using equation (8.15).

The spectral energy density $\rho_\nu$ is determined via the direction averaged intensity $J_\nu$ defined as:

$$J_\nu = \frac{1}{4\pi} \int 4\pi I_\nu d\Omega,$$

which is related to $\rho_\nu$ by means of

$$J_\nu = \frac{1}{c} \int 4\pi \rho_\nu d\Omega,$$

where $c$ is the speed of light.

In principle, we need to calculate $I_\nu(\vec{r})$ for all directions $\vec{\Omega}$. In our model this is approximated by using a limited number of probe lines to sample the continuum of directions $\vec{\Omega}$:

$$J_\nu = \frac{1}{4\pi} \int 4\pi I_\nu d\Omega \approx \frac{1}{4\pi} \sum_i I_{\nu,i} \Delta\Omega,$$

where the intensities $I_{\nu,i}$ are obtained by integrating equations (8.15) along the probe lines.

Once $J_\nu$ is known, the influence of the radiation on the local species densities can then be accounted for by

$$S_{pq} = \int \frac{4\pi}{h\nu} (\kappa(\nu)J_\nu - j_\nu) d\nu,$$

where $S_{pq}$ is the transfer rate from the lower state $p$ to the upper state $q$ corresponding to $h\nu$. This transfer rate is included in the source terms $S_i$ of equation (8.1) for the species involved. Note that $I_\nu$ is a function of $j_\nu$ and $\kappa$, which in turn depend on the species densities, making an iterative solution procedure necessary.

In our fluorescent lamp model, we used this method to include the radiation transport of the two dominant mercury lines $6^1P_1 \rightarrow 6^1S_0$ at 253.7 nm and $6^1P_1 \rightarrow 6^1S_0$ at 185.0 nm. To account for the isotopic effects as discussed by [78], we reduced the radiation absorption term for these two lines by a factor of 5. All other radiative transitions occurring in the plasma were treated in the CRM using the escape factor method.

### 8.2.6 Mixture

The mixture in our model consists of ten distinct “transport sensitive” particle types: electron, argon ground state, argon first excited state ($^3P_0$), argon ion state, mercury ground state, the first four excited states of mercury (the two metastable states $6^3P_0$ and $6^3P_2$, and the two main radiating states $6^3P_1$ and $6^1P_1$), and the mercury ion state (see figure 8.1 on the next page). In order to consider such a mixture, we solved mass balances for eight of the particle types. The density of the argon ground state was adjusted at each iteration to
8.2 Overview of the model

Variable colour fluorescent lamps

Figure 2.3.1: Schematic drawing of the working principle of a fluorescent lamp. The electrical current is injected into the discharge by means of two electrodes that are positioned at the ends of the discharge tube. The electrodes provide the electrons necessary to maintain the discharge. The electrodes are covered with a mixture of the oxides of barium, strontium and calcium. These materials are known for their relatively high electron emission, even at low temperatures as 800°C. The electrodes emit the electrons by means of thermionic emission. The electrodes are incorporated in an electric system, which contains a stabilising device, a ballast. Stabilisation is necessary since the voltage-current characteristic of the discharge has a negative slope.

The discharge tube is filled with a mixture of noble gases and mercury, the latter being present in both liquid and vapour form. The pressure of the noble-gas mixture is a few hundred Pa, the mercury vapour pressure is approximately 1 Pa. This is the vapour pressure at a temperature of 42°C. Although the noble-gas pressure is much higher than the mercury pressure, the discharge is dominated by mercury. Virtually no excitation and ionisation of the noble-gas atoms occurs. This is due to the fact that the energy of the excited states of the noble-gas atoms is much higher than the energy of the excited states of mercury. This is clearly illustrated in figure 2.3.2, which shows the energy schemes of mercury and neon.

Figure 2.3.2: The energy schemes of the atoms of mercury (left) and neon (right). Note the different energy scales.

Compared to the energy of the excited states, the electron temperature is rather low. This results in the presence of much more inelastic collisions of electrons with mercury than with neon. For example, in a Maxwellian electron energy distribution with a temperature of 1 eV, \(9 \times 10^4\) times more electrons are involved in inelastic collisions with mercury than with neon. Moreover, the amount of high-energy electrons is reduced by the inelastic collisions with mercury. This depletion of high energetic electrons favours the inelastic collisions with...
rather arbitrary, it was found to have little influence on the final results of the simulation. For this reason, no effort has been made to refine this percentage.

