Charles University Prague
Faculty of Mathematics and Physics

MASTER'S THESIS

Ondřej Maršálek

Advanced Techniques of Computer Modelling in Plasma Physics

Institute of Theoretical Physics

Supervisor: Prof. RNDr. Rudolf Hrach, DrSc.

Study programme: Theoretical Physics

I would like to thank my advisor, Professor Rudolf Hrach, for guidance, recommendations and scientific inspiration. I would also like to thank many friends and colleagues for stimulating discussions.

On a personal level, I would like to express my gratitude to my parents and family for patience and support and to Irene for more things than I could name here.

I hereby declare that this thesis is my own work, based on the sources and literature listed in the appended bibliography. I approve of making the thesis available for lending.

Prague, April 16, 2008 Ondřej Maršálek

CONTENTS

Title: Advanced Techniques of Computer Modelling in Plasma Physics
Author: Ondřej Maršálek
Department: Institute of Theoretical Physics
Supervisor: Prof. RNDr. Rudolf Hrach, DrSc.
Supervisor's e-mail address: Rudolf.Hrach@mff.cuni.cz

Abstract: This thesis is concerned with particle plasma simulations and focuses on the electrostatic part of the problem. The primary intended application is probe diagnostics in high-temperature plasma. Two two-dimensional plasma models with different methods of computing the electrostatic interaction are designed. These model are able to represent internal electrodes of arbitrary shapes in the two-dimensional geometry and include the influence of external magnetic field. The particle-in-cell method is a grid-based method and the obtained solution has the character of a mean-field approximation. The boundary integral/treecode method works without a grid and is more accurate, as it includes the short-range part of the electrostatic interaction as well. Both models are implemented and partially parallelized and the efficiency of these implementations is discussed. Results computed for various systems illustrate the features of the models. Differences between the two methods are discussed and directions of their potential future development are suggested.

Keywords: plasma, particle simulations, particle-in-cell, hierarchical algorithms, sheath

Název práce: Pokročilé techniky počítačového modelování ve fyzice plazmatu
Autor: Ondřej Maršálek
Katedra/ústav: Ústav teoretické fyziky
Vedoucí diplomové práce: Prof. RNDr. Rudolf Hrach, DrSc.
e-mail vedoucího: Rudolf.Hrach@mff.cuni.cz

Abstrakt: Tato diplomová práce se zabývá částicovými simulacemi plazmatu s důrazem na elektrostatickou část tohoto problému. Primární zamýšlená aplikace je sondová diagnostika ve vysokoteplotním plazmatu. Jsou navrženy dva dvourozměrné modely plazmatu s různými metodami výpočtu elektrostatického působení. Tyto modely jsou schopny popsat v dané dvourozměrné geometrii vnitřní elektrody libovolného tvaru a zahrnout vliv vnějšího magnetického pole. Metoda particle-in-cell používá mříž a získané řešení má charakter přiblížení středního pole. Metoda boundary integral-treecode nepoužívá mříž a je přesnější, protože zahrnuje krátkodosahovou část elektrostatické interakce. Oba modely jsou implementovány a částečně paralelizovány a je diskutována efektivita těchto implementací. Výsledky vypočtené pro různé systémy ilustrují vlastnosti obou modelů. Jsou diskutovány rozdíly mezi oběma metodami a navrženy možné směry jejich dalšího rozvoje.

Klíčová slova: plazma, částicové simulace, particle-in-cell, hierarchické algoritmy, stínící vrstva

CHAPTER
ONE

Introduction

Computer modelling and simulations have become a complement to theory as well as experiment that can provide new insights or data not available otherwise. This is especially true for simulations at the atomic level. These can often treat processes that are below the spatial or temporal resolution of experimental methods, while being sufficiently flexible to describe complex phenomena that are out of reach of analytical models.

This is also the case for plasma probe diagnostics. Traditionally used analytical models rely on approximations and only work for simple geometries and a limited range of parameters. Experimental results can not say much about the details of processes in the vicinity of probes in plasma. Particle simulations have the potential to assist in the interpretation of experimental data by providing detailed information about physical quantities and processes at the interface between plasma and immersed solids. The improvement of such methods motivates this thesis. An especially challenging problem in plasma simulations is the computation of the long-range electrostatic interaction. This thesis focuses on that problem and its solution in a plasma simulation using two different approaches — the grid-based particle-in-cell method and the grid-free boundary integral/treecode method.

We will start the discourse by reviewing well-known facts and published material relevant to the rest of the thesis. The areas of interest will be plasma physics, particle simulations generally and plasma simulations with emphasis on the particle approach. As a preparation for the formulation of two computer models, several theoretical topics are explored. Next, the two models are formulated and their implementation is described. Finally, some examples of results obtained using these models are presented. The results from both methods are compared and the implications of their differences are discussed.

CHAPTER
TWO

Literature review

This chapter will present a review of the current knowledge in plasma physics and simulations with emphasis on the particle approach. It is not intended to be a complete review but rather a choice of topics from standard textbooks, lectures and published articles that are necessary to understand the rest of the thesis and to put it in proper context. We shall start with the fundamentals of plasma physics and then move on to the options we have for representing and simulating plasma and its physical properties in a computer.

2.1 Plasma Physics

The standard definition says that plasma is a quasi-neutral ionized gas that demonstrates collective behaviour. Quasi-neutrality is the requirement that there is no substantial space charge over macroscopic length scales. Collective behaviour is implied by the long-range character of the electrostatic interaction. It is especially this property that gives plasma its unique features.

While this description and set of requirements can be further refined by imposing constraints on relations between physical quantities, this is still very general and therefore covers a wide range of parameters.

The main references used for this section are [Che74] and [Krl05] and will not be further explicitly mentioned.

2.1.1 Basic properties

Generally speaking, there can be several types of charged particles in a plasma. These are usually electrons and one or more types of ions of various charges, but there are other possibilities, for example an electron-positron plasma or dusty plasma. Here we will speak only about plasmas obtained by ionizing gases. When we use n_α for number densities of the individual components and q_α for the charge of one particle of the appropriate component we can write the quasi-neutrality condition as

$$\sum_\alpha n_\alpha q_\alpha \approx 0. \tag{2.1}$$

For plasma in or near thermodynamic equilibrium we can use the usual concept of temperature and expect the Maxwell distribution of velocities. In certain situations there can be an independent temperature T_α for each of the components. This is called non-isothermal plasma in contrast with isothermal plasma, where the temperatures of all the components are the same.

As is obvious from the definition, there are mobile charge carriers in plasma and they will move to compensate any space charges or external potentials. This has two immediate consequences — plasma oscillations and Debye shielding.

The Debye length is a typical distance at which we can expect applied potentials to be shielded by plasma. Its derivation can be found in any standard plasma textbook and if we neglect the ionic contribution (which is usually the case) its value is

$$\lambda_D = \sqrt{\frac{\varepsilon_0 k_B T_e}{n_e e}}. \tag{2.2}$$

A related quantity is the number of particles in the Debye sphere, the so called plasma parameter

$$N_D = \frac{4}{3}\pi n \lambda_D^3. \tag{2.3}$$

Any fluctuation of space charge will induce movement of charged particles compensating this space charge and producing oscillations with a characteristic frequency called the plasma frequency. Strictly speaking, these are different for individual components but the highest frequency, which is produced by electrons, is usually of interest and is given by

$$\omega_{pe} = \sqrt{\frac{n_e e^2}{m_e \varepsilon_0}}. \tag{2.4}$$

Plasma can be ionized partially or completely. This is characterized by the degree of ionization — the proportion of atoms or molecules that are ionized. We can also distinguish between collisional plasmas — their behaviour is strongly influenced by collisions with neutral particles — and collisionless plasmas — the influence of collisions is not significant. This difference is determined by the density of the neutral gas. There can be several types of collisions, elastic as well as inelastic, but we will not go into more details because it is not necessary for the purpose of this thesis.

We can now put these basic properties together and impose restrictions that guarantee that an ionized gas behaves as plasma under the above mentioned definition. We require that the size of the system L is substantially greater than the Debye length, that the number of particles in the Debye sphere N_D is much greater than one and that the mean time between collisions is smaller than the period of plasma oscillations. This ensures quasi-neutrality and collective behaviour.

$$\lambda_D \ll L \tag{2.5}$$

$$N_D \gg 1 \tag{2.6}$$

$$\omega_{pe}\tau > 1 \tag{2.7}$$

As was mentioned above, plasma occurs in a wide range of parameters. Number densities can range from 10^6 m^{-3} to 10^{24} m^{-3}, temperatures from 10^{-2} eV to 10^4 eV and ionization degrees from about 10^{-6} or 10^{-7} to 1. There are, however, two distinct categories of plasma, commonly called low-temperature and high-temperature. The term low-temperature plasma is usually used for plasma that is non-isothermal, with ions at room temperature and electrons at roughly $T = 1$ eV, and has a very low degree of ionization. The term high-temperature plasma, on the other hand, is used for plasma that is completely ionized and isothermal, with temperature ranging from 10 eV to 10^4 eV. This kind of plasma is typical for controlled fusion experiments.

2.1.2 Single particle motion

Although plasma generates its own electric as well as magnetic field and therefore requires a self-consistent approach, it is useful to examine the motion of single charged particles in external electromagnetic fields. Several special cases can be solved analytically, for a general electromagnetic field we have to resort to numerical solution of the equations of motion.

The force exerted by electromagnetic field on charged particles is called the Lorentz force and has the form

$$\mathbf{F}_L = q\left(\mathbf{E} + \mathbf{v} \times \mathbf{B}\right). \tag{2.8}$$

The trajectory of a charged particle under the influence of $\mathbf{F}_L$ can be calculated by solving the classical equations of motion. We will now review the most important cases, the derivation of these can again be found in most plasma textbooks.

As can be seen from the formula (2.8), the electric field alone will cause charged particles to accelerate in the direction of the field lines. The magnetic field alone, on the other hand, will change only the direction of the velocity, not the magnitude, as can be seen from the vector product — the magnetic force is always perpendicular to the velocity. In a homogeneous magnetic field, a charged particle will move uninfluenced along the magnetic field lines and will perform harmonic oscillations in the two directions perpendicular to the field lines, resulting in circular or helical motion, so-called gyration. The frequency of this motion is called the cyclotron frequency and can be expressed as

$$\omega_c = \frac{|q|\,B}{m}. \tag{2.9}$$

The radius of gyration, sometimes called the Larmor radius, is then

$$r_g = \frac{v_\perp}{\omega_c} = \frac{m v_\perp}{|q|\,B}, \tag{2.10}$$

where $v_\perp$ is the magnitude of the velocity projected into the plane perpendicular to the magnetic field.

When there is an additional homogeneous electric field present, it acts independently in the direction along the magnetic field lines and creates a drift of the centre of gyration in the direction perpendicular to both the electric and the magnetic field.

In more general fields, other drifts can arise, but we will not go through the details here, as they will not be important in this work. The general character of the motion remains the same, its main feature being the greatly decreased mobility in directions perpendicular to magnetic field lines. This is also the main idea of magnetic confinement in experiments of the tokamak type as well as others.

2.1.3 Kinetic description

Most plasma systems of interest are out of thermodynamic equilibrium. A useful statistical description of such a system at the level of classical mechanics is the probability distribution $f_\alpha(\mathbf{x}, \mathbf{v}, t)$ (sometimes called the Boltzmann distribution), where the index α again denotes the various components. This is a distribution on the one-particle phase space that is also time-dependent. It can be obtained from the distribution function on the full N-particle phase space by integrating over the appropriate coordinates. In the following text we will omit the arguments of f and the index α.

It can be shown [Lip07] that the Boltzmann distribution satisfies a continuity equation on the one-particle phase space that has the form

$$\frac{\mathrm{d}f}{\mathrm{d}t} = \frac{\partial f}{\partial t} + \mathbf{v} \cdot \nabla_{\mathbf{x}} f + \frac{\mathbf{F}}{m} \cdot \nabla_{\mathbf{v}} f = 0, \tag{2.11}$$

where $\mathbf{F}$ includes all external as well as internal forces. In a gas, however, the forces exerted during collisions of neutral particles are huge compared to other forces and also act on very short space and time scales. This allows us to treat particle collisions as instantaneous momentum transfer, which leads to Boltzmann's formulation of his famous equation

$$\frac{\partial f}{\partial t} + \mathbf{v} \cdot \nabla_{\mathbf{x}} f + \frac{\mathbf{F}}{m} \cdot \nabla_{\mathbf{v}} f = \left(\frac{\delta f}{\delta t}\right)_c, \tag{2.12}$$

where the right-hand side is the so-called collision term and $\mathbf{F}$ now does not contain inter-particle forces due to collisions. The collision term can be expressed as an integral over momenta and collision probabilities and is usually the difficult part of solving (2.12) for f.

We are interested in applying this approach to plasma, where particle interactions cannot be treated as single point collisions. One way to do that is to drop the collision term and include a mean electromagnetic force from the particles in the force term. The result is called the Vlasov equation:

$$\frac{\partial f}{\partial t} + \mathbf{v} \cdot \nabla_{\mathbf{x}} f + \frac{q}{m} \left(\mathbf{E} + \mathbf{v} \times \mathbf{B} \right) \cdot \nabla_{\mathbf{v}} f = 0. \tag{2.13}$$

Together with Maxwell's equations, this represents a self-consistent system of equations. Charge and current densities, to be used as sources in Maxwell's equations, can be obtained from f by integrating over velocities. Maxwell's equations, in turn, provide a way to calculate the force term in the Vlasov equation. This is an approximation that can be used to obtain some interesting results in plasma theory.

There are also ways to approximate the collision term in plasma but it is beyond the scope of this work to go into the details.

2.1.4 Fluid equations

For systems in or near thermodynamic equilibrium it is possible to go from the statistics given by the Boltzmann equation to a continuum description. To do this, we neglect the velocity distribution and only use mean velocity while assuming the Maxwell distribution. The corresponding equations are called fluid equations because each component is represented by a continuous fluid. These equations can be obtained as moments of the Boltzmann equation with respect to velocity. They will not be used in this thesis, so we present only a brief overview here. In the following formulae we will omit the index α for individual components that should be present for every quantity.

First, we define averaged quantities by the expressions

$$n(\mathbf{x}, t) = \int f(\mathbf{x}, \mathbf{v}, t) \mathrm{d}\mathbf{v}, \tag{2.14}$$

$$\bar{\mathbf{v}}(\mathbf{x}, t) = \int \mathbf{v} f(\mathbf{x}, \mathbf{v}, t) \mathrm{d}\mathbf{v}, \tag{2.15}$$

$$\mathbf{P}_c(\mathbf{x}, t) = \int m\mathbf{v} \left(\frac{\delta f(\mathbf{x}, \mathbf{v}, t)}{\delta t} \right)_c \mathrm{d}\mathbf{v}, \tag{2.16}$$

$$\mathbb{P}(\mathbf{x}, t) = mn \int f(\mathbf{x}, \mathbf{v}, t)(\mathbf{v} - \bar{\mathbf{v}}(\mathbf{x}, t))(\mathbf{v} - \bar{\mathbf{v}}(\mathbf{x}, t)) \mathrm{d}\mathbf{v}, \tag{2.17}$$

where n is the particle density, $\bar{\mathbf{v}}$ is the mean velocity, $\mathbf{P}_c$ is the transfer of momentum through collisions and $\mathbb{P}$ is the stress tensor.

By integrating the Boltzmann equation over velocities, one obtains the continuity equation for particle density

$$\frac{\partial n}{\partial t} + \nabla \cdot n\bar{\mathbf{v}} = 0. \tag{2.18}$$

This equation expresses the conservation of mass (of the same type of particles). For situations with processes that change the number of particles of certain type — ionization, for example — there would be a corresponding right-hand side.

Multiplying the whole Boltzmann equation by $m\mathbf{v}$ and integrating over velocities, we get, after some work, the equation of motion for the fluid

$$mn\left(\frac{\partial \bar{\mathbf{v}}}{\partial t} + \bar{\mathbf{v}} \cdot \nabla \bar{\mathbf{v}}\right) = qn\left(\mathbf{E} + \bar{\mathbf{v}} \times \mathbf{B}\right) - \nabla \cdot \mathbb{P} + \mathbf{P}_c, \tag{2.19}$$

which is the expression of the conservation of momentum, if we consider the equations for all the components as well as the equations for the electromagnetic field.

Multiplying the Boltzmann equation by $mv^2/2$ and integrating over velocities would give us the equation of conservation of energy. Another option is to consider the equation of state relating pressure and density instead.

Together with Maxwell's equations this forms a full set of partial differential equations for the quantities of interest. Compared to the previous case, we have, however, lost the resolution of the distribution of velocities.

2.1.5 Plasma probe diagnostics

The primary motivation of this work is plasma probe diagnostics using the so-called Langmuir probe [LS24]. Generally speaking, this involves one or more metal probes inserted into a plasma with either a constant or varying potential. Measuring the currents through these probes then allows one to determine various plasma properties, for example particle temperatures and densities or plasma potential [Bro74]. This is potentially a broad area of physics in its own right and here, we will present only the very basics to make the motivation clear.

There are many possible probe geometries and configurations. We will talk only about the single-probe method, as the main ideas in other cases are similar. A single probe, usually a thin wire (called a cylindrical probe), is inserted into the boundary region of a plasma. The current through probe is measured for a range of potentials, obtaining a current-voltage characteristic. For highly negative potentials (relative to the plasma potential at the point of the probe), the ion current is dominant. Ions (we consider only the case of positive ions here) are accelerated towards the electrode and electrons are repelled, unable to reach the electrode through the potential barrier. At a certain negative potential, called the floating potential, the ion and electron currents reach the same value. The reason that the floating potential is negative and not zero is the difference in the energies of the electrons and ions given by their different temperatures (in non-isothermal plasma). Electrons are as likely as ions to get to the probe, even if there is a potential bias against them. With rising probe potential, ions are repelled and electrons attracted, creating a sheath of accelerated electrons. The associated space charge shields the probe potential, which is in agreement with the elementary idea of Debye shielding. An example of a current-voltage characteristic is shown in Figure 2.1.