8.3 Results

In order to assess the importance of the three temperature EEDF, we compared results obtained using this description with those calculated under the assumption of a Maxwellian EEDF. In addition, we repeated the calculation with a three temperature EEDF, but without taking the Penning ionisation effect into account.

As can be seen by comparing figures 8.2 and 8.3 on the next page, including a non-Maxwellian EEDF has a strong effect on the number of argon ions present in the discharge. In the Maxwellian case, the density of argon ions is about 13% of the total ion density, while the non-Maxwellian calculation results in between 2% and 4% of the ions being argon. This is in much better agreement with experimental observations. As was shown in [20], the reduction of the number of higher energy electrons decreases the argon ionisation rate, and thus the production of argon ions. The production of mercury ions, and the excitation of mercury atoms is affected less, because of the lower ionisation and excitation energies.

However, omitting the Penning effect, while including a non-Maxwellian EEDF (figure 8.4 on page 110), results in an argon ion-density of about 8% of the total ion density. This shows that the Penning effect is a strong mechanism for the reduction of argon ionisation and an important channel for mercury ionisation.
8.3 Results

Figure 8.3: The argon and mercury ion densities as a function of radius in the case of a three group EEDF. In this case the argon ionisation is significantly lower than for the Maxwellian case.

<table>
<thead>
<tr>
<th></th>
<th>Current (mA)</th>
<th>Radiated power (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Literature [69]</td>
<td>425</td>
<td>65</td>
</tr>
<tr>
<td>non-Maxwellian EEDF</td>
<td>427</td>
<td>69</td>
</tr>
<tr>
<td>Maxwellian EEDF</td>
<td>653</td>
<td>62</td>
</tr>
<tr>
<td>non-Maxwellian EEDF w/o Penning</td>
<td>337</td>
<td>70</td>
</tr>
</tbody>
</table>

Table 8.1: The electric current and radiated power given in literature and as calculated by our three models.

Figure 8.5 on the next page shows the electron temperatures as calculated by the non-Maxwellian, Maxwellian, and no Penning models. The temperature for the Maxwellian model at 9600 K is substantially lower than the expected 11600 K (1 eV) [69], while the model without Penning ionisation results in a temperature of 12400 K that is too high. However, the temperature of 11350 K found for the non-Maxwellian case including Penning is in almost perfect agreement.

Another interesting point of comparison is the amount of current going through the plasma. Under the simulated conditions, we should expect a current of about 425 mA [69]. As stated before, the model adjusts the current through the plasma in such a way that the dissipated power is 0.3 W/cm. As shown in table 8.1, only the non-Maxwellian EEDF (including Penning) yields a result that is in reasonable agreement. The differences in current between the three cases can be explained by the differences in electron density and the resulting conductivity (see figure 8.6 on page 111).

Finally, we compared the percentage of radiated power in the 253.7 nm and
Figure 8.4: The argon and mercury ion densities as a function of radius in the case where the Penning effect is omitted. This results in a (too high) relative argon ion contribution of 8%.

Figure 8.5: The electron temperature for the non-Maxwellian EEDF (includes the Penning effect), the Maxwellian EEDF, and the non-Maxwellian EEDF without Penning ionisation. These last two give unrealistic low and high temperatures, respectively. However, the non-Maxwellian EEDF calculation including Penning ionisation yields a result that is in good agreement with the 11600 K encountered in reality.
185.0 nm lines to the absorbed EM power, i.e. the efficiency of the light source. For this, 65% is commonly suggested [69]. As can be seen in table 8.1, all three models give values that seem reasonable. In the Maxwellian case, we find 62%, while the non-Maxwellian calculation and the one omitting the Penning effect yield somewhat higher values of 69% and 70%, respectively. The fact that the non-Maxwellian calculation gives a value that is slightly too high could be attributed to two factors.

First, the experimental values are specified for an AC driven lamp, and our model describes a DC situation. While the average current and dissipated power are the same for the experiment and the simulation, it is entirely conceivable that the sequence of recombination and ionisation that results form an AC current may very well result in a different average radiation production. In order to verify this it would be necessary to do a (partially) time-dependent simulation.