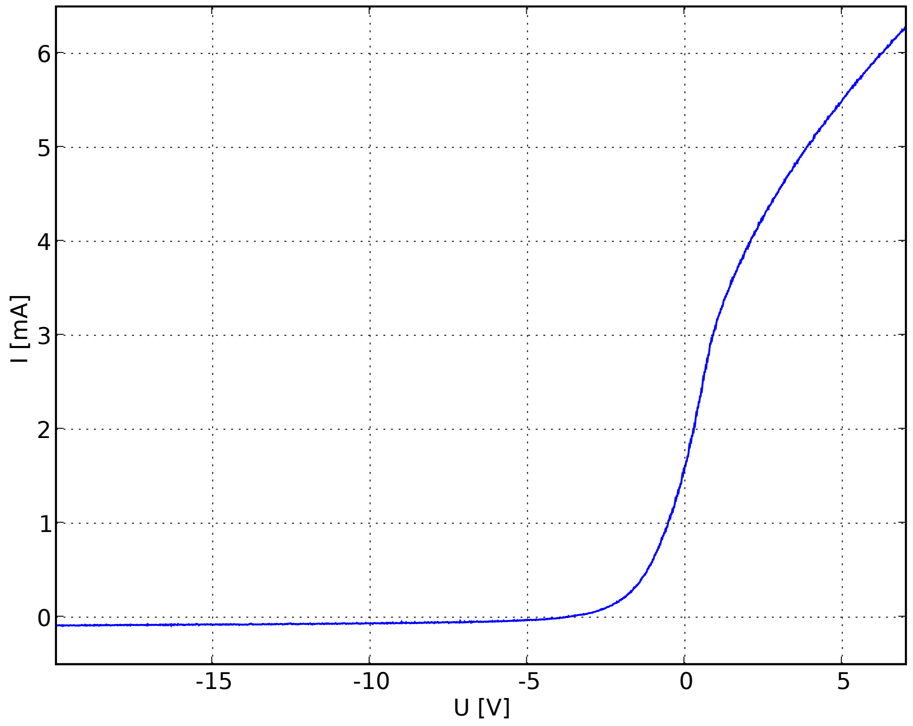


Figure 2.1 The figure shows an example of an experimental current-voltage characteristic for a cylindrical probe of radius $r = 22.5\,\mu$m, measured in argon in a pulse magnetron. Electron density was roughly $n_e = 10^{17}$ m^{-3}, electron temperature $T_e = 2$ eV, pressure $p = 10$ Pa. The data were kindly provided by Jan Klusoň.

This description is very rough and the details depend on many things. The interpretation of the current-voltage characteristic has traditionally been dependent on analytical models of the vicinity of the probe. The demand for higher accuracy as well as complicated probe geometries in high-temperature plasma applications make this a good target for computer simulations. The basic idea is to substitute the analytical model with a simulation that would relate the probe potential to the probe current. An additional advantage is a detailed view of the vicinity of the probe (or probes) and access to information that is not available to experiment.

2.2 Molecular dynamics

The molecular dynamics method is a general computational approach to the classical many-body problem. It can be applied to various systems in physics from atomic resolution fluids and solids to astrophysics. This section will present its general features, while those aspects specific to plasma simulations will be presented in the following section. A good introduction to the subject can be found for example in [AT89] or [FS01].

The main advantage of molecular dynamics is the atomic resolution of the system. This means that the method can in principle provide not only the resulting phenomena but also details of the processes by which these phenomena arise. Such information is often inaccessible to experiment and is potentially useful to better understand the phenomenon at hand.

2.2.1 Method overview

The basic idea of molecular dynamics is to represent the position and velocity of each particle in the system (usually an atom, ion or electron), consider the interactions of these particles and use classical mechanics to evolve the system in time from some initial condition, obtaining a trajectory in phase space.

Because of the number of degrees of freedom, the only useful possibility is a numerical approach. The system is discretized in time and propagated in finite steps that approximate a full trajectory. Depending on the specific system being studied, this trajectory can then be used to study the properties of a dynamic process or as a set of samples from an equilibrium distribution or a stationary state for obtaining averaged values. Any quantity that can be expressed as a function on phase space can be calculated from the trajectory. The most common ones include energy, density profiles, radial distribution functions, temperature, pressure and diffusion coefficients.

2.2.2 Time evolution

The governing equations of molecular dynamics are the classical equations of motion. The original formulation by Newton

$$m\frac{\mathrm{d}^2\mathbf{x}(t)}{\mathrm{d}t^2} = \mathbf{F} \tag{2.20}$$

can be equivalently expressed in other ways. Without going through all the details of theoretical mechanics we will show only the two most often used reformulations — the Lagrangian formalism and the Hamiltonian formalism. We work under the assumption that the interaction potential does not depend on time explicitly.

The Lagrangian is defined using the generalized coordinates q_j (which can describe the presence of constraints in the system) and their derivatives $\dot{q}_j$ as

$$L(q_j, \dot{q}_j) = T(\dot{q}_j) - V(q_j, \dot{q}_j), \tag{2.21}$$

where T is the kinetic energy, V is the interaction potential and j indexes all degrees of freedom. The corresponding equations of motion take the form

$$\frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial L}{\partial \dot{q}_j} - \frac{\partial L}{\partial q_j} = 0. \tag{2.22}$$

The Hamiltonian formulation can be derived from the previous one. Starting with the generalized momenta

$$p_j = \frac{\partial L}{\partial \dot{q}_j}, \tag{2.23}$$

we can obtain the Hamiltonian as

$$H(q_j, p_j) = \sum_j p_j \dot{q}_j - L(q_j, \dot{q}_j(p_j)), \tag{2.24}$$

where we assume that equations (2.23) can be inverted. The corresponding equations of motion are

$$\dot{p}_j = -\frac{\partial H}{q_j}, \tag{2.25}$$

$$\dot{q}_j = \frac{\partial H}{p_j}. \tag{2.26}$$

These formulations are essentially equivalent and provide (at least in principle) a way to propagate a classical system in time. A point in phase space $[\mathbf{x}(t_0), \mathbf{p}(t_0)]$ is mapped onto a point $[\mathbf{x}(t), \mathbf{p}(t)]$ at some later time $t > t_0$. We would like to be able to approximate the trajectory generated by this process numerically in discrete steps, getting from $[\mathbf{x}(t), \mathbf{p}(t)]$ to $[\mathbf{x}(t + \Delta t), \mathbf{p}(t + \Delta t)]$. Mathematically, this means solving the corresponding differential equations of motion numerically. There are several well known schemes of doing that (so-called propagators).

There are ways to include the influence of constraints but we will not discuss them here. We will, however, look at the case with a magnetic field present. This is a special case, because the magnetic force depends on velocity, which is a requirement that has to be built into the propagator.

The traditional way of deriving propagators is by finite differencing. There is also an approach that is more systematic and produces strictly symplectic propagators. This will be discussed later in 4.1.1 “Symplectic propagators” (page 29). The expansions needed to derive most of the following propagators can be found in C.1 “Numerical differentiation” (page 101).

We will now go through the main propagators, reference their origin and comment on their use. In all the cases without magnetic field, $\mathbf{F}(t)$ is a shorthand for $\mathbf{F}(\mathbf{x}(t))$, as the force depends only on the position.

Verlet

The classic Verlet propagator [Ver67] does not contain explicit velocities. If needed, they can be computed later using the expression below. There is a class of propagators that produce trajectories identical to those of the Verlet propagator. We will mention that for each of them.

The formula to get from t to $t + \Delta t$ is

$$\mathbf{x}(t + \Delta t) = 2\mathbf{x}(t) - \mathbf{x}(t - \Delta t) + \frac{\mathbf{F}(t)}{m}\Delta t^2. \tag{2.27}$$

The expression for velocities is the usual derivative by finite differences. Notice that we also need the position one step ahead.

$$\mathbf{v}(t) = \frac{\mathbf{x}(t + \Delta t) - \mathbf{x}(t - \Delta t)}{2\Delta t} \tag{2.28}$$

Velocity Verlet

The velocity Verlet propagator [SABW82] is an improvement of the previous one for cases where we need velocities at the same times as positions. It belongs to the Verlet class of propagators, which can be verified by reproducing the original expression by algebraic manipulations.

$$\mathbf{x}(t + \Delta t) = \mathbf{x}(t) + \mathbf{v}(t)\Delta t + \frac{\mathbf{F}(t)}{2m}\Delta t^2 \tag{2.29}$$

$$\mathbf{v}(t + \Delta t) = \mathbf{v}(t) + \frac{\mathbf{F}(t + \Delta t) + \mathbf{F}(t)}{2m}\Delta t \tag{2.30}$$

Leap-frog

The leap-frog propagator uses velocities at times shifted by half a time step, hence the name. The book [AT89] cites the article [Hoc70] as its origin, but the earliest use of this propagator known to the author is in [FLS63]. It is very simple to implement and therefore used quite a lot, if the shift of velocities is not a problem. This propagator belongs to the Verlet class.

$$\mathbf{x}(t + \Delta t) = \mathbf{x}(t) + \mathbf{v}(t + \Delta t/2)\Delta t \tag{2.31}$$

$$\mathbf{v}(t + 3\Delta t/2) = \mathbf{v}(t + \Delta t/2) + \frac{\mathbf{F}(t + \Delta t)}{m}\Delta t \tag{2.32}$$

Beeman

The Beeman propagator [Bee76] is another example of a Verlet class propagator. It produces the same trajectory and slightly more precise velocities. The price for this is a more complicated expression and the need to hold more values from past time steps in memory.

$$\mathbf{x}(t+\Delta t) = \mathbf{x}(t) + \mathbf{v}(t)\Delta t + \frac{\Delta t^2}{6m}\left[4\mathbf{F}(t) - \mathbf{F}(t-\Delta t)\right] \tag{2.33}$$

$$\mathbf{v}(t+\Delta t) = \mathbf{v}(t) + \frac{\Delta t}{6m}\left[2\mathbf{F}(t+\Delta t) + 5\mathbf{F}(t) - \mathbf{F}(t-\Delta t)\right] \tag{2.34}$$

Position Verlet

The position Verlet propagator [TBM92] has been found using the technique described in 4.1.1 “Symplectic propagators” (page 29). Somewhat confusingly, it does not belong to the Verlet class of propagators. It has been shown to produce better results than Verlet class propagators for longer time steps [TBM92].

$$\mathbf{v}(t+\Delta t) = \mathbf{v}(t) + \mathbf{F}(t)\Delta t\left[\mathbf{x}(t) + \frac{\Delta t}{2m}\mathbf{v}(t)\right] \tag{2.35}$$

$$\mathbf{x}(t+\Delta t) = \mathbf{x}(t) + \frac{\Delta t}{2}\left[\mathbf{v}(t) + \mathbf{v}(t+\Delta t)\right] \tag{2.36}$$

Velocity-corrected Verlet

This scheme [FS01] is not a true propagator, but rather a way to compute velocities for a known trajectory more precisely than with the Verlet or Beeman propagators. It is really just a numerical derivative of the trajectory that can be obtained as explained in C.1 “Numerical differentiation” (page 101). Notice that the formula

$$\mathbf{v}(t) = \frac{1}{12\Delta t}\left[\mathbf{x}(t-2\Delta t) - \mathbf{x}(t-\Delta t) + \mathbf{x}(t+\Delta t) - \mathbf{x}(t+2\Delta t)\right] \tag{2.37}$$

needs positions of two steps ahead.

HARHA

None of the above propagators include the influence of magnetic field. This can be achieved by a slight modification to the leap-frog propagator. The propagator now called “half acceleration, rotation, half acceleration” (HARHA) can be found in [BL91] but was used in various forms earlier. Here we present a slight modification that expresses the rotation of the velocity vector exactly, unlike the formulation in [BL91].

The formulae are as follows:

$$\mathbf{v}^{-}(t) = \mathbf{v}(t - \Delta t/2) + \frac{\mathbf{F}(t)}{m}\frac{\Delta t}{2}, \tag{2.38}$$

$$\mathbf{v}^{+}(t) = \mathbb{R}(\mathbf{B}) \cdot \mathbf{v}^{-}(t), \tag{2.39}$$

$$\mathbf{v}(t + \Delta t/2) = \mathbf{v}^{+}(t) + \frac{\mathbf{F}(t)}{m}\frac{\Delta t}{2}, \tag{2.40}$$

$$\mathbf{x}(t + \Delta t) = \mathbf{x}(t) + \mathbf{v}(t + \Delta t/2)\Delta t. \tag{2.41}$$

This is the same as the leap-frog propagator, only the velocity update is split into two stages with a rotation of the velocity vector inserted between them. The rotation is performed around the axis given by the direction of the magnetic field and the angle is $\omega_c \Delta t$, as expected. The construction of the matrix $\mathbb{R}$ can be found in Appendix D "Rotation of a vector in 3D" (page 110). A rigorous derivation of this propagator, showing its symplecticity, will be presented in 4.1.1 "Symplectic propagators" (page 29).

Gear propagators

The Gear propagators are a class of propagators that are sometimes used in molecular dynamics. They are not symplectic but are more accurate, especially for shorter time steps. The details can be found in [NKK03]. Figures 2.2 and 2.3 show Verlet and Gear (of three different orders) propagation of an orbiting planet. Figure 2.4 shows the energy during a simulation of a Lennard-Jones fluid.

Error terms

The error terms for most propagators can be expressed from the expansions from which they arise. These are, however, local errors only. Global errors should be used to determine the order of a propagator. It can be shown that propagators of the Verlet class are second order propagators, meaning that the global error in position is $\mathcal{O}((\Delta t)^2)$.

2.2.3 Force calculation

To propagate a system from one time to another, it is necessary to be able to calculate the forces acting on particles in a given state. These can be of two different kinds — external and internal forces. External forces are imposed on the system from outside and usually depend only on the position and velocity of each particle independently. Internal forces are forces from other particles in the system and generally they depend on the positions and velocities of all the particles in the system. These can be divided by the number of particles involved into two-body, three-body, etc. interactions, where the interactions of more than two particles account for polarization effects or bonds in molecules. Polarizability is often neglected, either because its influence is small or because of the computational demands. In the pair-additive approximation the force

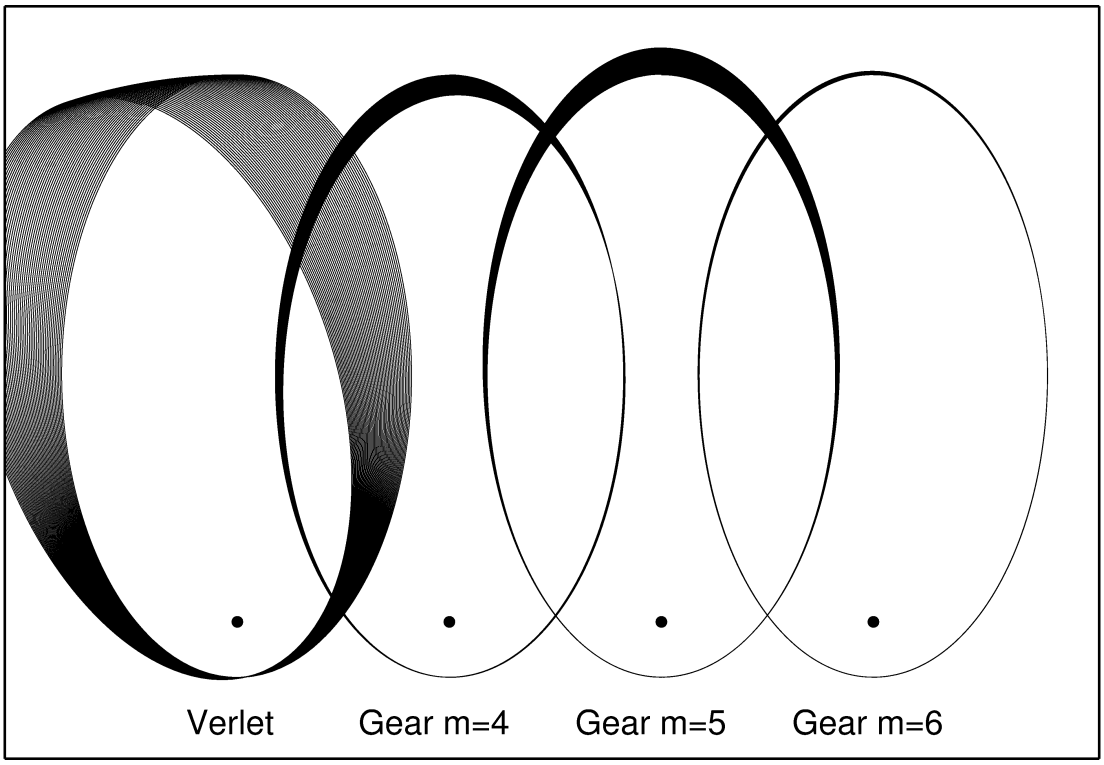


Figure 2.2 An orbiting planet propagated with the stated methods. Notice the qualitative difference. The Verlet propagator conserves the total energy (a consequence of its symplecticity) but causes precession. The Gear propagators do not conserve energy but the precession is significantly smaller. Reprinted from [NKK03] with permission.

on particle j is

$$\mathbf{F}_j = -\sum_{k \neq j} \nabla U(|\mathbf{r}_k - \mathbf{r}_j|), \tag{2.42}$$

where $\mathbf{r}_k$ is the position of particle k.

In the dynamics of complex molecular systems (for example biomolecules) the set of functions and parameters defining the internal forces is usually called a force field. In plasma physics the description of the interactions is more simple and the term "force field" is usually not used.

In its most straightforward formulation for pair interactions only, molecular dynamics has an asymptotic time complexity of $\mathcal{O}(N^2)$, because one has to consider the interaction of all particle pairs. This would, unfortunately, make any computations of useful system size prohibitively expensive in computer time. It is therefore necessary to find ways to improve the scaling of the method with system size. The available options depend on the nature of the interaction.

There are two distinct classes of particle interactions — short-range and long-range. Short-range interactions fulfill the following requirement for the interaction energy $U(r)$

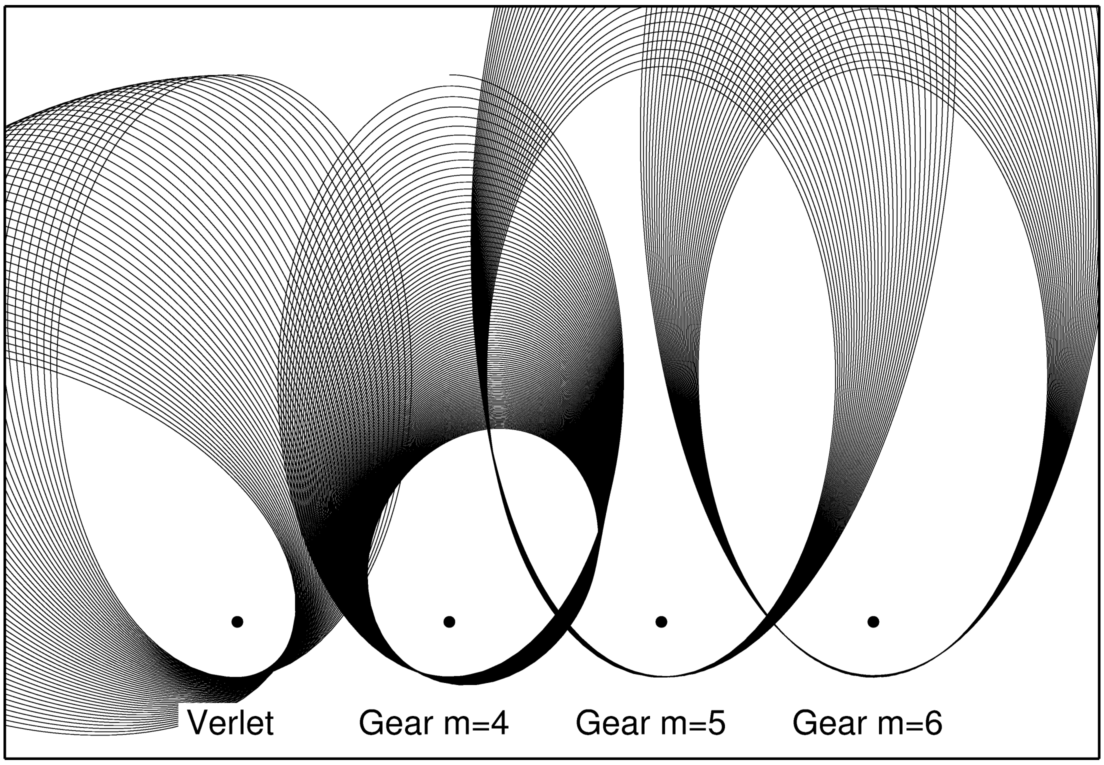


Figure 2.3 Same as Figure 2.2 but with double the time step. Reprinted from [NKK03] with permission.

$$\int_0^\infty U(r) r^2 \mathrm{d}r < \infty. \tag{2.43}$$

Furthermore, typical short-range interactions vanish much faster than is required by the above condition. The most important practical difference between these two is the possibility of truncating the interaction by using a cutoff distance in the case of short-range interactions.

For short-range interactions it is usually possible to modify the formula for energy or force in such a way that is has a strictly finite support and at the same time is not very different from the original one. This is done by setting the value of the energy to zero for $r > r_C$, where r_C is a chosen cutoff distance, and shifting the potential so that it is continuous:

$$U_{cutoff}(r) = \begin{cases} U(r) - U(r_C), & r < r_C \\ 0, & r > r_C \end{cases}. \tag{2.44}$$

Another option is to use a switching function for a smooth transition between the original function and the zero at $r > r_C$, which gives a continuous interaction potential as well as its derivative [NKK03]. This truncation procedure considerably limits the number of interactions for each particle and combined with other techniques allows to

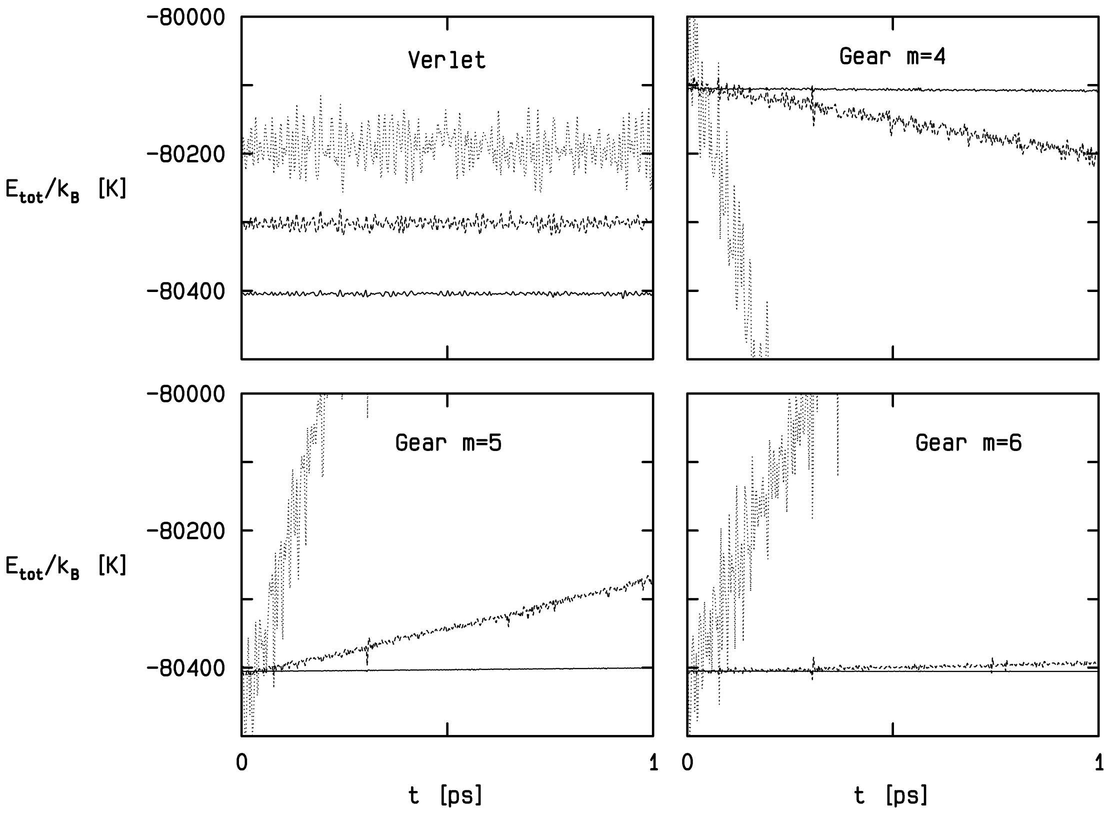


Figure 2.4 The total energy of a Lennard-Jones fluid using the stated propagators. There is an obvious drift for the Gear propagators, whereas the Verlet propagator seems to have a negligible drift but bigger fluctuations. Reprinted from [NKK03] with permission.

change the asymptotic time complexity from $\mathcal{O}(N^2)$ to $\mathcal{O}(N)$. This is, however, beyond the scope of this thesis.

Long-range interactions, on the other hand, cannot be changed in such a way, because the cutoff distance would be in most cases comparable to the system size. A usual representative of this type is the electrostatic interaction. Several approaches to this problem exist and they will be discussed in section 2.3 "Plasma simulations" (page 19).

2.2.4 Boundary conditions

So far, we have avoided the problem of the boundaries of the system. Most of the time, there are specific requirements on boundary conditions given by the physical properties of the system.

Open boundary conditions

When one considers only direct interactions within the system, there are no explicit boundary conditions. This is usually called open boundary conditions and it is the easiest situation to implement, as no extra effort is necessary. This type of boundary conditions is useful for example for simulating open clusters of atoms or galaxies in astrophysics. There are, however, finite size and surface effects that influence the system, which is not always desirable.

Periodic boundary conditions

In simulations where we are interested in bulk properties, we would like to minimize the surface effects. This is the primary motivation behind periodic boundary conditions. The system (usually a box) is replicated an infinite number of times in each direction, which produces a system of infinite size that is periodic, the period being the original box size. This is achieved in the following way: particles that exit the system on one side are inserted on the other side and for each particle only the interaction with one replica of each of the other particles is considered. This replica is chosen so that it is the one with minimum distance to the particle of interest. A simple illustration is provided in Figure 2.5. This should be further combined with a suitable spherical truncation procedure to avoid the influence of the shape of the box.

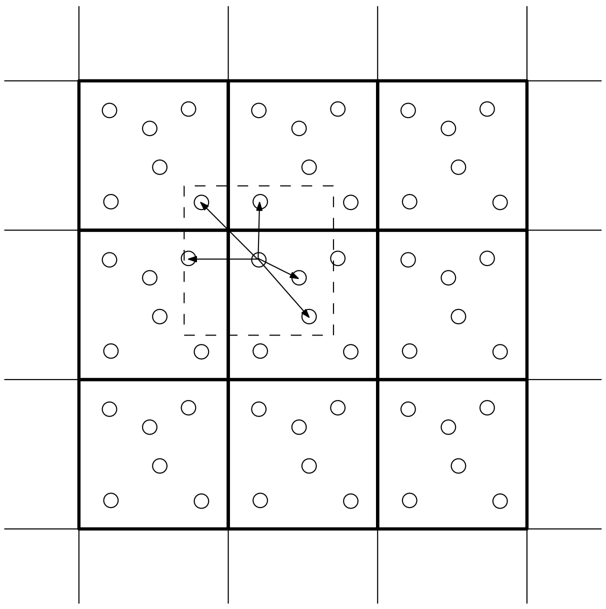


Figure 2.5 An illustration of the periodic boundary conditions in 2D, the nearest eight replicas of the system are drawn. Arrows show all the potential interactions for one of the particles, dashed box shows the effective limits of the system with respect to this particle.

There are no surface effects in such a system. On the other hand, there is artificial periodicity and one has to be careful about having a system large enough to be able to neglect its effects. It is also possible (and useful for certain applications) to have a system that is periodic in one or two dimensions and open in the rest. The most general three-dimensional periodic box is a triclinic cell, as implemented for example in the molecular dynamics package GROMACS [HKVL08].

Special boundary conditions

In certain cases, neither of the two above options is useful. There may be special requirements on the boundary conditions, usually an interface of some kind. This has to be, obviously, approached on a per case basis.

An example would be a simulation involving the surface of a solid, where several layers of atoms subject to normal molecular dynamics are usually terminated by a layer of atoms fixed in place or attached to fixed points via harmonic oscillators. An analytical correction for the rest of the solid below the last layer may also be added. Another notable example would be the interface with bulk plasma, but this will be treated in detail in the following parts of this thesis.

2.2.5 Pair correlation function

An important quantity that can be obtained from a particle simulation is the pair correlation function, sometimes also called radial distribution function. It is defined by the expression

$$g(r) = \frac{N(N-1)}{n^2 Q} \int \cdots \int \exp\left[-\beta U(\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_N\right] \mathrm{d}\mathbf{r}_3 \ldots \mathrm{d}\mathbf{r}_N, \tag{2.45}$$

where Q is the configuration integral

$$Q = \int \exp\left[-\beta U(\mathbf{r}^N)\right] \mathrm{d}\mathbf{r}^N. \tag{2.46}$$

It can be shown that various quantities can be expressed using the pair correlation function for an interaction potential that is a sum of pair interactions. For example, it is possible to express the interaction energy as

$$\langle U \rangle = 2\pi N n \int_0^\infty U(r) g(r) r^2 \mathrm{d}r. \tag{2.47}$$

Details can be found for example in [NKK03].

2.2.6 Further considerations

There are lots of other options and problems to consider in general molecular dynamics. One of the most important areas is the simulation of different thermodynamic ensembles. Without any modifications, molecular dynamics runs the microcanonical ensemble, sometimes called the NVE ensemble, as the conserved quantities are the number of particles, volume and total energy. Sometimes it is desirable to simulate a different ensemble for which several techniques exist. We will mention only the use of a thermostat, which can provide a canonical ensemble simulation (an NVT simulation).

Some thermostats provide an exact canonical ensemble while others provide a simulation at a set temperature that only approximates a canonical ensemble. The common feature of all of them is the fact that the time averaged temperature is equal to a chosen value, while the instantaneous value of temperature and the total energy fluctuate. The details of the fluctuations of the temperature depend on the specific thermostat used.

Perhaps the most simple approach is velocity rescaling. After a given number of simulation steps all the velocities are rescaled so that the kinetic temperature is equal to the desired temperature. This does not give en exact canonical ensemble and also distorts the dynamics of the systems quite a lot.

Other temperature control schemes are based on artificial degrees of freedom that are coupled to all the particles of the system. An example of this type is the Nosé-Hoover thermostat [Hoo85], which provides an exact canonical ensemble.

In this work we will use only the Andersen thermostat [And80]. The basic idea is the existence of a virtual bath of the desired temperature whose particles collide with those of the simulated system with a given collision frequency ν. The effect of such a collision is resetting of the velocity of the particle randomly from the Maxwell distribution. The collision frequency determines the strength of the coupling to the bath. This provides an exact canonical ensemble, as has been proven in the original article. The dynamics of the system can obviously be influenced considerably, on the other hand, this approach is very useful for quenching a system starting from an unphysical initial condition.

Other important topics include Verlet lists and constraints but these are not important in plasma simulations, therefore we will not discuss them here.

2.3 Plasma simulations

In this section we will go through the major possible approaches to plasma simulations while focusing on the particle-based methods. Generally, we are interested in the time evolution of a plasma system, either to get the time average of equilibrium or stationary state properties or to get the details of a dynamic process. The options we have roughly follow the different levels of plasma theory.

Some of the methods can be combined in so-called hybrid models. This usually involves using a more fundamental technique for parts of the system where higher resolution and better description is required and a higher level technique elsewhere. The decomposition can be done for example in real space or across plasma components. One example would be the use of a particle code for the vicinity of an electrode and a fluid code for larger areas of plasma near equilibrium with a boundary region between them. Another example is using a particle code for one particle type or the tail of its velocity distribution and a fluid code for the rest.

There is one approximation that is applicable to a wide range of situations independent of the specific method chosen. Often, we can neglect the terms in Maxwell's equations that account for wave propagation and keep only Poisson's equation of electrostatics. This is justified at timescales much larger than the typical wave period and for spatial features much larger than the typical wavelength of electromagnetic waves that could be present in the studied system. Unless stated otherwise, we will always work in the electrostatic approximation because it is sufficient for the systems we want to be able to treat.

2.3.1 Boltzmann equation modelling

A straightforward approach is to solve the partial differential equations at the level of the Boltzmann kinetic equation or the Vlasov equation numerically. The main problem here would be the number of dimensions. Although Maxwell's equations have 3+1 dimensions, the full Boltzmann equation has 6+1 dimensions, because the velocities are also independent variables. Solving this partial differential equation would be quite demanding computationally.

2.3.2 Fluid modelling

Another option is to take the fluid equations and treat them numerically, which is in principle similar to the previous case, except we do not have the velocity dimensions here. This is certainly an advantage, but it suffers from the limitation imposed by the equations themselves — we should expect poor results for situations with velocity distributions much different from the equilibrium Maxwell distribution.

This model must be accompanied by a suitable choice of boundary conditions for the partial differential equations. These depend on the specific system.

2.3.3 Particle modelling overview

The most detailed approach to plasma modelling is at the particle level. Classic texts in this area include [BL91] and [HE88]. The particles in the simulation can be interpreted in several different ways but the basic features are the same. In this type of models we run molecular dynamics of individual particles of the different plasma components. These can be regarded either as samples from the Boltzmann distribution $f(\mathbf{x}, \mathbf{v}, t)$ or as representations of real particles. For practical reasons, a coarse-graining of the model can be performed by grouping several particles into a superparticle or pseudoparticle with the mass and charge being integer multiples of the original ones. Conceptually, this is somewhere between the previous two possibilities.

As has been said, the basic idea is in fact the same in all cases. We need to compute the force acting on each of the particles in each time step, for example in the electrostatic limit by solving Poisson's equation. An external field can also be accounted for. The resulting force is then used to propagate the particles in time by Δt.

The self-consistent field computation can be regarded as a problem that is well defined by itself and is usually the most demanding part of the model. It will be discussed separately in more detail later. For the time being, let us assume that we have a way to compute this interaction.

Boundary conditions in particle modelling are a little more complicated than in the fluid case. There have to be the usual boundary conditions for Maxwell's equations (or the Poisson's equation in the often used electrostatic limit) as well as boundary conditions for particles. For a lot of interesting applications, including probe diagnostics, these fall into the "special" category mentioned in 2.2.4 "Boundary conditions" (page 16). On the outside one usually needs an interface with bulk plasma. This means that particles that exit the system are simply discarded and a suitable flux of particles from the outside has to be provided. Metal electrodes in the interior of the domain usually need a Dirichlet boundary condition for the electrostatic potential and absorb any particles that cross their geometrical border. Some more advanced models include processes that occur at surfaces that are in contact with plasma, for example secondary emission or charge accumulation on dielectric surfaces.

In systems that are not completely ionized, it is often necessary to include the influence of the particles of the neutral gas or gases. A stochastic procedure is used, where the atoms of the neutral gas are not explicitly simulated. This is called Monte-Carlo collisions and it corresponds to the right-hand side of the Boltzmann kinetic equation. Events are generated randomly from a set of possible processes of known scattering cross-sections. The results depend on the specific method used, which is a problem in its own right, as well as on the available cross-section data. The focus of this thesis is only fully ionized plasma and the details of Monte-Carlo collisions will not be discussed here.

A general model is formulated in three spatial dimensions, but certain systems have symmetries that allow for the reduction of the number of dimensions by one or two. The real simulated area becomes a column in one dimension and a slab in two dimension. The area of the base (of the column) or the slab thickness are numerical parameters that do not have a direct physical meaning. One has to be careful with the interpretation of particle properties and calculation of physical quantities. The specific way of doing this depends on the model details, especially on the way the electric field is computed. The common feature of all these situations is that the pseudoparticle technique is implicitly present and that the total number of particles in the system can be set thanks to the scaling freedom provided by the base area or slab thickness. The choice of these parameters influences the computational demands of the model.