In addition, our approach to handle isotopic effects might well be oversimplified. A more elaborate model of the radiative effects of the different mercury isotopes could produce better results, but is outside the scope of this article.
8.4 Conclusion

As shown, the inclusion of a non-Maxwellian EEDF through the use of the 3 electron-group model strongly improves the realism of the simulation of the positive column of a fluorescent tube. By itself, the reduction of the argon ions is already an important improvement. However, especially the more realistic current shows that the electrical conductivity and, indirectly, the electron density are in much better agreement with reality. The results of the non-Maxwellian model could be improved by a more thorough account of the time-dependent ionisation and recombination effects in an AC lamp, and by more carefully considering the effects of the various mercury isotopes, most likely improving the radiative energy loss term. Furthermore, it should be noted here that our model yields very satisfying results, considering the fact that all parameters used where based on physical conditions, and no arbitrary adjustments were made. This fact by itself already strengthens confidence in the model that was constructed, even though there still exists room for improvement. Finally, the overall agreement shows that the model could be used to optimise lamp efficiency and performance, and to guide changes in geometry and excitation.

Acknowledgement

This research is supported by the Technology Foundation STW, Applied Science Division of NWO.
Chapter 9

A time-dependent fluorescent lamp simulation

9.1 Introduction

Fluorescent lamps form a very large group of plasmas. Examples of these lamps are the ordinary tube light frequently used for office lighting, energy saving compact fluorescent lamps with their well-known domestic applications, backlighting for liquid crystal displays, and UV lamps for disinfection. Although the concept of these lamps has been known since the 1920s, there still is a substantial lack of insight in the role of the various mechanisms of excitation and (radiation) transport. Most models of such lamps include adjustment factors to increase their agreement with experiments. For this reason, we constructed a model [22] which includes as many processes as possible and is solely based on elementary data, without arbitrary adjustments to improve the results.

We will now present a brief overview of the various mechanisms included in the model, and refer to [22] for more details.

- A total of ten different species were included explicitly. They consisted of three argon states (ground state, first excited state and ion state), six mercury states (ground state, first four excited states and the ion state), and the electron. These species were treated as transport sensitive (TS) species, meaning that their densities were determined from the competition between transport and elementary (de)excitation, ionisation and recombination processes.

- The densities of these TS species were computed by means of ten particle and ten momentum balances. The latter were cast in a friction diffusion-form, allowing the simple Fick diffusion expression to be replaced by a self-consistent diffusion model.

- We used two collisional radiative models (CRMs), one for argon (containing 21 levels) and one for mercury (containing 19 levels), to relate the densities of the other (excited) species to that of the TS levels. These dependent levels are denoted by local chemistry (LC) levels.
Chapter 9: A time-dependent fluorescent lamp simulation

- Penning ionisation of mercury via excited argon was included.
- Two energy balances were used, one for the heavy particles, the other for the electrons.
- A three electron-group model was used to take a non-Maxwellian electron energy distribution function (EEDF) into account.
- Radiation transport was treated using the ray tracing control volume (RTCV) method of van der Heijden [16], replacing the less accurate method of escape factors.

The model produced very good results with respect to transport properties such as the electrical conductivity, the ratio of argon ions to mercury ions, etc. It was found that the efficiency of the modelled lamp, 69% percent of its input power was converted into UV radiation, was close to, but slightly higher than the value of 65% as found in many experimental studies. As pointed out in [22], one of the reasons might be that we modelled a DC lamp whereas the fluorescent lamp is mostly driven with an alternating current.

This paper is devoted to the construction of a time-dependent version of our DC model, in order to gauge the effect of time-dependence on the simulated discharge, particularly in the area of radiation efficiency.

9.2 Setup of the model

The model used in this chapter is essentially the same as the one presented in chapter 8. The only differences are a time-dependent discretisation method and power incoupling, and some simplifications to decrease the running time of the model. First we will discuss modifications to the model which were made to realise a time-dependent model. After that, the simplifications will be dealt with.

9.2.1 Time-dependent modifications

In order to make a time-dependent simulation, a time-dependent discretisation method needs to be utilised. We chose to use the Implicit Euler Method that was introduced in section 4.2.3, for its stability and robustness.

This discretisation method was used for the energy equations for the heavy particles and electrons:

\[
\frac{\partial (c_h T_h)}{\partial t} + \nabla \cdot \vec{q}_h = Q_{eh}, \quad (9.1)
\]

\[
\frac{\partial (c_e T_e)}{\partial t} + \nabla \cdot \vec{q}_e = Q_{Ohm} - Q_{in} - Q_{eh}, \quad (9.2)
\]

The only additions to these equations compared to equations (8.8) and (8.10) are the time-dependent terms. As in the steady-state case, convection flow was not included.
9.2 Setup of the model

Compared to the previous model, the set of specific particle balances that were used was significantly different. Since we only considered one ion, as will be explained in the next section, and because the temperature gradients are small, we decided to use the ordinary Fick equation for (ambipolar) diffusion:

$$\frac{\partial n_i}{\partial t} - \nabla \cdot \left( D_i \nabla n_i \right) = S_i,$$

(9.3)

where $n_i$ is the density of species $i$, $D_i$ is its diffusion coefficient, and $S_i$ is the source term due to local chemical and radiative interactions. The diffusion coefficient $D_i$ was replaced by $D_i^{amb} = D_i (1 + T_e/T_h)$ for the ion, to account for ambipolar diffusion, in accordance with equation (7.2).

The other obvious change is the introduction of a time-dependent power-coupling. This was achieved by using a sinusoidal alternating current with a frequency of 50 Hz and an amplitude of 601 mA. This corresponds to a RMS current of 425 mA. The choice for a harmonic current was made, since experimental data shows that this is a reasonable assumption. The model was run with an adjustable timestep no smaller than 5 $\mu$s (see section 4.2.4), and results were saved approximately every 0.1 ms. Since no effect of the electric field on the ions was included, and only the heating of electrons due to power dissipation was considered, the plasma had a period of 10 ms rather than 20 ms. However, this simulated 10 ms took about 14 days of real calculation time on a 2.8 GHz Pentium 4 system.

9.2.2 Simplifications to the model

Compared to the steady-state simulation, a number of simplifications were made with the aim of reducing the running time of the model. In order to obtain a notion of the impact of these changes on the results, a number of preparatory simulations were done with and without various simplifications, and their results were compared. These preparations were done with a model of a steady-state plasma.

The grid

The first and most obvious change is a reduction of the number of grid points. A reduction of the number of grid points along the radius from 50 to 20 should strongly reduce the runtime of the model. However, this may have consequences for the accuracy of the calculation. To verify the influence of this change, we compared a steady-state calculation as presented in the previous chapter with 50 radial grid points to one with only 20 points. This reduction in the number of points decreased the runtime by about a factor of 4.

Figures 9.1 on the next page and 9.2 on page 117 show the electron temperature and density for the 50-points and 20-points case. As can be seen, the only significant change is in the part of the electron density close to the wall. This is due to the fact that the steep gradient in the density cannot be represented accurately enough with only 20 points. However, the overall influence of this
Chapter 9: A time-dependent fluorescent lamp simulation

Figure 9.1: The electron temperature as calculated by a model with 50 grid points, one with only 20 grid points, one that further omits argon excitation and ionisation, and one that as a further simplification only traces the 253.7 nm Hg-line (the 185 nm line is handled through an escape factor). As can be seen, the reduction in the number of grid points does not have a significant effect, while the removal of the argon chemistry and the simpler radiation model lead to an electron temperatures that are about 4% and 9% higher, respectively.

will be small, since only a small part of the total electrons are affected. In other words, if one is mainly interested in the radiation balance 20 radial points suffice.

The mixture

A further speed-up can be obtained by removing the argon excited state and the argon ion from the model. This seems reasonable since the argon ions are only a few percent of the total ion density, as was shown in figure 8.3 on page 109. In addition, as mentioned before, it is then possible to replace the self-consistent diffusion model by a faster ambipolar Fick model. The electron temperature and density profiles of this are also shown in figures 9.1 and 9.2. This further reduced the runtime by 50%.

Transport of radiation

Finally, we decided to replace the ray tracing of radiation for the 185 nm $^1P_1 \rightarrow ^1S_0$ line by a simple constant escape factor of $10^{-3}$. Since this line accounts for only a small part of the UV radiation transport (about 5%), this seemed acceptable. The most important 253.7 nm $^3P_1 \rightarrow ^3S_0$ line was still considered using the radiation transport model. This further reduced the runtime of the model by
9.2 Setup of the model

The effect of the simplifications as given in figure 9.1 on the electron density. Again, the reduction of grid points does not lead to a significant change, but the omission of argon excitation and ionisation, followed by the simpler radiation model) reduce the electron density by a total 30%.