Particle interaction

Perhaps the most crucial part of a particle plasma simulation is the computation of the particle interaction. The reason is the fact that it is the most demanding part as for computer time and approximations have to be made. The choice of these approximations has a substantial influence on the behaviour of the model.

Traditionally, the methods fall into three distinct categories.

Particle-particle These methods compute the interaction as a sum of particle-particle interactions. This does not take into account the boundary conditions and additional steps have to be taken to impose them.

Particle-mesh These methods use a spatial mesh through which interactions are computed.

Particle-particle, particle-mesh This is a combination of the two previous methods, part of the interaction is taken as direct action, the rest through a mesh.

In the rest of this section we will go through several approaches to this central problem of plasma simulations.

2.3.4 Particle-in-cell

Perhaps the most widely used method is called particle-in-cell. It is a particle-mesh method. The two textbooks mentioned above, [BL91] and [HE88], focus almost exclusively on this way of computing interactions.

Particle positions are interpolated to a mesh to obtain particle densities and therefore also the charge density. This is then used to solve Poisson's equation on the grid. This solution is then differentiated numerically to get the electric field, which can be interpolated from the grid back to the particles. Boundary conditions can be imposed on the solution of Poisson's equation, making this method suitable for typical problems in modelling the vicinity of probes. Because of the grid, this is quite a rough approximation and it neglects the interaction of particles within a single cell. Its asymptotic time complexity is $\mathcal{O}(N \log N)$ and in its efficiency it relies heavily on a fast Poisson solver on a grid.

Conditions of stability

There are several criteria that need to be fulfilled for a particle-in-cell simulation to be stable. Because various sources differ in the exact numbers, we will present only the approximate bounds here.

- The grid spacing must resolve the Debye length.

- A particle should not pass a distance greater than the grid spacing in one time step. The average velocity is usually used for this comparison.
- The time step should resolve the plasma frequency.
- The time step should resolve the cyclotron frequency.
- The number of particles per cell should be sufficient. Numbers in the range 10–100 are usually given for the numerical noise to be at an acceptable level.

Interpolation of charge

To be able to solve Poisson's equation on a grid, we have to obtain particle densities (and thus the charge density) on this grid first. This is done by one of several interpolation schemes. The most simple one is called Nearest Grid Point and it just assigns each particle to the grid point that is nearest to it. A whole class of interpolation schemes is called Cloud-in-Cell. Parts of each particle are assigned to several nodes in its vicinity. This is done by a weighting scheme, in which each particle is smeared out by a function that represents its "shape". Several examples can be found in [BL91] or [Ver05]. Another option is to use linear interpolation between nearest grid points. In two dimensions that would mean bilinear interpolation between four grid points.

Solving Poisson's equation

Having the charge density, we can solve Poisson's equation

$$\Delta\varphi(\mathbf{x}) = -\frac{\rho(\mathbf{x})}{\varepsilon_0} \tag{2.48}$$

on the grid. Boundary conditions on the value and/or normal derivative can be imposed on this solution. Partial or full periodicity can also be achieved. The specific way of imposing boundary conditions depends on the solver used.

On a grid, differentiation is represented as finite differences and the partial differential equation (2.48) changes to a system of linear equations for the unknown values of φ in the grid points. A linear system can be represented by a matrix, whose form depends on the chosen order of the differentiation scheme and on the boundary conditions. This is a standard problem in numerical mathematics and there are several ways of solving it, some more suitable than others.

The electric field (and thus the force acting on particles) can be obtained by numerical differentiation on the grid.

2.3.5 P3M

P3M is an abbreviation that stands for Particle-particle, particle-mesh [EHL80]. It combines the use of a grid with direct particle forces. Its implementation is more complicated than in the case of particle-in-cell and the time complexity is also quite high, approaching $\mathcal{O}(N^2)$ [CKV+06]. It is not used much nowadays.

2.3.6 Ewald summation

The method of Ewald summation [Ewa21] is based on dividing the problem into sums in real and reciprocal space in a specific way. These sums converge faster than the original sum in real space and therefore can be truncated. This algorithm and all its later variants work only in periodic boundary conditions and are therefore unsuitable for a large class of plasma simulations and we will not discuss it in more detail here.

2.3.7 Direct summation

This is the most direct approach one can take. For each particle, the total force is calculated as the sum of forces from all the other particles in the system, therefore it belongs to the particle-particle category. The time complexity is obviously $\mathcal{O}(N^2)$, which is unacceptable for problems that usually arise in plasma simulations. This method is, however, very accurate, as there is no approximation involved. By itself, it has no mechanism of imposing boundary conditions.

2.3.8 Hierarchical algorithms

It has been known for a long time that direct summation can be improved substantially while retaining almost all of its accuracy. There are several approaches but the basic idea is the same in all cases. Because the electrostatic interaction decreases with distance, it would be natural to include the interaction with near particles exactly while taking particles that are further away only approximately, perhaps grouping them in some way. Generally, these algorithms are sometimes called treecodes. The first idea was published in 1985 [App85] and was further refined by Barnes and Hut [BH86] (now called the Barnes-Hut algorithm) and independently as the Fast Multipole Method by Greengard and Rokhlin [GR87]. These methods would loosely fit into the particle-particle category, although this is not entirely accurate. A big advantage over mesh based methods is the inclusion of small distance interaction. All of them are suitable only for open boundary conditions.

Barnes-Hut algorithm

The Barnes-Hut algorithm was motivated by astrophysical simulations of gravitational problems. The spatial domain containing all the particles is divided into eight cubes and each of these cubes is further divided in such a way that there is at most one particle in each bottom level cube. Each box on each level (called a node) holds information about its centre of mass — the center of mass of all the particles it contains. Thus we have clusters of particles at several levels of detail. The corresponding data structure is an octree in three dimensions, a quadtree in two dimensions and a binary tree in one dimension. This division is illustrated for a simple configuration of 12 particles in Figure 2.6.

The potential at a specific point is then computed using the tree in the following way. Starting at the root node of the tree, the distance r between the point and the centre of mass of the node is measured and compared to the size of the box d. Generally, this is called a multipole acceptance criterion and in the Barnes-Hut algorithm has the form

$$\frac{r}{d} > \theta, \tag{2.49}$$

where θ is a chosen parameter that influences the accuracy of the result as well as its computational demands. If this criterion is met, only the interaction with the whole cluster is included by adding the potential from its centre of mass. In the opposite case, more detail is required and the same is repeated for all the descendant nodes of the current one. The procedure converges to direct summation in the limit $\theta \to \infty$. Force can be computed in a similar way. The asymptotic time complexity of this algorithm is $\mathcal{O}(N \log N)$ and it produces satisfactory results for relatively small values of θ. It has to be modified for the purpose of electrostatic problems either by using two independent trees for positive and negative charges or by using a more complicated tree that holds both charge types.

Fast Multipole Method

The Fast Multipole Method was originally proposed for gravitational as well as electrostatic problems. It is more sophisticated and also more complicated to implement than the Barnes-Hut algorithm. It uses the same basic division as Barnes-Hut (although the tree is balanced in the original formulation), but rather than using only the centre of mass, a multipole expansion of each cube is performed around the centre of that cube. The expansions for higher levels are obtained from lower levels with the use of formulae for translation and addition of multipole expansions.

The evaluation of potential or force is also more complicated, using not only particle-particle and particle-cluster interactions, but also cluster-cluster terms. As a result, the asymptotic time complexity improves to $\mathcal{O}(N)$. However, the prefactor at the linear term is very high, making it useful only for very large systems.

Other variants

There is a potentially large number of possible modifications of the two original algorithms. One can include higher terms of the expansion in a Barnes-Hut type of algorithm, use a different kind of expansion and other options. The basic idea remains

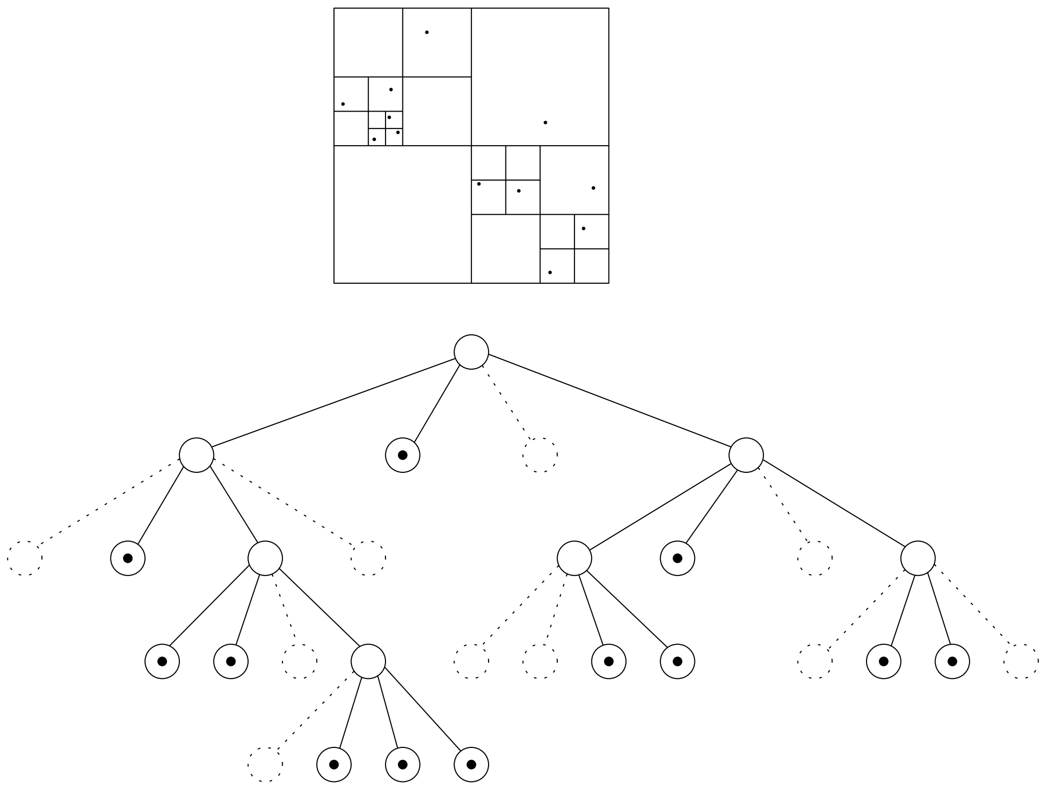


Figure 2.6 A Barnes-Hut tree in two dimensions. Circles represent nodes of the tree and each of these corresponds to a square at the appropriate level. Circles with a dot inside represent leaves — nodes that contain a single particle. Dashed lines represent parts of the tree that are not present, because there are no particles in the corresponding areas.

the same and the specific choice of details depends on the demands of the problem the algorithm is being applied to.

2.3.9 Boundary integral/treecode

Recently, there has been progress in the use of particle-particle methods together with boundary conditions. The published work ([CKV04], [CKV] and [CKV+06]) uses one specific kind of hierarchical algorithms to compute particle interaction, but a framework can easily be generalized from this, allowing the use of other treecodes. There were several applications, but none of them in the area of probe diagnostics and magnetized plasmas.

A reformulation of the electrostatic problem using Green's theorem allows to impose (possibly mixed) boundary conditions on the value and/or the normal derivative of the electrostatic potential. There are no restrictions on the geometry of these borders, allowing the treatment of a wide range of problems, including probe diagnostics. To impose the boundary conditions, virtual point charges or dipoles are created at the boundary. Their values are set using the configuration of particles within the domain in such a way that the potential computed from the interior particles and the virtual particles satisfies the boundary conditions.

CHAPTER
THREE

Objectives

This work is primarily motivated by plasma probe diagnostics. There are diverse areas of interest in the physics involved in plasma probes as well as in the computer methods. The focus of this thesis is particle modelling of the vicinity of probes immersed in plasma with emphasis on the solution of the electrostatic problem with boundary conditions. The objective is to address the following issues:

- design a 2D electrostatic particle model of a spatial domain of plasma with external magnetic field,
- incorporate two different methods of computing the electrostatic interaction — the particle-in-cell method and the boundary integral/treecode method,
- implement the model,
- study and improve the computational efficiency of the model and discuss the possibility of its extension to three dimensions and
- apply the model to plasma probe problems, compare the two methods of computing interaction and discuss the results.

CHAPTER

FOUR

Methodology

This chapter will present all the details that are necessary to formulate and implement a plasma model. In its first part, we will prepare several aspects of the model that need some more elaboration, in the second part, we will talk about the efficient implementation of three different models.

4.1 Theory

In this section we will look at a method of deriving symplectic propagators and use it to obtain the HARHA propagator. We will also show explicitly how to deal with two different problems of a plasma model. The first one will be a way to impose a boundary condition with algorithms, that are originally only usable with open boundary conditions. For this purpose, we will reformulate the electrostatic problem. The second problem will be the simulation of an interface between the model and bulk plasma, both with and without magnetic field.

4.1.1 Symplectic propagators

There exists an elegant and systematic way to derive numerical symplectic propagators of Hamiltonian systems [TBM92]. We will review this process and use it to derive a propagator that includes the influence of magnetic field.

Symplecticity

We will work with a Hamiltonian that does not have an explicit time dependence and is separable into two terms — the kinetic term that is a function of canonical momenta and the potential term that is a function of the generalized coordinates. Without loss of generality we will leave out the indices over multiple particles. The Hamiltonian then becomes

$$H(\mathbf{q},\mathbf{p}) = T(\mathbf{p}) + V(\mathbf{q}) = \frac{\mathbf{p}^2}{2m} + V(\mathbf{q}) \tag{4.1}$$

and the Hamilton's equations of motion are

$$\dot{\mathbf{q}} = \{\mathbf{p}, H\} = \nabla_{\mathbf{p}} H = \frac{\mathbf{p}}{m}, \tag{4.2}$$

$$\dot{\mathbf{p}} = \{\mathbf{q}, H\} = -\nabla_{\mathbf{q}} H = -\nabla_{\mathbf{q}} V(\mathbf{q}). \tag{4.3}$$

The evolution operator U can be used to represent a formal solution of the equations of motion:

$$\begin{bmatrix}\mathbf{q}(t)\\ \mathbf{p}(t)\end{bmatrix} = U(t,t_0)\begin{bmatrix}\mathbf{q}(t_0)\\ \mathbf{p}(t_0)\end{bmatrix}. \tag{4.4}$$

This operator is unitary, expressed by $U(t_0,t) = U^{-1}(t,t_0)$, which is equivalent to time reversibility. It represents a transformation of the phase space that preserves the symplectic form. This, in turn, implies the conservation of phase space volume and other quantities under the transformation. Details can be found for example in [SSC93]. Note that symplecticity is a "stronger" property than just energy conservation.

Original procedure

Let us now turn our attention to the derivation of symplectic numerical propagators. A comprehensive treatment of the procedure can be found in the original paper [TBM92] as well as the book [FS01]. We will need the Trotter formula

$$\exp(A+B) = \lim_{p\to\infty}\left[\exp\left(\frac{A}{2p}\right)\exp\left(\frac{B}{p}\right)\exp\left(\frac{A}{2p}\right)\right]^p, \tag{4.5}$$

where A and B are operators that do not commute.

The time change of any function on the phase space can be written as

$$\dot{f}(\mathbf{q},\mathbf{p}) = \{f, H\} = iLf(\mathbf{q},\mathbf{p}), \tag{4.6}$$

where L is the Liouville operator given by

$$iL = \{\dots, H\} = \dot{\mathbf{q}}\cdot\nabla_{\mathbf{q}} + \dot{\mathbf{p}}\cdot\nabla_{\mathbf{p}}. \tag{4.7}$$

The formal solution of equation (4.6) can be written as

$$f(t) = \exp\left(iL(t-t_0)\right) f(t_0), \tag{4.8}$$

which also gives the formal solution (4.4) of the equations of motion for the special case of f being one of the coordinates or momenta. We can now identify the expression for the evolution operator U. We would like to be able to approximate this operator for short time steps, which would allow a numerical solution of the equations of motion.

The Liouville operator can be split into two parts:

$$iL_{\mathbf{q}} = \dot{\mathbf{q}} \cdot \nabla_{\mathbf{q}}, \tag{4.9}$$

$$iL_{\mathbf{p}} = \dot{\mathbf{p}} \cdot \nabla_{\mathbf{p}} \tag{4.10}$$

and complications arise from the fact that these two operators do not commute, which means that an exponential of their sum cannot be expressed as a product of exponentials. Let us now have a look at the action of partial evolution operators given as

$$U_{\mathbf{q}}(t, t_0) = \exp\left(iL_{\mathbf{q}}(t - t_0)\right), \tag{4.11}$$

$$U_{\mathbf{p}}(t, t_0) = \exp\left(iL_{\mathbf{p}}(t - t_0)\right). \tag{4.12}$$

It is well known that an operator of this form causes a shift of the appropriate coordinate. We say that $iL_{\mathbf{q}}$ or $iL_{\mathbf{p}}$ is a generator of translation in the designated coordinate. This can easily be verified by the use of a Taylor expansion of the exponential. As an example, we can write

$$U_{\mathbf{q}}(t, t_0) f(\mathbf{q}, \mathbf{p}) = f(\mathbf{q} + (t - t_0)\dot{\mathbf{q}}, \mathbf{p}). \tag{4.13}$$

Both of these operators are obviously unitary, which means that their combinations will be unitary as well. The expressions for $\dot{\mathbf{q}}$ and $\dot{\mathbf{p}}$ can be taken from Hamilton's equations. Moreover, we are usually interested in the time propagation of positions and velocities, therefore we already have $\dot{\mathbf{q}}$.

We can now apply the Trotter formula (with truncation) to the full evolution operator to obtain an approximate propagator in terms of the above partial propagators to get from t to $t + \Delta t$. One possible approximation is

$$U_{Euler}(\Delta t) = U_{\mathbf{q}}(\Delta t) U_{\mathbf{p}}(\Delta t) \tag{4.14}$$

which is the Euler–Cromer, or symplectic Euler, propagator. This is, however, usually not used. Another choice could be

$$U_{velocity\ Verlet}(\Delta t) = U_{\mathbf{p}}\left(\frac{\Delta t}{2}\right) U_{\mathbf{q}}(\Delta t) U_{\mathbf{p}}\left(\frac{\Delta t}{2}\right), \tag{4.15}$$

which gives us the velocity Verlet propagator. By switching the order of the operators we get the position Verlet propagator

$$U_{position\ Verlet}(\Delta t) = U_{\mathbf{q}}\left(\frac{\Delta t}{2}\right) U_{\mathbf{p}}(\Delta t) U_{\mathbf{q}}\left(\frac{\Delta t}{2}\right) \tag{4.16}$$

first obtained in [TBM92]. The explicit expressions for these propagators can be found in 2.2.2 “Time evolution” (page 9).