In order to further assess the impact of the simplifications, table 9.1 lists the current through the plasma and the percentage of radiated power for the various models. As can be seen, the current becomes lower for each additional simplification, due to the reduction of the electron density. The percentage of radiated power only seems to be affected significantly by the introduction of an escape factor for the 185 nm line.

While the modifications presented here do seem to decrease the realism of

<table>
<thead>
<tr>
<th>Model</th>
<th>Current (mA)</th>
<th>Percentage of radiated power</th>
</tr>
</thead>
<tbody>
<tr>
<td>50-point</td>
<td>425</td>
<td>68%</td>
</tr>
<tr>
<td>20-point</td>
<td>384</td>
<td>69%</td>
</tr>
<tr>
<td>no Ar*, Ar+</td>
<td>351</td>
<td>68%</td>
</tr>
<tr>
<td>only 253.7 nm</td>
<td>325</td>
<td>74%</td>
</tr>
</tbody>
</table>

Table 9.1: The current through the plasma and the percentage of radiated power for the various simplifications presented. Note that the total dissipated power was kept at 0.3W/cm for all models. While the current is clearly affected by the simplifications, the radiation only seems to be affected by the change to the radiation model.
the model, the resulting total reduction of the runtime by about a factor of 16 is a necessity, and their impact is not so severe that physically meaningful trends cannot be distinguished. For this reason, it is still interesting to do a time dependent simulation under these conditions. Furthermore, inclusion of all the effects that were removed would result in a simulation taking well over 200 days to complete a single 10 ms period. Clearly, this would not be feasible. For this reason, we decided to trade in some realism for speed, and do a time dependent simulation with the simplified model.

9.3 Results

Figure 9.3 shows the electro-magnetic power that is dissipated in the plasma volume, the radiated UV power, and the imposed current through the plasma as a function of time. As can be seen, the maximum dissipated power is reached before the maximum current. This is caused by the fact that the electron density, and thus the conductivity, is lagging a little behind the current.

In figure 9.4 on the facing page, we divided the dissipated power by the current, yielding the voltage over the plasma volume. This again shows that there exists a phase-difference between current and voltage.

The electron temperature and density in the centre of the discharge are shown in figure 9.5 on the next page. This shows that the electron density lags 1-2 ms behind the current, as was already concluded from the position of the maximum dissipated power.

Figures 9.6 on page 120 and 9.7 on page 121 show the electron temperature
9.3 Results

Figure 9.4: The current through and voltage over one cm of the plasma as a function of time.

Figure 9.5: The electron temperature and density in the centre of the tube as a function of time. Note that the maximum electron temperature is reached before the maximum current, while the maximum electron density is reached after the maximum current.
and density profiles at various times. While the temperature profile remains horizontal at each point in time, the electron density changes shape. The density does not vary evenly over the whole radius. The plasma seems to slightly concentrate the electrons in the centre when the current is increasing (compare 1, 2, 3, 4 and 5 ms), in order to carry the current when the electron production is falling behind. After that, when the current is dropping, the density profiles simply decrease evenly over the radius.

Since the production of radiation is mainly driven by the electron temperature, one would expect a correlation between the two. For this reason, figure 9.8 shows the electron temperature as well as the percentage of radiated power. We plotted the percentage instead of the total power, to remove the influence of the EM power dissipation as a function of time. As can be seen, the link between the temperature and radiated power indeed seems to exist, although it is not a simple relationship.

Finally, table 9.2 on page 122 shows the dissipated EM power and radiated power averaged over the time period. It is interesting to notice that the averaged EM power of 0.32 W/cm is in very good agreement with literature. This is surprising since the model we used has quite a deviation with respect to the current in the steady state case (325 mA for 0.3 W/cm as opposed to 425 mA), as shown in table 9.1 on page 117. For this reason, the significance of the agreement should not be overestimated. Table 9.2 also shows that the percentage of radiated power is marginally lower for the time-dependent case than for the steady-state case.
Figure 9.7: The radial electron density profile as a function of time.

Figure 9.8: The percentage of radiated power and the electron temperature as a function of time. As expected, the maximum radiated power almost coincides with the maximum electron temperature. The smaller values of the radiated power at both the 0 and 10 ms points (zero current) should be considered with care. Inaccuracies in the calculation may result from the comparison of a small radiated power with a small power dissipation.
Table 9.2: A comparison of the time-averaged dissipated power and time-averaged radiated power given in literature, and as found by our model, for a RMS current of 425 mA. The significance of the agreement is not clear, since the model (a simplified version of the steady-state model) finds a very different current for the steady-state case (see table 9.1).

9.4 Conclusions

As shown in the previous section, we are able to do a time-dependent simulation of the positive column of a fluorescent lamp. A noteworthy conclusion is that the radiation production at time-averaged power does not seem to be significantly different from the time-averaged radiation production. The slight difference that exists (74% vs. 72%) may well be due to the fact that a constant escape factor for the 185 nm line is not appropriate for time-dependent behaviour. The alternating recombination and excitation phases probably require an escape factor that changes in time.

However, the electrical conductivity seems to be strongly affected by the time-dependence. The time-averaged current through the plasma was about 22% higher than the current in the steady-state case for the same dissipated power. In addition, this conductivity was strongly affected by the chemistry, as could be seen from the reduction in current for the various simpler steady-state models. Because of these influences, it would be interesting to see how a time-dependent version of the full steady-state model presented in chapter 8 behaves. Unfortunately, this is not feasible with the computational hardware that is currently available. As an alternative, one could consider using multiple computers in parallel to calculate the radiation transport and/or enhancing the simple approximation by artificially increasing the mercury ionisation rate, in order to account for the Penning effect while not taking argon excitation into account.

Finally, it may be concluded that time-dependent behaviour such as the cycle of recombination and ionisation/re-ignition has a significant effect on at least some of the averaged discharge parameters. For this reason, it can add to the understanding of the physical phenomena involved in fluorescent lamp discharges.
Chapter 10

General conclusions

A model of the positive column as found in fluorescent lamps was constructed using the existing modular plasma simulation framework PLASIMO. In order to do this, the framework has been extended with aspects important for describing the behaviour of such discharges. However, most of these extensions are also useful for simulations of other types of plasmas. In chapter 3, it was shown how using a modern object oriented programming language such as C++ provides several benefits for modelling in general, and for the simulation of plasmas with many coupled interactions in particular. The addition of time-dependent discretisation to the framework, and the improvement of its modularity and extensibility were described in chapter 4. After that, chapter 5 presented a collisional radiative modelling program that can be used to accurately describe the various (de)excitation, ionisation and recombination processes that occur in a plasma. An extension to this model was made in chapter 6, where non-Maxwellian electron energy distribution functions were approximated by a three electron group model. Furthermore, the diffusion processes occurring in a mixture of many different species were described using a self-consistent diffusion model based on frictions (chapter 7). All these enhancements (with the obvious exception of time-dependent discretisation) were combined in chapter 8 to create a steady-state model of an argon-mercury positive column that gave very satisfactory results. Finally, a simplified version of the steady-state model was made time-dependent in chapter 9, showing the importance of understanding the dynamic behaviour.

The most important conclusions are now presented per chapter:

- Using the objected-oriented programming language C++ makes it easy to create modular and extensible modelling codes. At the same time, models written in this language are not significantly slower than ones written in more traditional languages such as Fortran. In addition, C++ makes it relatively easy to allow users to construct models at runtime, allowing for more user-friendly modelling software.

  Chapter 3.

- The existing PLASIMO modelling framework has been extended to include time-dependent discretisation. In addition, it has been made more mod-
Chapter 10: General conclusions

ular. Both enhancements improve the applicability of the framework to a variety of discharges.

Chapter 4.

- The use of collisional radiative models makes it possible to introduce a hierarchy of species. The separation of these species into local chemistry dominated and transport sensitive ones allows complex (de)excitation, recombination and ionisation processes to be described by tracking only the densities of a limited amount of species.

Chapter 5.

- Deviations from Maxwellian electron energy distribution functions (EEDFs) that occur in low-pressure argon-mercury fluorescent lamp plasmas can be described satisfactorily by using a three electron-group model that considers the production and destruction of higher-energy electrons. Assuming a Maxwellian EEDF for these plasmas leads to a serious overestimation of the buffer gas ionisation.

Chapter 6.

- The use of a self-consistent diffusion model as opposed to the simple Fick ambipolar diffusion approach is necessary to describe medium-pressure mixtures of multiple atoms, especially ones that contain several ion species in comparable quantities.

Chapter 7.

- In order to accurately describe the chemistry of the argon-mercury mixture in low-pressure fluorescent lamps, it is necessary to consider both a non-Maxwellian EEDF and quenching due to the so-called Penning effect. Otherwise, the ionisation of argon will be exaggerated and that of mercury underestimated, resulting in, among other things, a too small electrical conductivity. A fluorescent lamp model that includes these effects produced satisfactory results with respect to the conductivity, current and radiated power.