The leap-frog propagator can be obtained by the following observation. The evolution of a system using the velocity Verlet propagator is obtained by applying the sequence of operators (4.15) repeatedly. The borders of time steps are arbitrary and therefore the leap-frog propagator is simply putting these borders at different places in this sequence — those, that have positions and momenta (or velocities) shifted by half a time step. It seems that the same argument holds for the position Verlet propagator, yielding a leap-frog propagator that has velocities retarded rather than advanced by half a time step. This leap-frog should have the same trajectory as the position Verlet propagator. This has not been investigated in the present work.

It can be shown that a propagator obtained by this procedure is indeed symplectic and corresponds exactly to the time evolution of a system with a Hamiltonian that is close to the original one but not identical.

Magnetic field

The case with magnetic field is slightly more complicated, but with a modification we can apply the above procedure. The Hamiltonian of a charged particle in an external magnetic field is

$$H(\mathbf{p}) = \frac{(\mathbf{p} - e\mathbf{A})^2}{2m} + e\varphi \tag{4.17}$$

and Hamilton's equations are

$$m\dot{\mathbf{q}} = \mathbf{p} - e\mathbf{A}, \tag{4.18}$$
$$\dot{\mathbf{p}} = e\left(-\nabla_{\mathbf{q}}\varphi + \dot{\mathbf{q}} \times \mathbf{B}\right). \tag{4.19}$$

The Liouville operator can then be split into three non-commuting terms:

$$iL_{\mathbf{q}} = \dot{\mathbf{q}} \cdot \nabla_{\mathbf{q}}, \tag{4.20}$$
$$iL_{\mathbf{p}} = -e\nabla_{\mathbf{q}}\varphi \cdot \nabla_{\mathbf{p}}, \tag{4.21}$$
$$iL_{\mathbf{B}} = e\dot{\mathbf{q}} \times \mathbf{B} \cdot \nabla_{\mathbf{p}} \tag{4.22}$$

Let us first show that the partial propagator generated by $iL_{\mathbf{B}}$ has the effect of rotating the velocity vector. By exchanging the order of the vectors in the cross product and taking $1/m$ out of $\nabla_{\mathbf{p}}$ the above expression can be written as

$$iL_{\mathbf{B}} = -\frac{e\mathbf{B}}{m} \cdot \dot{\mathbf{q}} \times \nabla_{\dot{\mathbf{q}}}. \tag{4.23}$$

A generator of a spatial rotation can be written as $\mathbf{n} \cdot L$, where $\mathbf{n}$ is a direction vector and $L = \mathbf{r} \times \nabla_{\mathbf{r}}$ is the angular momentum operator. In analogy with that we can see that the above expression has the form of a rotation generator. The prefactor is the cyclotron frequency ω_c (see (2.9)). We can now write the partial propagator

$$U_{\mathbf{B}}(\Delta t) = \exp\left(-\frac{e\mathbf{B}\Delta t}{m} \cdot \dot{\mathbf{q}} \times \nabla_{\dot{\mathbf{q}}}\right) \tag{4.24}$$

The Trotter formula can be generalized for a sum of operators [TBM92]. This can be used to write an approximate propagator as

$$U_{HARHA}(\Delta t) = U_{\mathbf{q}}\left(\frac{\Delta t}{2}\right) U_{\mathbf{p}}\left(\frac{\Delta t}{2}\right) U_{\mathbf{B}}(\Delta t) U_{\mathbf{p}}\left(\frac{\Delta t}{2}\right) U_{\mathbf{q}}\left(\frac{\Delta t}{2}\right), \tag{4.25}$$

which is a generalization of the expression (4.16) for the position Verlet propagator. For practical reasons it is usually implemented in the leap-frog fashion with velocities retarded by half a time-step. When applied to positions and velocities, this expression represents the HARHA propagator stated earlier in 2.2.2 "Time evolution" (page 9). From the way it was derived here it seems that there should also be an analogy related to the velocity Verlet instead of the position Verlet. As in the case of the leap-frog propagator, this has not been investigated as it is not the focus of this thesis.

4.1.2 Reformulation of the electrostatic problem

We will now formulate the electrostatic problem and then recast it in a way that is suitable for further use. This procedure is based on [CKV$^+$06].

We want to find the solution $\varphi(\mathbf{x})$ of Poisson's equation in the interior of some domain $\Omega \subset \mathbb{R}^n$ subject to a general Newton boundary condition for $n = 1, 2, 3$. Dirichlet and von Neumann boundary conditions are just special cases and a mixed boundary condition could be used as well. Formally, we want to solve the equation

$$\Delta\varphi(\mathbf{x}) = -\frac{\rho(\mathbf{x})}{\varepsilon_0}, \quad \mathbf{x} \in \Omega \setminus \partial\Omega \tag{4.26}$$

with the boundary condition being imposed in the sense of a limit

$$\lim_{\mathbf{x}\to\mathbf{z}} \left(a\varphi(\mathbf{x}) + b\mathbf{n} \cdot \nabla\varphi(\mathbf{x})\right) = \gamma(\mathbf{z}), \quad \mathbf{z} \in \partial\Omega, \tag{4.27}$$

where $\mathbf{n}$ is the normal vector to the boundary of Ω.

For the following text, we will need the concept of Green's functions. The Green's function $G(\mathbf{x}|\mathbf{y})$ of a differential operator (in this case the Laplace operator in the Poisson's equation) is defined by the equation

$$\Delta G(\mathbf{x}|\mathbf{y}) = \delta(\mathbf{x}|\mathbf{y}), \quad \mathbf{x},\, \mathbf{y} \in \mathbb{R}^n. \tag{4.28}$$

Thus it can be considered as the "reaction" of the operator at point $\mathbf{x}$ to a unit "source" at point $\mathbf{y}$. It can be used to write the solution of an inhomogeneous equation as a convolution of the right-hand side and the Green's function.

Green's functions for the Poisson's equation in one, two and three dimensions are [Sou]:

$$G(x|y) = \frac{|x-y|}{2}, \quad x, y \in \mathbb{R}; \tag{4.29}$$

$$G(\mathbf{x}|\mathbf{y}) = \frac{\log|\mathbf{x}-\mathbf{y}|}{2\pi}, \quad \mathbf{x}, \mathbf{y} \in \mathbb{R}^2; \tag{4.30}$$

$$G(\mathbf{x}|\mathbf{y}) = -\frac{1}{4\pi}\frac{1}{|\mathbf{x}-\mathbf{y}|}, \quad \mathbf{x}, \mathbf{y} \in \mathbb{R}^3. \tag{4.31}$$

The motivation to reformulate the above electrostatic problem is the intention to use a treecode to evaluate φ. The only thing we can do with a treecode is to evaluate the influence of point charges. The boundary condition in the above formulation prevents the use of a treecode. Therefore, we will now formulate the problem so that the boundary conditions can be represented by charge and dipole density. This can then be discretized into point charges in a numerical representation of a model.

For the next step, we need Green's second identity [Wei]

$$\int\limits_{\Omega} (h\Delta g - g\Delta h)\,\mathrm{d}\mathbf{x} = \oint\limits_{\partial\Omega} (h\nabla g - g\nabla h)\cdot \mathbf{n}\mathrm{d}S. \tag{4.32}$$

Let us take $h = \varphi$ and $g = G$ in (4.32) which produces the following four terms:

$$\begin{aligned}&\int\limits_{\Omega} (\varphi(\mathbf{x})\Delta_{\mathbf{x}} G(\mathbf{x}|\mathbf{y}) - G(\mathbf{x}|\mathbf{y})\Delta_{\mathbf{x}}\varphi(\mathbf{x}))\,\mathrm{d}\mathbf{x} = \\ &\oint\limits_{\partial\Omega} (\varphi(\mathbf{x})\nabla_{\mathbf{x}} G(\mathbf{x}|\mathbf{y}) - G(\mathbf{x}|\mathbf{y})\nabla_{\mathbf{x}}\varphi(\mathbf{x}))\cdot \mathbf{n}\mathrm{d}_{\mathbf{x}}S.\end{aligned} \tag{4.33}$$

This can be simplified, because by definition we have in the first term

$$\Delta G(\mathbf{x}|\mathbf{y}) = \delta(\mathbf{x}|\mathbf{y}) \tag{4.34}$$

and in the second term

$$\Delta\varphi(\mathbf{x}) = -\frac{\rho(\mathbf{x})}{\varepsilon_0}. \tag{4.35}$$

The integration with $\delta(\mathbf{x}|\mathbf{y})$ gives simply $\varphi(\mathbf{y})$, yielding the expression

$$\begin{aligned}\varphi(\mathbf{y}) = &-\int\limits_{\Omega} G(\mathbf{x}|\mathbf{y})\frac{\rho(\mathbf{x})}{\varepsilon_0}\mathrm{d}\mathbf{x} + \\ &\oint\limits_{\partial\Omega} (\varphi(\mathbf{x})\nabla_{\mathbf{x}} G(\mathbf{x}|\mathbf{y}) - G(\mathbf{x}|\mathbf{y})\nabla_{\mathbf{x}}\varphi(\mathbf{x}))\cdot \mathbf{n}\mathrm{d}_{\mathbf{x}}S,\end{aligned} \tag{4.36}$$

which is a formal solution of (4.26). It can be split into two distinct parts — the particular solution

$$\varphi_P(\mathbf{y}) = -\int\limits_{\Omega} G(\mathbf{x}|\mathbf{y}) \frac{\rho(\mathbf{x})}{\varepsilon_0} \mathrm{d}\mathbf{x} \tag{4.37}$$

and the homogeneous solution

$$\varphi_H(\mathbf{y}) = \oint\limits_{\partial\Omega} \left(\varphi(\mathbf{x}) \nabla_{\mathbf{x}} G(\mathbf{x}|\mathbf{y}) - G(\mathbf{x}|\mathbf{y}) \nabla_{\mathbf{x}} \varphi(\mathbf{x})\right) \cdot \mathbf{n} \mathrm{d}_{\mathbf{x}} S. \tag{4.38}$$

The particular solution corresponds to the Poisson's equation with the original right-hand side in open boundary conditions. The homogeneous solution corresponds to a charge and/or dipole density distributed on the boundary of Ω. By setting these densities appropriately we can impose any boundary condition and therefore obtain a solution to the original problem.

For a given boundary condition, we have to find proper boundary distributions of charge and dipole moment. These can be obtained from (4.36) by taking the limit that defines the boundary condition. What remains of the expression is an integral equation for the charge and dipole moment densities on the boundary. We will show the limit explicitly, because there is one subtle point along the way [Pra]. Let us take the limit $\mathbf{y} \to \mathbf{z}$, where $\mathbf{y} \in \Omega$ and $\mathbf{z} \in \partial\Omega$. Further, the value and normal derivative (whether known or unknown) of φ at the boundary are

$$\varphi(\mathbf{z}) \equiv \alpha(\mathbf{z}), \tag{4.39}$$

$$\nabla \varphi(\mathbf{z}) \cdot \mathbf{n} \equiv \beta(\mathbf{z}); \quad \mathbf{z} \in \partial\Omega. \tag{4.40}$$

Then, the limits for individual terms are as follows:

$$\varphi(\mathbf{y}) \to \alpha(\mathbf{z}), \tag{4.41}$$

$$\varphi_P(\mathbf{y}) \to \varphi_P(\mathbf{z}), \tag{4.42}$$

$$\oint\limits_{\partial\Omega} \varphi(\mathbf{x}) \nabla_{\mathbf{x}} G(\mathbf{x}|\mathbf{y}) \cdot \mathbf{n} \mathrm{d}_{\mathbf{x}} S \to \frac{\alpha(\mathbf{z})}{2} + \oint\limits_{\partial\Omega} \alpha(\mathbf{x}) \nabla_{\mathbf{x}} G(\mathbf{x}|\mathbf{z}) \cdot \mathbf{n} \mathrm{d}_{\mathbf{x}} S, \tag{4.43}$$

$$-\oint\limits_{\partial\Omega} G(\mathbf{x}|\mathbf{y}) \nabla_{\mathbf{x}} \varphi(\mathbf{x}) \cdot \mathbf{n} \mathrm{d}_{\mathbf{x}} S \to -\oint\limits_{\partial\Omega} G(\mathbf{x}|\mathbf{z}) \nabla_{\mathbf{x}} \varphi(\mathbf{x}) \cdot \mathbf{n} \mathrm{d}_{\mathbf{x}} S. \tag{4.44}$$

Only the third limit cannot be performed by direct substitution and the additional term is due to the singular nature of the limit [Pra], [DiB94]. Together, we have the following expression at the boundary:

$$\frac{\alpha(\mathbf{z})}{2} = \varphi_P(\mathbf{z}) + \oint\limits_{\partial\Omega} \alpha(\mathbf{x}) \nabla_{\mathbf{x}} G(\mathbf{x}|\mathbf{z}) \cdot \mathbf{n} \mathrm{d}_{\mathbf{x}} S - \oint\limits_{\partial\Omega} \beta(\mathbf{x}) G(\mathbf{x}|\mathbf{z}) \mathrm{d}_{\mathbf{x}} S. \tag{4.45}$$

Let us now illustrate the situation by a specific example of a Dirichlet boundary condition given by

$$\lim_{\mathbf{y}\to\mathbf{z}} \varphi(\mathbf{x}) = \alpha(\mathbf{z}), \quad \mathbf{z} \in \partial\Omega. \tag{4.46}$$

All the terms in (4.45) except for the last one can be evaluated at the boundary. What remains is an inhomogeneous Fredholm integral equation of the first kind for $\beta(\mathbf{z})$. After it is solved, β can be used to evaluate the homogeneous solution. For the particularly simple case of $\alpha(\mathbf{z}) = 0$, we have

$$\varphi_P(\mathbf{z}) = \oint_{\partial\Omega} \beta(\mathbf{x}) G(\mathbf{x}|\mathbf{z}) \mathrm{d}_{\mathbf{x}} S. \tag{4.47}$$

In other cases, including mixed boundary conditions, the procedure would be similar, only the resulting integral equation would be different.

4.1.3 Boundary integral/treecode

As has been stated in 2.3.9 “Boundary integral/treecode” (page 26), treecodes can be used in combination with boundary conditions. It is the above reformulation that allows this approach. It is important to realize that this is a very general possibility and is limited only by the ability of the algorithm to compute the potential and force from a set of point charges given by the charge density

$$\rho(\mathbf{x}) = \sum_i q_i \delta(\mathbf{x}|\mathbf{y}_i), \tag{4.48}$$

where charges q_i are at points $\mathbf{y}_i$. There are no other requirements on the underlying electrostatic algorithm than to be able to evaluate the expressions for the electrostatic potential

$$\varphi(\mathbf{x}) = \sum_i q_i G(\mathbf{x}|\mathbf{y}_i) \tag{4.49}$$

and the electric field

$$\mathbf{E}(\mathbf{x}) = -\sum_i q_i \nabla_{\mathbf{x}} G(\mathbf{x}|\mathbf{y}_i). \tag{4.50}$$

Boundary condition

To be able to impose the boundary condition, equation (4.45) must first be solved. We will show how to do that in the case of the Dirichlet boundary condition.

The kernel of the equation, which is the Green’s function, needs to be discretized. This can be done by dividing the boundary into panels and integrating over each of them. In this work, the integrals were approximated by the value in the middle of the panels times the area (length in 2D). This works well enough for simple geometries. The result is a matrix G_{ij} and so the equation (4.45) is represented by a linear system.

This system is dense and its size depends on the chosen resolution of the discretization of the boundary. The matrix is independent of the system state (it depends only on the Green's function) and can therefore be prepared before the simulation. This includes computing the values as well as an LU decomposition (see C.3.2 “Direct methods” (page 107)), as we will need many solutions of this system with different right-hand sides.

To obtain the full solution of the Poisson's equation with boundary conditions, we need to perform several steps. The particular solution can be evaluated using a treecode without any additional information. This is used to get the values of the potential at the boundary. That is, in turn, used to solve (4.45) using the above numerical procedure. Finally, the homogeneous solution can be evaluated. The sum of the particular and homogeneous solutions at any point in the interior of Ω gives the correct result that respects the desired boundary condition.

Treecode choice

Let us now have a look at the underlying electrostatic algorithm. The most straightforward choice would be direct summation, but it is prohibitively slow for any problems of useful size. A more practical option is to use some of the algorithms mentioned in 2.3.8 “Hierarchical algorithms” (page 24).

There are two phases of the algorithm and an evaluation of some charges is needed in both. As a normal Barnes-Hut tree is not well-suited for both positive and negative charges, there would be four trees to be built and applied to each particle. The first implementation of this approach did not look very promising and therefore a different approach was needed. An additional requirement is a relative simplicity of the treecode, at least for this initial implementation. With a reasonably modular code, it is possible to exchange the treecode for a different one while keeping the rest of the programme intact. For this reason, the use of the Fast Multipole Method was not attempted.

A treecode was developed that is simple, able to accommodate two signs of charge and well suited for the current application. Unlike the Barnes-Hut tree, it is built from the bottom up, first sorting particles to regular bins (the bottom-most level of the tree), then adding these bins together to form the higher levels of the tree. If the approximation by the lowest level node is not enough, the influence of all the particles in that node is applied.

4.1.4 Bulk plasma interface

In the modelling of plasma probe diagnostics as well as some other applications, a special kind of boundary condition for the model is needed. It has to represent the interface with bulk plasma faithfully. This interface should simulate the placement of the probe in a much larger spatial region than is actually represented in the model.

This includes two areas — a boundary condition for Poisson's equation and a condition for particle behaviour.