Chapter 8.

- A time-dependent simulation of an argon-mercury positive column showed that dynamic effects have an important influence on the electrical characteristics of the plasma. Significant differences in (averaged) current and/or total dissipated power exist between a steady-state solution and a time-dependent solution of a 50 Hz discharge, due to recombination and re-ignition effects.

Chapter 9.
Bibliography


[38] http://www.netlib.org/.


Summary

Fluorescent lamps are the most important plasma-based light sources and the most common man-made plasmas. In order to improve these lamps or adapt them to other uses, a better understanding of the physical mechanisms involved in such lamps is necessary. For the purpose of extending our insight into this kind of plasma, we decided to create a model of an argon-mercury positive column discharge as typically found in low-pressure fluorescent lamps. In addition, the construction of this model provided many opportunities to make extensions to the existing PLASIMO framework, both in the field of numerical model design and in physics, that are also useful for simulating other types of discharges.

The most significant contributions in the field of model design were the implementation of a time-dependent discretisation and the creation of a system for passing data between the various parts of the model. The first should allow us to model a wide variety of dynamic behaviour, while the latter makes a very modular construction of the model possible and allows runtime changes in its configuration.

In order to describe the many (de)excitation, ionisation and recombination mechanisms that occur in a composition such as the argon-mercury mixture in a fluorescent lamp, one can make use of a collisional radiative model. By distinguishing between on one hand the many states solely determined by local chemistry, and on the other hand the few species that are transport sensitive, such a model allows taking (the effect of) an entire atomic state distribution into account while only explicitly tracking the few transport sensitive levels. This greatly reduces the computational effort necessary to consider a complex composition.

An important factor in the electron excitation processes that occur in the positive column of a low-pressure fluorescent lamp is the electron energy distribution function (EEDF). Due to the low electron density, this EEDF is not Maxwellian. Fortunately, this can be described efficiently by using a three electron-group model, revealing that assuming a Maxwellian EEDF may lead to a severe overestimation of the argon ionisation rate.

The diffusion processes that occur in an argon-mercury positive column cannot always be accurately described by the traditional (ambipolar) Fick diffusion law. For this reason, we developed a self-consistent diffusion model based on frictions, that also takes ambipolar diffusion, thermal diffusion, the external electric field and the thermophoretic force into account.

All these mechanisms were combined with other parts of the modelling toolkit
in a steady-state model of an argon-mercury positive column in a low-pressure fluorescent lamp. This model confirmed the importance of the non-Maxwellian EEDF and the so-called Penning effect. The results for the electrical conductivity and radiated power were in good agreement with experimental observations.

Finally, a time-dependent model of the positive column was created, based on the steady-state model. It showed that consideration of the dynamic behaviour of such plasmas is important, and may have a significant influence on the results.
Samenvatting

Fluorescentie lampen zijn de belangrijkste plasma lichtbronnen en de meest voor-
komende plasma’s die door mensen worden gemaakt. Om dit type lampen te
verbeteren of ze aan te passen voor andere toepassingen, is een beter begrip van
de fysische mechanismen die een rol spelen in zulke lampen vereist. Met het doel
meer inzicht te verkrijgen in dit soort plasma’s, hebben we een model van een
argon-kwik positieve kolom ontlanding gemaakt, zoals die voorkomt in lage druk
fluorescentie lampen. Bovendien biedt het bouwen van een dergelijk model veel
mogelijkheden om het bestaande PLASIMO systeem uit te breiden, zowel op het
gebied van numerieke technieken als op het gebied van natuurkundige toepassin-
gen, waarbij beide soorten van uitbreidingen ook bruikbaar bleken te zijn voor
het simuleren van andere soorten van ontlanding.

De belangrijkste bijdragen op het gebied van model-ontwerp zijn de imple-
mentatie van tijdsafhankelijke discretisatie en het maken van een systeem voor
het uitwisselen van gegevens tussen de verschillende onderdelen van een model.
De eerste bijdrage maakt het mogelijk om een grote diversiteit aan dynamische
processen te simuleren, terwijl de tweede bijdrage een zeer modulaire opzet van
modellen toelaat.

Een botsings-stralings model kan worden gebruikt om de grote verscheiden-
heid aan (de)excitatie, ionisatie en recombinatie processen in een mengsel, zoals
het argon-kwik mengsel in een fluorescentie lamp, te beschrijven. In zo’n model
can onderscheid worden gemaakt tussen aan de ene kant toestanden waarvan de
dichtheden slechts worden bepaald door lokale chemische processen, en aan de
andere kant de enkele deeltjes die ook worden beïnvloed door transport. Hier-
door is het mogelijk om (de effecten van) een distributie van atmosfere toestanden
behandelen, terwijl slechts enkele transport-gevoelige niveaus hoeven te worden
beschouwd. Dit leidt tot een belangrijke reductie van de hoeveelheid benodigd
rekenwerk voor ingewikkelde mengel-samenstellingen.

Een belangrijke factor in de elektronen excitatie-processen die een rol spelen
in de positieve kolom van een lage druk fluorescentie lamp is de elektronen energie
distributie functie (EEDF). Door de lage elektronen dichtheid is deze EEDF niet-
Maxwells. Gelukkig kan deze situatie doeltreffend worden beschreven met behulp
van een drie elektronen-groepen model, dat aantoont dat de aanname van een
Maxwellse EEDF kan leiden tot een ernstige overschatting van de argon ionisatie.

De diffusie in een argon-kwik positieve kolom kan niet altijd nauwkeurig wor-
den beschreven met de traditionele (ambipolaire) Fick diffusie wet. Daarom
hebben we een zelf-consistent diffusie model ontwikkeld dat is gebaseerd op fric-
ties en tevens ambipolaire diffusie, thermische diffusie, een extern elektrisch veld en thermoforese beschouwt.

Al deze mechanismen worden samen met andere onderdelen van PLASIMO gecombineerd tot een tijdsonafhankelijk model voor een argon-kwik positieve kolom in een lage druk fluorescentie-lamp. Dit model bevestigt het belang van een niet-Maxwellse EEDF en het Penning-effect. De resultaten voor de elektrische geleidbaarheid en uitgestraald vermogen zijn in goede overeenstemming met experimentele gegevens. Tenslotte is er een tijdsonafhankelijk model geconstrueerd, gebaseerd op een gesimplificeerde versie van het tijdsonafhankelijke argon-kwik model. Dit laat zien dat het beschouwen van dynamisch gedrag van dergelijke plasma’s van belang is, en dat het een belangrijke invloed kan hebben op de resultaten.
Dankwoord

Dit proefschrift is het resultaat van vier jaar werken aan het simuleren van plasma’s in het modelleerteam van de groepen ETP en EPG. Op de eerste plaats wil ik graag mijn copromotor Joost van der Mullen bedanken, zonder wiens goede combinatie van begeleiden en sturen, mij m’n gang laten gaan als ik dat nodig had, stimuleren, en kritische vragen stellen, ik nooit zover zou zijn gekomen.

Verder ben ik veel dank verschuldigd aan mijn eerste promotor Daan Schram, voor zijn soms verrassende, en altijd interessante inzichten in de natuurkunde, en de opbouwende kritiek en nuttige suggesties bij het schrijven van mijn proefschrift.

Uiteraard zijn vier van zulke jaren niet vol te houden zonder een prettige omgeving. Voor de gezelligheid en werksfeer wil ik in de eerste plaats de teamleden van het eerste uur, Ger Janssen, Jan van Dijk, Harm van der Heijden, Colin Johnston en Kurt Garloff bedanken. Verder natuurlijk de studenten Astrid Stijfs en Michiel van den Donker, en de twee laatste aanwinsten voor het team, Wouter Brok en Bart Broks. En tenslotte de vele andere leden van de groepen ETP en EPG voor de prettige samenwerking en goede sfeer.

Voor het plezier buiten de werksfeer moet ik talloze mensen bedanken. Sommigen heb ik hierboven al genoemd, anderen die ik nog speciaal wil vermelden zijn mijn ouders, Han en natuurlijk de velen in en rond Nwyvre.

_Bart Hartgers, maart 2003._

About the author

Bart Hartgers was born in Amsterdam on March 31 1973. He lived in Monnickendam until 1980, when he moved to Terneuzen. In 1991 he started studying applied physics at the Eindhoven University of Technology, where he graduated in 1997. From 1999 until 2003 he worked as a Ph.D. student in the group Equilibrium and Transport of Plasmas of the Eindhoven University of Technology.