Electrostatic potential boundary conditions

In the case of a domain without internal electrodes, it is not obvious what the boundary condition for the electrostatic potential should be. The instantaneous value of φ fluctuates around an average value, in bulk plasma with no external fields we should expect $\varphi = 0$. One possibility is to use that average value as a Dirichlet boundary condition, but using the average derivative (also zero) as a von Neumann boundary condition or some suitable combination of both as a Newton boundary condition seems to be an equally plausible option.

If there are electrodes (used as probes) present in the interior of the domain, a Dirichlet boundary condition must be used to set the bias of the probe with respect to the plasma.

Particle injection

As for the particles, we have to discard those that exit the domain and generate new particles that enter the domain from bulk plasma. As the simulation runs in discrete time steps, we have to inject particles that could have entered during the whole time interval. This involves generating a random number of particles and a correct position as well as velocity for each. At the boundaries with metal electrodes, particles are only discarded.

The number of particles that pass an interface during a given time interval should be Poisson distributed. However, even if we generate a constant flux through a relatively large interface and distribute the particles randomly along the interface, there will still be proper fluctuations in the numbers of particles for small parts of that interface.

We assume the Maxwell distribution of velocities in the bulk. It would be possible to work with a different distribution as well, perhaps obtained from experiment or another model, but things would be more complicated. We can split the problem into the dimension perpendicular to the interface and two dimensions along the interface. The spatial distribution along the interface depends on the specific model, but in most cases it is simply uniform. The velocities along the interface are distributed according to the Gaussian distribution in each of the two dimensions.

The spatial and velocity distributions perpendicular to the interface are interconnected and a look at the one-dimensional phase space will help us understand the situation. Let us have an ideal one-dimensional gas with a Gaussian distribution of velocities in the half-space given by $x < 0$. The other half-space is empty and $x = 0$ represents our desired interface. The phase-space density is shown in Figure 4.1.

If we now let the system evolve for some finite time interval, each point will move

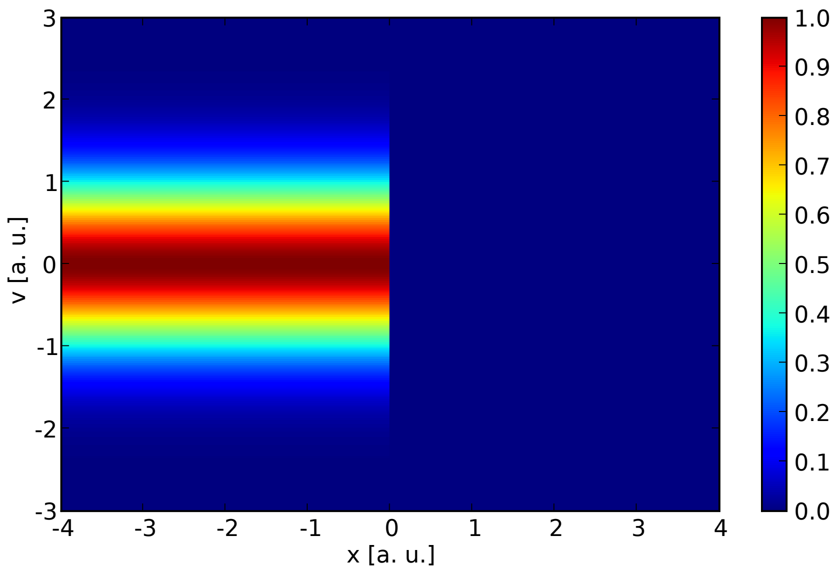


Figure 4.1 A phase portrait of a one-dimensional ideal gas. The gas is only in the half-space $x < 0$. Arbitrary units are used.

along x by a distance proportional to its velocity. The result is shown in Figure 4.2. The particles we want to generate will be drawn from the distribution marked by the dashed lines in Figure 4.2. Particles exiting the domain during that time step will be in the corresponding area on the other side of the interface.

Drawing samples from the two-dimensional distribution mentioned before may seem complicated, but it is much easier if done in two steps. First, we will generate the velocity. The distribution of velocities perpendicular to the interface is also Gaussian, but the velocity distribution of the flux across the interface will be biased by velocity magnitude, because faster particles will pass more often than slow ones. The correct distribution is therefore the Rayleigh distribution

$$f(v) \sim v \exp(-\alpha v^2), \tag{4.51}$$

where α is the usual factor that appears in the Maxwell velocity distribution. For each of these particles we can assume that it has passed the interface at some random time during the time interval. Therefore we will place it at a distance d from the interface

$$d = v u_{(0,1]} \Delta t, \tag{4.52}$$

where $u_{(0,1]}$ is a random number with uniform distribution on the interval $(0, 1]$.

This way, we can simulate the particle behaviour at an interface with bulk plasma.

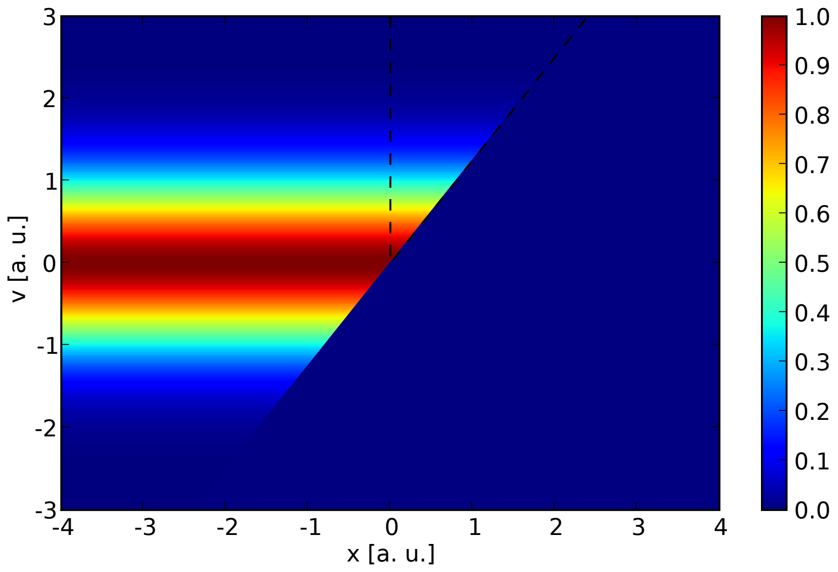


Figure 4.2 Same as Figure 4.1, but after some finite time interval.

Particle injection with magnetic field

To do the same as above for a system with magnetic field, we have to discuss velocity distribution in such a system first. The dimension along magnetic field lines is unaffected by the field and so the distribution of velocities remains Gaussian. In the plane perpendicular to the field lines, the direction of velocity vectors rotates with the cyclotron frequency (2.9). The projections into two perpendicular directions in that plane are, however, still Gaussian distributed, as the magnetic field changes only the direction and not magnitude of velocity.

With this in mind, we can repeat the above procedure with a slight modification to obtain a way to inject particles across an interface with any orientation relative to the direction to the magnetic field.

First, we generate the particles exactly at the interface, with the velocity distributions in the individual directions as mentioned above. Then, we advance the particles in time by a random fraction of the time step, only this time, we have to use a propagator that includes the influence of the magnetic field.

All the particles will move the corresponding fraction of the gyration circle, some perhaps returning back across the interface. This is to be expected and these particles should be discarded for the description of the interface to be accurate.

Applicability conditions

There are certain conditions that limit the use of the above explained interface. We cannot, for example, wrap it tightly around a probe and expect it to work properly.

Perhaps the most important problem is with the boundary condition on φ. If we imagine the correct values of φ (for example in a simulation of a much larger spatial domain), the boundary, let us say with an imposed value of φ, should be placed so that the imposed value is not very different from the correct one. In the opposite case, the whole solution is influenced by this boundary condition, "drawn" in one direction or the other. We should therefore check in a simulation that the parts of the domain that are adjacent to the boundary have electrostatic potential properties similar to bulk plasma — the average value as well as derivative close to zero.

Another consideration should be the flux of particles. As we require both sides of the boundary to resemble bulk plasma, the flux of particles from the inside should also be close to that from the outside. This condition can be formulated as follows — the flux of particles to the electrodes inside the domain should be significantly lower than the flux of particles into the whole domain from bulk plasma. Because of the overall balance of the number of particles in a stationary state, this also ensures that the flux across the interface is similar in both directions.

4.2 Implementation

In this section we will discuss the details of implementation of the methods covered so far. We will go through the implementation of three plasma models that has been done for the present work. All the actual computation is implemented in the C++ programming language. Auxiliary tasks, including data processing and plotting, is implemented in Python.

The design of the programmes is quite modular and extensions can be added in a way that does not require the understanding of the whole code. This modularity is demonstrated by the fact that the same programme contains two completely different ways of solving Poisson's equation and they can be switched easily.

4.2.1 Particle models common features

There are certain aspects of a particle plasma model that are common to all cases, regardless of the dimension, force computation and other details. These are mainly related to the organization of the whole run and the structure of one time step.

Simulation structure

The overall structure of a plasma simulation is not very different from any other type of physical simulation. There are two basic parts — equilibration and measurement. The simulation starts from an initial condition that can be obtained in various ways, but usually this is not an equilibrium or a stationary state of the model itself. It takes some time for the system to settle down — equilibrate or achieve a steady state. During this transition, collection of data for averaging is obviously not possible. In certain special cases, we are interested in a dynamic process. This requires a carefully prepared initial condition, usually from a previous steady-state run.

If we are interested in equilibrium or steady-state properties, we start collecting data for averaging after equilibration is over. The length of this part of the simulation depends on the required signal/noise ratio and on the stability of the simulation — in certain cases it is not possible to have an arbitrarily long run. Samples are taken with the period of several time steps to ensure contributions from sufficiently different states. In some simulations (for example liquids, biomolecules and similar areas), it is possible to save the whole trajectory and analyze it later. In most plasma simulations this is not feasible because the number of particles is much higher and saving such data would require large storage. Instead, the quantities of interest are chosen at the start of the simulation and only their values are saved. In the presented implementation, for example, particle densities on the grid are saved instead of particle positions. In the one-dimensional code, a phase space density on a grid is saved instead of the full positions and velocities information.

The above outline is only the most simple case. There can be others, for example a parameter changing gradually during a simulation or several different phases of data collection.

Time step structure

The whole simulation is composed of individual time steps and regardless of the details, the structure of each of them is the same.

The core of one time step is to get from $\mathbf{x}(t)$ to $\mathbf{x}(t+\Delta t)$, all other information can be obtained from this trajectory. The necessary tasks are the following:

- compute the force (and thus acceleration) acting on each particle based on the positions of all particles and external fields,
- use the acceleration together with the chosen propagator to obtain positions (and possibly velocities) in the next time step,
- resolve collisions with neutral particles and the action of a thermostat (if applicable in the specific model)
- discard and generate particles as needed and
- perform measurements (only once in several time steps).

The leap-frog propagator is used in the one-dimensional model without magnetic field and the HARHA propagator is used in two dimensions with magnetic field. They can be found in 2.2.2 "Time evolution" (page 9).

4.2.2 Particle-in-cell acceleration computation

The computation of particle acceleration in a particle-in-cell model is composed of several steps.

First, we obtain particle densities on the grid. Each particle contributes to several nodes, except for the most simple case of the Nearest Grid Point scheme. The densities of all types of particles are added together, weighted by the charges of the plasma components. The result of this part is the right-hand side of the Poisson's equation on the grid.

Each particle was distributed into two (one dimension) or four (two dimensions) surrounding nodes using linear weights. Next, the Poisson's equation is solved on the grid. The resulting electrostatic potential at the grid points is differentiated numerically (see C.1 "Numerical differentiation" (page 101)) to obtain the electric field. This electric field is used to compute the acceleration of each particle, using its charge and an interpolation scheme, to get smooth transitions between the grid nodes.

The formula (C.7) was used for differentiating the potential and a bilinear interpolation from four surrounding grid points (in two dimensions) was used to obtain particle accelerations.

4.2.3 Particle-in-cell 1D code

A planar one-dimensional model was implemented. That means that the assumed symmetry is translation in a plane. This model has limited applications and does not allow the use of magnetic field. Still, there are some phenomena it can describe, including the two-stream instability. The only possible probe geometry is a planar probe.

Both Dirichlet and periodic boundary conditions were implemented. The Poisson equation was solved using the Thomas algorithm (see C.3.2 "Direct methods" (page 107)). In the case of Dirichlet boundary conditions particles were treated as described in 4.1.4 "Bulk plasma interface" (page 37). All of the previous discussion of particle-in-cell also applies.

The output from this program is saved to hard disk each n time steps, where n is chosen before the run. Some basic information about the system is saved together with phase space density on a grid. This obviously also contains (after the appropriate integration) the velocity distribution as well as particle density for each component.

This model is implemented in the "Plasma1D" programme that is enclosed with this thesis. A separate program in Python called "ProcessPlasma1D" for plotting the output is also provided.

4.2.4 Particle-in-cell 2D code

A two-dimensional model has far more options as for the geometries allowed. The assumed symmetry in this case is a translation in one dimension, so the model is represented as a slab. This model is also much more demanding computationally and optimizing its performance was a substantial part of the work involved.

The shape of the domain is a rectangular slab, two spatial dimensions are explicitly present and the thickness of the slab serves as a parameter for setting the number of particles. When computing particle densities, the density obtained on the grid must be scaled by the slab thickness so that the units are m^{-3}.

Each particle has two spatial coordinates and three independent velocities, so only the position of the particle perpendicular to the plane of the model is suppressed. There are two reasons to keep the velocity perpendicular to the plane. A very important reason is the use of an external magnetic field. As the magnetic force "mixes" all the velocity components, it is necessary to keep all of them in the model. Another reason would be the ability to free paths of the particles between collisions with neutral particles. These collisions are, however, not needed in this model.

This model is implemented in the "PES" programme that is enclosed with this thesis.

Boundary conditions

The outer boundary condition corresponds to an interface with bulk plasma and is implemented using the procedure explained in 4.1.4 "Bulk plasma interface" (page 37). A Dirichlet boundary condition is used for the electrostatic potential, as we are interested in putting electrodes in the interior of the domain. Each of the four edges is treated separately. A constant flux of particles is generated for each edge because, as has been mentioned earlier, the distribution of this number of particles along the edge will produce fluctuations in any segment along the edge.

This treatment of the interface with bulk plasma in combination with the particle-in-cell method works without any problems in all situations where the applicability conditions are met.

There is a Dirichlet boundary condition at the electrodes in the interior of the domain. The electrodes are held at a fixed potential, so this is the correct choice. Any particles that hit the probes are discarded.

Magnetic field

The model is able to describe an external magnetic field of very general properties. In fact, we only require it to share the translational symmetry of the whole model. A magnetic field constant in space as well as time was implemented, but in principle it could be exchanged for a more general field. The implementation would not be very difficult, but the computational demands would increase. The details of this performance hit were not investigated.

The influence of magnetic field is included in the model through the use of the HARHA propagator (see 2.2.2 "Time evolution" (page 9)). The specific term that rotates the velocity vector contains a rotation matrix $\mathbb{R}$ that is a function of a direction vector and an angle. For a constant magnetic field, this matrix is also constant and can be prepared before the simulation. For a more general magnetic field, this matrix would have to be updated for each time step or even for each particle.

A natural orientation for a constant magnetic field is in the plane of the model. The other case, perpendicular to the plane is not very useful, as movement of particles across the field is very limited. Intermediate cases with magnetic field at an angle to the plane of the model are possible.

Poisson solver

The solution of Poisson's equation in two dimension is substantially more computationally demanding than in one dimension. As explained in C.2 "Poisson's equation on a grid" (page 102), the corresponding matrix is a tridiagonal one with additional fringes, having five non-zero elements in most rows. The size of the matrix is also important. For a grid 500×500, the full matrix would have 62.5×10^9 elements. The matrix is, however, sparse, so only about 1.25×10^6 values have to be stored, together with some position information.

The method chosen for the solution of the Poisson's equation on the grid is LU decomposition. The successive overrelaxation used previously by our group [Š06] did not perform well [PLH07] compared to this method. Because the system is sparse and high efficiency is required, a specialized library was used — UMFPACK (see 4.2.6 "Libraries used" (page 49)).

The construction of matrix rows for various boundary conditions is explained in C.2 "Poisson's equation on a grid" (page 102). In this code, a method used previously in our group is implemented [Pek]. The Dirichlet boundary condition is represented explicitly in the matrix by a 1 on the diagonal in the appropriate row. The particle densities are added together, weighted by charges of the plasma components, divided by the dielectric constant and a -1 and scaled by the slab thickness to obtain the right-hand side of the Poisson's equation. Then, this right-hand side has to be modified to accommodate the boundary condition. This is done using two additional vectors. One is a stencil vector s that contains a 1 at positions where a Dirichlet boundary condition should be imposed and a 0 anywhere else. The other is a vector that contains the appropriate values of the boundary condition at positions where it should be imposed and undefined values elsewhere, as they are not important. The desired right-hand side R can be written, using the stencil s, the right-hand side without the boundary condition R_0 and the values of the Dirichlet boundary condition D, as

$$R = (1 - s)R_0 + sD. \tag{4.53}$$

The whole procedure explained above is performed in a single loop, obtaining the correct right-hand side from the particle densities on the grid and the desired boundary values.

This solution is used to obtain particle accelerations as explained in 4.2.2 "Particle-in-cell acceleration computation" (page 43).

This part of the code could be further improved by using a yet stronger solver. The use of multigrid methods is currently being investigated in our group [Pek]. A fully parallel solver, if adapted properly to the problem at hand, could also provide an efficiency increase.

Pair correlation function

A histogram is used to obtain the pair correlation function from a simulation. We measure the distances between particles and bin them to a histogram covering the range that we are interested in. Longer distances are discarded. After correct normalization we get a discretized PCF.

In a system that does not have periodic boundary conditions, a margin of the same width as the range of the PCF must be discarded. In this direct implementation the time complexity is $\mathcal{O}(N^2)$, as we have to measure the distance from each particle to all the others. However, most of these are discarded, because the required range of the PCF is usually much shorter than the size of the system.

Because of the often large number of particles, a more effective approach was taken in this implementation. First, all the particles are sorted into a regular grid of bins with spacing equal to the range of the PCF. The distance from each particle is then measured only to those particles that are in the nine bins centred on the bin with the particle of interest. This reduces the time complexity to $\mathcal{O}(N)$ (for a given particle density) and has allowed a significant speedup of this programme.

Collisions and temperature control

After each time step, collisions are processed and temperature control is applied. Both of these methods are optional.

Collisions with neutral particles are not the subject of this work, but a simple way to include elastic collisions was implemented for the purpose of comparison with a fluid model developed in our group. These collisions are characterized only by a collision frequency. In each event, the velocity direction of the plasma particle is changed randomly, maintaining its magnitude. There is no angular or energetic dependence.

The Andersen thermostat (see 2.2.6 "Further considerations" (page 18)) was used for some runs in order to quench a random initial distribution of particles to a desired temperature.

Data output

The whole simulation is run in the two usual phases — equilibration and data collection. Equilibration is just a number of time steps (that can be configured) that are run without any data output. During data collection, the simulation is divided into "cycles". At the end of each cycle, some instantaneous values are saved. Starting at a certain (configurable) cycle number, data are added to buffers, that are normalized at the end of the run. These result in averaged, hopefully smooth, values.

Four grid quantities are saved — particle densities for ions and electrons, electrostatic potential and charge density. Velocity distributions and radial distribution functions are saved for each plasma component and some additional information, like the numbers of particles, is also saved. Optionally, the positions and velocities of all the particles can be saved at the end of a run to be used as an initial condition for another simulation.

The data saved to hard disk are further processed by a program in Python to produce plots of averaged values as well as for individual cycles.

Parallelization

Because of the computational demands of this model, the code was parallelized. The simulation is distributed over several nodes and the Message Passing Interface (MPI) standard is used for communication (see 4.2.6 "Libraries used" (page 49)). Because the linear system solver provided by UMFPACK is not parallelized, this part of the computation is run in serial on one of the nodes.

The particle decomposition approach was used to distribute the workload across nodes. This effectively means that there are several almost independent simulations that are linked via the Poisson's equation solution. A good way to think about this is to imagine the slab of thickness d being distributed on N nodes so that each node contains a slab of thickness d/N.

The collection of charge density is modified so that the densities from all the nodes are summed in a buffer at the master node. The Poisson's equation is solved there and the resulting potential is distributed back to all the nodes to be used for force computation. The speedup achievable this way depends on the proportion of the time taken by the Poisson solver. If this proportion is small, the speedup can be almost linear on several nodes.

There are some other locations in the code where parallelization must be taken into account. This involves mainly input and output operations. Saving particle configuration, loading it back, saving data during the run, console output — all these had to be modified for the parallel implementation.

4.2.5 Boundary integral/treecode

Because the two-dimensional code has well separated modules, it was possible to exchange only the Poisson's equation solver and keep everything else in place.

The computation of forces has a substantial impact on the whole simulation. The results depend on it strongly, as does the computational efficiency.

Treecode

The algorithm used here is the one described in 4.1.3 "Boundary integral/treecode" (page 36). Before the run, the depth of the tree is set. Based on that depth, a regular grid of square bins is created and all particles are sorted into these bins. Each bin holds the positions and charges of all its particles. Then an average positive charge and an average negative charge is calculated for each bin, creating the bottom level of the tree. This is used to create the higher levels of the tree by adding together the charges of four nodes at a time, all the way to the root of the tree. All of this is done for the positive and negative charge separately, but in one data structure representing the tree.

An optimization is performed at the beginning of the run by precomputing the interaction lists for each particle bin. Each bin holds a list of nodes that satisfy the multipole acceptance criterion and a list of bins, whose particles must be accounted for directly. This interaction list is used for all particles that fall into that bin, so that there is no tree traversal when computing the potential or force. The influence of predetermined lists is included, which is significantly faster than checking the multipole acceptance criterion for each particle-node pair in each time step.

One tree is created for the particles being simulated. This is used to evaluate the potential at the boundary, which in turn permits the solution of the boundary integral equation. Then a second tree is created for the virtual charge at the boundary and the influence of this tree together with the first one provides the desired solution for the particles in the interior of the domain.

Boundary solution

The integral equation representing the Dirichlet boundary condition has been shown to reduce to a dense linear system. This system is solved by LU decomposition, as the matrix is given only by the geometry and is therefore constant during the whole run. This LU decomposition is performed using the GNU Scientific Library, which in turn uses a BLAS implementation for basic linear algebra operations.

Regularization

For reasons of numerical stability and allowable time step size, we regularize the Green's function in two dimension by the expression

$$G(\mathbf{x}|\mathbf{y}) = \frac{\log\left(|\mathbf{x}-\mathbf{y}| + \delta^2\right)}{2\pi}, \quad \mathbf{x}, \mathbf{y} \in \mathbb{R}^2, \tag{4.54}$$

where δ is a numerical parameter that has to be chosen for each simulation.

Parallelization

Because the application of the tree structure is independent for each particle, it is very well suited for simple parallelization on a shared-memory architecture. This was implemented using the OpenMP compiler extension standard.

4.2.6 Libraries used

There are several specialised libraries used in the two programmes implemented as part of this thesis. They are used in cases where they provide some functionality that would be difficult or impossible to implement by hand efficiently. They are used in a way that should not limit the flexibility of the rest of the code.

BLAS

BLAS stands for Basic Linear Algebra Subprograms. It is a standard library interface that provides basic linear algebraic operations — vector and matrix multiplications of various kinds. There are lots of BLAS implementations. GotoBLAS was chosen for this work — an implementation highly optimized for various platforms with inner loops hand-coded in assembly language by Kazushige Goto.

GotoBLAS is free for academic use and can be obtained on the web [Got].

UMFPACK

UMFPACK is a library that provides efficient solution of sparse linear systems by LU decomposition [DD97], [DD99], [Dav04b], [Dav04a]. It was chosen among several libraries of similar type [Pek]. It uses a BLAS implementation for basic linear algebra operations. GotoBLAS is used for this purpose, as explained above. UMFPACK is used to solve Poisson's equation.

UMFPACK can be downloaded and used under the terms of the GNU General Public License [UMF].

GNU Scientific Library

> *"The GNU Scientific Library (GSL) is a numerical library for C and C++ programmers. It is free software under the GNU General Public License. The library provides a wide range of mathematical routines such as random number generators, special functions and least-squares fitting. There are over 1000 functions in total with an extensive test suite."* [GSL]

The LU decomposition solver from this library was used for solving the integral equation involved in the boundary integral/treecode method. Internally, it can utilize a BLAS implementation and GotoBLAS was used for this purpose, for reasons explained above.

The GNU Scientific Library can be downloaded and used under the terms of the GNU General Public License [GSL].

MPI

The Message Passing Interface (MPI) is a standard for process communication on shared as well as distributed memory systems. It has become the de-facto standard parallelization tool in high-performance computing. MPI allows passing of data (messages) between nodes in a collective as well as point-to-point fashion. There are several implementations of this standard. The OpenMPI implementation was used for the present work. OpenMPI can be downloaded and used under the BSD license [Ope].

Matplotlib

Matplotlib is a 2D plotting library for the Python programming language. It was chosen for its output quality, the ability to export to several graphical formats and the possibility of being used from Python programs. This means that making plots in batches can be automated easily.

Matplotlib is free software and can be obtained from [mat].

4.2.7 Performance considerations

There are several ways in which we can make a programme more efficient, either with CPU time or with storage. Some of them involve algorithm design and others are of purely technical nature, related to the implementation of the algorithms. Because plasma simulations in more than one dimension are potentially very demanding, we will now look at the options available for performance increase.

Use of libraries

The use of a suitable library for a specific task can have a dramatic effect on performance. There are two main reasons for that. In the case of complicated problems, the use of advanced algorithms can increase performance, but such algorithms can be difficult to understand in detail and implement properly. An implementation by a specialist in the given area is a much better choice in such a case. An example of this case is the UMFPACK library. Another possible advantage is the quality of the code itself. Well maintained libraries that are designed for high-performance computing are usually superior to anything a single person can achieve with his own implementation. Careful optimization tailored for various architectures, perhaps with parts of the library hand-coded in assembly language, can increase performance substantially. GotoBLAS is an example of such a library.

Algorithm change

To a large extent, the choice of algorithms determines code performance. We will only mention a few points from this very large topic that are relevant to this thesis.

First, there is the issue of time step choice. Apart from setting it as large as stability and precision requirements permit, we can actually increase it substantially for certain parts of the system. It can be shown rigorously [TBM92] that an independent (and larger) time step can be used for "slow" degrees of freedom. In practice that means that some degrees of freedom are updated more often than others. This is in loose analogy with the adiabatic approximation known from quantum mechanics. Another step would be separating a group of degrees of freedom entirely (in our case the ions) and using a different "rate of time". Specifically update their positions as often as those of electrons but with a larger time step. This approach is commonly used in particle-in-cell plasma simulations. It seems to be justified only in a steady state and one must be careful to treat transitions obtained with this type of dynamics accordingly — as unphysical processes. Moreover, it relies on the mean-field character of the particle-in-cell method and can not be used directly for treecodes. This will be discussed in detail in 5.5 "Methods comparison and discussion" (page 67).

There are several other modifications that can be done, but we will not explore them here, as their responsible use would require a substantial amount of work. An example would be the use of a non self-consistent approach, in which the Poisson's equation is not solved in each step but a fixed value of the electrostatic potential is used. This is usually obtained as an averaged value, either from the same model or a different one.

The use of parallelization often requires the change of a large portion of the serial code and generally causes the code to be more complicated. On the other hand, there is a big potential for performance increase. It is the only (scalable) option one has to overcome the limitation of the speed of a single CPU core. Two different approaches to parallelization were used in the present work. The use of the MPI standard is and example of explicit parallelization — any communication between the processes must be expressed in the code. This means more work for the programmer but also more control over the programme. In certain cases, implicit parallelization is sufficient.

The OpenMP compiler extensions provide this type of parallelism — one states what should be parallelized, but does not make any specific communication calls. The scaling depends, among other things, on the hardware used and on the algorithm itself. If a large part of the workload can be distributed, scaling will be better than in the case of a large part of the code that has to run in serial. The details of parallel computing performance are, however, beyond the scope of this work.

Code optimization

The performance of the programme can be improved without changing the algorithms by improving the actual binary code that is executed. This can be achieved by writing better code, at least in cases where it is obvious that a certain code is better. Generally, the use of a good optimizing compiler can improve binary code performance. On Intel processors, the Intel Compiler increased the performance of the programmes enclosed with this thesis by up to 30% compared to the GNU C/C++ Compiler.

Profile-motivated optimization

In cases where it is not obvious that a certain formulation of the same part of the code is faster, profiling can help with the decision. This information should not be underestimated, as the performance characteristics of modern CPUs are often rather complicated and sometimes surprising. For this reason "optimization by intuition" can actually decrease performance. Profiling provides a rational framework to guide code optimization. It also provides information as to whether one should be concerned about the performance of a certain part of a programme in the first place. As the famous quotation says:

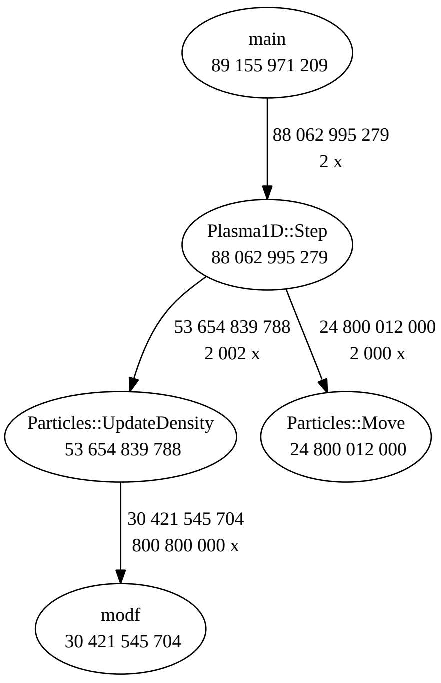
> *"We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil."* — C. A. R. Hoare

The initial design of a program should not undermine the possibility of future optimizations, but only seeing an actual bottleneck should motivate any serious code optimizations.

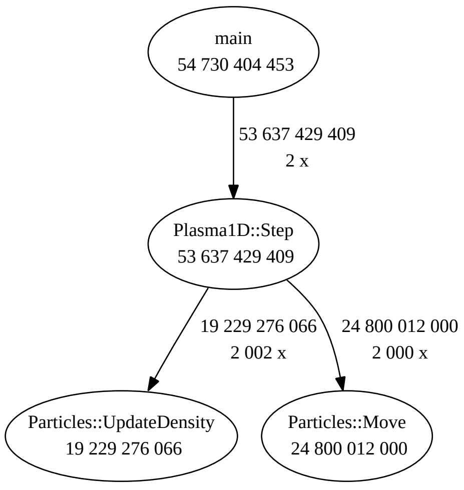
Profiling provides detailed information about the run of a programme. The specific information produced depends on the profiling software used and the methods it uses. Generally, some information about times spent in specific methods, numbers of execution of individual lines of source code, cache memory misses or similar characteristics are obtained. A cache miss occurs when data required in the CPU for processing are not found in the cache memory. This causes a slowdown, as fetching data from main memory takes longer. Profiling is only relevant on code that is compiled with the same flags as for production runs, because the resulting binary code can be very different for different compiler optimizations.

In the present work, the Valgrind framework was used [Val]. Its Callgrind tool provides detailed information about the execution of a programme at the price of a considerable slowdown. The profile created by Callgrind provides data for each line of source code. These data include instruction fetch counts (as a measure of time spent on that code line), memory read and write counts and various cache misses.

Let us now have a look at a simple example of the use of a profiler for code optimization. We will first look at the profile of a small test run of the programme Plasma1D and then try to find a bottleneck and improve it. Figure 4.3(a) shows the callgraph of the original version.



(a) original version



(b) modified version

Figure 4.3 Callgraphs for the original and modified version of Plasma1D, showing methods as nodes. The numbers in the nodes show the CPU instruction count for that node. Each arrow is marked with the instruction count and number of calls. Nodes that contribute less than 1% of the total instruction count are not included.

We can see that `Particles::UpdateDensity` takes a substantial part of the whole run. We can also see, that the majority of that is the library function `modf`. This function was the first choice, because it is "natural" for the task — we need to separate the position $x[i]$ divided by grid spacing h into its integer part and the rest. This gives the cell index and an offset that is used for interpolating the particle between two nodes. Let us now have a look at the instruction counts for individual lines in that method.

```
4,004  void Particles::UpdateDensity() {
    .
4,004      const double h = Parent.h;
```

```
         2,002      const int Ngrid = Parent.Ngrid;
             .
     5,041,036      for ( int i=0; i<Ngrid; i++ )
     5,013,008          d[i] = 0;
             .
             .      double ind;
 3,203,212,012      for ( int i=0; i<N; i++ ) {
 5,605,614,014          offset[ i ] = modf( x[i]/h, &ind );
30,421,469,628  => ???:modf (800797998x)
 2,402,400,000          index[ i ] = static_cast<int>( ind );
 6,406,400,000          d[ index[i]   ] += 1 - offset[i];
 5,605,600,000          d[ index[i]+1 ] += offset[i];
             .      }
             .
         4,004  }
```

We can see that the call of the `modf` function really is a major part of the whole method. Using this information, we can try to come up with a different way to do the same thing, hopefully avoiding a function call altogether. The modified version is listed below and the corresponding callgraph is in Figure 4.3(b).

```
         2,002  void Particles::UpdateDensity() {
             .
         4,004      const double h = Parent.h;
         2,002      const int Ngrid = Parent.Ngrid;
             .
     5,041,036      for ( int i=0; i<Ngrid; i++ )
     5,013,008          d[i] = 0;
             .
 3,203,208,008      for ( int i=0; i<N; i++ ) {
 2,402,404,004          const double tmp = x[i]/h;
   800,800,000          const unsigned int ind =
                            static_cast<unsigned int>( tmp );
 4,004,000,000          const double off = offset[ i ] = tmp - ind;
 4,804,800,000          d[ ind     ] += 1 - off;
 4,004,000,000          d[ ind + 1 ] += off;
             .      }
             .
         2,002  }
```

We can see that the new version of `Particles::UpdateDensity` is more than twice as fast, mainly due to the use of a type cast instead of the function. A small speedup is also achieved thanks to the use of local variables.

This technique has been successfully used at several other places of the programmes implemented in the present work.

CHAPTER

FIVE

Results and discussion

In this chapter, we will look at some results computed using the programmes explained earlier. These results are supposed to provide examples of the capabilities of the programmes for certain sets of parameters and one should bear in mind that there are many more possibilities as for the geometries as well as parameters.

A direct comparison to experimental results would require a careful setup that mimics the specific experimental conditions and is therefore not performed here. However, in principle it is a possible use of the programmes and could be performed. The results presented here should be regarded as illustrations of the models' features, limitations and capabilities. Simulations were usually run for a single fixed value of probe potential, as this is the way electrostatic probes are being used in tokamak applications.

The results that follow are divided into sections based on the model used. Every different setup is explained and comments on the results are provided. A complete set of plots can be found in Appendix B. No plots are presented in this chapter for practical reasons — even with a small subset of all the plots it would be hard to keep them close to the text where they are referred to. Particle densities are normalized relative to the bulk density.

Unless stated otherwise, isothermal hydrogen plasma with temperature $T_e = T_i =$ 10 eV was used. Other parameters (if relevant) will be stated in each case:

- bulk particle density (of one species) n,
- time steps for electrons and ions Δt_e and Δt_i,
- domain dimensions L_x and L_y,
- grid spacing h_x and h_y,
- slab thickness d,
- magnetic induction B,

- probe diameter r_P,
- probe potential φ_P and
- probe spacing d_P.

5.1 Particle-in-cell 1D model

The one-dimensional particle-in-cell model was implemented as a starting point and a way to test basic features of the model. Its capabilities are rather limited especially because of the geometry that is given by the translational symmetry. This means that it can represent the following situations:

- an infinite metal wall at both ends,
- an infinite metal wall at one end and a bulk plasma interface at the other or
- a periodic system.

Some results were obtained for a planar probe geometry — a metal wall held at constant potential at one end. This is not very useful as the geometry distorts the results quite severely. Here, I present results for a well know feature in plasma physics — two-stream instability in periodic boundary conditions.

Cold two-stream instability

The initial condition is a stationary solution of the Vlasov-Poisson system. Two streams of electrons going in opposite directions with uniform velocities ("cold" streams). This solution is unstable and any perturbation will reveal that. This perturbation can be either provided as a part of the initial condition or simply by the random distribution of particles in the streams — slight variations in particle densities. This has also been the case in the results presented here — there is no added perturbation in the initial condition. Phase space snapshots from this simulation are presented in Figures B.1, B.2 and B.3. We can see that the time scales of electrons and ions are so different that there is no significant movement of ions within the time of the simulation and they only provide a neutralizing background. Normal adiabatic approximation was used in these simulations, so there is no separation of timescales for electrons and ions.

The relevant parameters used were the following:

$$\begin{aligned} T_e &= 10^4\,\mathrm{K} & T_i &= 300\,\mathrm{K} \\ \Delta t_e &= 10^{-12}\,\mathrm{s} & \Delta t_i &= 10^{-9}\,\mathrm{s} \\ L_x &= 0.01\,\mathrm{m} & h_x &= 10^{-5}\,\mathrm{m} \\ n &= 10^{15}\,\mathrm{m}^{-3} \end{aligned}$$

The electron temperature is not relevant in the "cold" case. The cross section of the column has been set so that there are several hundred particles of each species in one cell.

Hot two-stream instability

I have also performed a "hot" modification of the previous simulation — each of the streams has a Gaussian velocity distribution apart from the drift. The rest of the parameters were the same as in the previous case. Phase snapshots can be found in Figures B.4, B.5 and B.6. The positions of ions were fixed, as their development is not significant on the given time scale.

5.2 Particle-in-cell 2D model

The two-dimensional particle-in-cell model is substantially more useful than the one-dimensional case. There are two main reasons for that. The geometry is more flexible — internal electrodes of arbitrary shapes can be put in arbitrary places, as long as they respect the translational symmetry of the model. Only circular electrodes were used in the present work, but the implementation can be easily modified to use any other shape. A non-trivial influence of magnetic field can be accounted for, as long as the magnetic field has the symmetry of the whole model — translational symmetry in the direction perpendicular to the plane of the model.

The results shown in the rest of this section present some example use of the model, but there are many other possibilities.

5.2.1 Bulk properties

As an initial test of the two-dimensional model, let us first examine a domain with no internal electrodes and a bulk plasma interface at the outer boundary. This allows us to check basic features and stability of the model depending on various parameters.

Uniform random distribution of coordinates and Maxwellian distribution of velocities is used as an initial condition. This is what one would expect in equilibrium bulk plasma. However, because the system is interacting, one would also expect a pair correlation function different from that of a random distribution. We can therefore expect some transition period between the initial condition and a stationary state of the model.

Simulations were performed with separated time scales and with various parameters from the following ranges (apart from those stated at the beginning of the chapter):

$$L_x, L_y \in [0.00025, 0.015]\,\mathrm{m} \qquad \Delta t_e \in [10^{-11}, 10^{-15}]\,\mathrm{s}$$
$$n \in [10^{15}, 10^{18}]\,\mathrm{m}^{-3} \qquad \Delta t_i \in [10^{-9}, 10^{-13}]\,\mathrm{s}$$
$$B = 1\,\mathrm{T}$$

Slab thickness was set based on particle density so that the total number of particles was suitable for the hardware used — usually not more than 20×10^6 particles. Grid spacing was set to resolve the Debye length with several grid points.

Snapshots of velocity distributions, spatial particle densities and all three pair correlation functions were taken at regular intervals to check the behaviour of the model. Velocity distributions for two different densities can be found in Figures B.7 and B.9. An example of an electron-ion pair correlation function for $n = 10^{15}\,\mathrm{m}^{-3}$ is shown in Figure B.8 and it is obvious that there is no correlation at all. At very high magnification, there seems to be a slight shift, but it is lower than numerical noise, its order is about 10^{-3}. Generally, all the pair correlation functions up to about $n = 10^{17}\,\mathrm{m}^{-3}$ look this way. Pair correlation functions for $n = 10^{18}\,\mathrm{m}^{-3}$ can be found in Figures B.10, B.11 and B.12. The tests generally confirm the requirements imposed on particle-in-cell simulations (see 2.3.4 "Particle-in-cell" (page 22)). There are, however, at least two interesting points to note.

For higher densities, starting somewhere between $n = 10^{17}\,\mathrm{m}^{-3}$ and $n = 10^{18}\,\mathrm{m}^{-3}$, the initial condition causes the system to heat-up due to random fluctuations in particle density. This effect has to be countered by the use of a thermostat — the Andersen thermostat in our case. After an initial period during which the pair correlation function adjusts, the system is stable without the need for a thermostat. In the simulation with $n = 10^{18}\,\mathrm{m}^{-3}$, the grid spacing was set to $h_x = h_y = 7.5 \times 10^{-6}\,\mathrm{m}$, which means that the pair correlation function is different from 1 at the length about two grid point distances.

The second point is the pair correlation function. For ideal gas, it has a constant value of one — there is no correlation at all. For an interacting system there is some structure whose details depend on the specific interaction potential and other quantities. In the particle-in-cell model we can see that for lower densities the pair correlation function looks like that of a non-interacting system. For higher densities (those that need cooling at the beginning) there is only a very slight deviation from unity. This illustrates the fact that the particle-in-cell method is effectively a mean-field approximation. We must bear this in mind when interpreting all the results of this method. Further illustration of this fact is provided by electron trajectories, several randomly chosen examples can be found in Figure B.13.

5.2.2 One cylindrical probe

The first probe configuration we will look at is a single circular probe — that is a cylindrical probe with the appropriate dimension reduction. The parameters used in these simulations were the following:

$$\begin{aligned}
L_x &= 0.06\,\mathrm{m} & L_y &= 0.01\,\mathrm{m} \\
h_x &= 2\times10^{-5}\,\mathrm{m} & h_y &= 2\times10^{-5}\,\mathrm{m} \\
T_e &= 10\,\mathrm{eV} & T_i &= 10\,\mathrm{eV} \\
\Delta t_e &= 2\times10^{-12}\,\mathrm{s} & \Delta t_i &= 2\times10^{-10}\,\mathrm{s} \\
r_P &= 3\times10^{-4}\,\mathrm{m} & \varphi_P &= \pm100\,\mathrm{V} \\
n &= 10^{17}\,\mathrm{m}^{-3} & d &= 2.5\times10^{-7}\,\mathrm{m} \\
B &= 0.5\,\mathrm{T}
\end{aligned}$$

The length of the domain was chosen larger than usual to minimize boundary influence and get information about the shape and extent of structures around the probe. The probe itself was placed slightly asymmetrically in order to estimate possible remaining boundary influence.

I performed simulations for both positive and negative probe potential. Both cases show an elongated structure in the direction of the magnetic field that is called magnetic presheath. In the positive probe potential case this structure is clearly shaped by electrons moving along the magnetic field with relatively small gyration radii. Plots of particle densities and electrostatic potential can be found in Figures B.14, B.15 and B.16. It is obvious that a smaller domain would suffice for probe at positive potential. A probe at negative potential creates a larger structure and this has to be taken into consideration when setting up a simulation. The results are shown in Figures B.17, B.18 and B.19. Figure B.20 shows the number of particles during the simulation. Data for averaging were collected from frame 100 onwards.

The magnetic presheath is a potential problem, as it extends to the boundary of the domain and it is not clear how long it would be. It is, however, clear, that we can not hope to include it completely in a simulation. If the potential in the magnetic presheath does not reach zero before encountering the boundary, as is often the case, a boundary artifact occurs. The boundary condition is set to $\varphi = 0$ and this value changes abruptly to the value in the magnetic presheath on a length of a few grid points. This can cause an artificial acceleration of particles entering the domain from the outside. It does not seem that this problem can be resolved within the present model. On the other hand, this effect may be negligible compared to the influence of the particle-in-cell approximation. This will be discussed in more detail at the end of this chapter.

5.2.3 One cylindrical probe in argon plasma

This simulation of "low-temperature" argon plasma was performed "on demand" for comparison with a fluid code being developed in our group. For this reason, the set of parameters used here was chosen to correspond to those used in the fluid model.

$$
\begin{aligned}
L_x &= 0.01\,\mathrm{m} & L_y &= 0.01\,\mathrm{m} \\
h_x &= 1.25\times10^{-5}\,\mathrm{m} & h_y &= 1.25\times10^{-5}\,\mathrm{m} \\
T_e &= 2\,\mathrm{eV} & T_i &= 300\,\mathrm{K} \\
\Delta t_e &= 10^{-12}\,\mathrm{s} & \Delta t_i &= 10^{-9}\,\mathrm{s} \\
r_P &= 10^{-4}\,\mathrm{m} & \varphi_P &= 10\,\mathrm{V} \\
n &= 10^{15}\,\mathrm{m}^{-3} & d &= 1.25\times10^{-5}\,\mathrm{m} \\
B &= 0.01\,\mathrm{T}
\end{aligned}
$$

Simple stochastic interaction with neutral particles is included as isotropic elastic scattering. This process is characterized by a collision frequency for each species. The values are:

$$\nu_e = 10^9\,\mathrm{s}^{-1} \qquad \nu_i = 5\times10^6\,\mathrm{s}^{-1}$$

The results are presented in Figures B.21 through B.26. Details of the vicinity of the probe are shown because of the small diameter of the probe. In the plots we can see that the magnetic presheath is significantly reduced and while there is a clear structure in the particle densities, no elongation is visible in the value of the electrostatic potential. There is also an interesting effect that can be seen better in the plots of the vicinity of the probe. The density of electrons decreases as they are accelerated towards the probe. There is, however, a ring of higher density. This is a purely geometrical effect, as the particles get denser due to the radial movement. The density in the sheath is then an interplay between this effect and the acceleration by the radial electric field. Very close to the probe the acceleration dominates and the density decreases again.

5.2.4 Six cylindrical probes

Arrays of Langmuir probes are sometimes used in tokamaks, it is therefore useful to be able to simulate such a situation. We will look at a relatively small array of 2×3 probes, but even this size shows the typical effects present in such a geometry.

For most of the runs the following parameters were used:

$$\begin{aligned}
L_x &= 0.015\,\mathrm{m} & L_y &= 0.0065\,\mathrm{m}\\
h_x &= 2\times 10^{-5}\,\mathrm{m} & h_y &= 2\times 10^{-5}\,\mathrm{m}\\
T_e &= 10\,\mathrm{eV} & T_i &= 10\,\mathrm{eV}\\
\Delta t_e &= 5\times 10^{-13}\,\mathrm{s} & \Delta t_i &= 5\times 10^{-11}\,\mathrm{s}\\
r_P &= 3\times 10^{-4}\,\mathrm{m} & \varphi_P &= \pm 100\,\mathrm{V}\\
n &\in [10^{15}, 10^{17}]\,\mathrm{m}^{-3} & d_P &= 0.0025\,\mathrm{m}\\
B &= 0.5\,\mathrm{T}
\end{aligned}$$

The probe dimensions, potential and spacing are taken as an example of values that are being used in experiments with a 8×8 probe array.

A set of simulations was performed with all parameters except for particle density fixed. The probe potential was $\varphi_P = 100$ V in these runs and three different density values were used — $10^{15}, 10^{16}, 10^{17}\,\mathrm{m}^{-3}$. The results are shown in Figures B.27 through B.35.

The results generally agree with those obtained for a single probe, with the addition of a "geometrical shadow" effect in the direction of the magnetic field — the internal probes are shielded by the boundary ones. The dependence of the results on the particle density is in agreement with the theoretical expectation — higher density plasma is more effective in shielding an applied potential and creates a shorter sheath.

Some results were obtained for different orientations of the magnetic field. These show the geometrical shadow as a main effect. They also illustrate the above mentioned problem with the length of the magnetic presheath. The values of particle densities and electrostatic potential in this area vary slightly depending on the distance between the probe and the boundary. An example of results from a simulation with positive probe potential and $n = 10^{16}\,\mathrm{m}^{-3}$ is shown in Figures B.36, B.37 and B.38.

There are two problems that were encountered in simulations with negative probe potential. As has been shown in the case of a single probe, the structure created by negative potential is larger than in the case of positive potential. This has to be taken into consideration. If we use domain dimensions that are sufficient for positive probe potential, we can get boundary effects that distort the results severely.

Another problem is related to the initial condition. While a uniform random distribution works well in some simulations, it can not be used here. There are areas between the probes where potential wells form and keep electrons trapped. These electrons do not escape, as they are held by the magnetic field lines and oscillate in the well. The same issue is not a problem for ions because of their bigger gyration radii. An example where both of the mentioned problems are present is shown in Figures B.39, B.40 and B.41. The electron traps are located just outside the probes along the magnetic field. If we take an empty domain as initial condition, we get a very different result, where there are no electrons in the areas of the traps from the previous case. A potential drawback of this approach is very slow convergence of the number of particles towards the end of the simulation. Theoretically, it takes infinitely long to fill the domain, as there will always be a part of the velocity distribution that is sufficiently slow not to reach the other end of the domain from the injection point. In practice, a finite tolerance is used, but the effect still has to be taken into account. Results from a simulation where both of these problems are solved are shown in Figures B.42, B.43 and B.44. A possible compromise could be using a uniform distribution everywhere except for a convex hull of the probes, or possibly a slightly larger area.

5.3 Boundary integral/treecode 2D model

The boundary integral/treecode method was used as an alternative way to compute the electrostatic interaction in the two-dimensional model. The rest of the model was the same as in the particle-in-cell case. As this is a new method whose properties are not explored very well, the focus of the present work has been on understanding the features and capabilities of this method with regard to probe diagnostics simulations.

As there is no grid in this method, any quantities we want to plot have to be evaluated at artificial grid points. This depiction can sometimes be somewhat misleading, as it does not capture fine features that may be present for example in the electrostatic potential.

5.3.1 Poisson solver

Before studying actual particle dynamics let us have a look at the boundary integral/treecode Poisson solver. As is apparent from its formulation, it provides a solution that is very different from that obtained on a grid. The solution can be evaluated at arbitrary points and includes the direct influence of charges at small distances.

Solution structure

The process to obtain the full solution that respects a required boundary condition involves three solutions — the particular solution, the homogeneous solution and the full solution, which is a sum of the previous two. To see the process in detail, we can look at these solutions for a random uniform distribution of 2000 charges and a Dirichlet boundary condition. The particular solution is depicted in Figure B.45. Its value at the boundary is used to solve the integral equation. Its solution is, in turn, used as a source for the homogeneous equation that is shown in Figure B.46. We can see that it is substantially smoother than the particular solution. This is due to the fact the there are no sources (charges) in the interior of the domain, only at the boundary.

The sum of these two solutions is the desired full solution which can be seen in Figure B.47 for our particular test case.

MAC influence

The multipole acceptance criterion (MAC) θ influences the error of the result relative to direct summation. A detailed study of the dependence of this error on θ was not performed, as the primary interest was a comparison with the particle-in-cell solver. In this case, there is a qualitative difference so the details of the influence of θ can be neglected. It is, however, useful to have at least some idea about the relevant range of this parameter. Figures B.48, B.49 and B.50 show the solutions for three different values of θ. We can see that very low values introduce artifacts into the solution. While these are probably present even for very high values of θ, their scale is negligible as can be seen from the plots. The value of δ was fixed at 10^{-6} m.

Regularization influence

As has been explained in 4.2.5 "Boundary integral/treecode" (page 48), the Green's function is regularized — shifted so that it has a finite value at zero distance. The parameter δ that is used for this regularization influences the smoothness of the solution. Because of that it also influences the allowable time step of the simulation. In the present work this criterion was used empirically to set δ. Examples of solutions for three different values of δ can be found in Figures B.51, B.52 and B.53. The range of values of φ is the same in all three figures. The value of θ was fixed at 2.

5.3.2 Bulk properties

As in the case of the particle-in-cell model, let us now examine bulk plasma properties in the boundary integral/treecode model. We can expect different behaviour because of the short-range part of the interaction that is not present in particle-in-cell.

Simulations were performed for various sets of parameters in an effort to get a basic understanding of the algorithm. Spatial densities and velocity distributions were measured to assess the stability of the algorithm and pair correlation functions were measured for comparison with particle-in-cell. In this case it is crucial to always use a thermostat until the particle correlation function stabilizes, otherwise the system will heat up.

The results presented here were obtained for the following parameters:

$$\begin{aligned} L_x &= 0.0025\,\mathrm{m} & L_y &= 0.0025\,\mathrm{m} \\ T_e &= 10\,\mathrm{eV} & T_i &= 10\,\mathrm{eV} \\ \Delta t_e &= 1 \times 10^{-13}\,\mathrm{s} & \Delta t_i &= 1 \times 10^{-13}\,\mathrm{s} \\ n &= 10^{15}\,\mathrm{m}^{-3} & d &= 2 \times 10^{-5}\,\mathrm{m} \end{aligned}$$

We can see that, unlike in the particle-in-cell case, the pair correlation functions, presented in Figures B.54, B.55 and B.56, look like something one could expect in an interacting system. There is a clear depletion for pairs that repel each other and an increase in concentration for attracted pairs. As was mentioned in 2.2.5 "Pair correlation function" (page 18), the pair correlation function is directly related to the thermodynamics of the system. The velocity distribution was measured to see any possible heating of the system.

Several randomly chosen examples of electron trajectories obtained with the boundary integral/treecode method in a magnetic field are shown in Figure B.57. The obvious difference between these and those obtained with particle-in-cell will be discussed in the following section.

There are, unfortunately, several issues with this method that make it, in its present form, unsuitable for the application to probe diagnostics simulations. The least of the problems is its lower computational efficiency. This is given by two factors — the algorithm itself is more demanding and the present implementation is parallelized in a simple way. Another issue is the dependence of the maximum allowable time step on the regularization parameter δ. There is also an apparent boundary effect in the presence of magnetic field — a depletion zone along the edge of the domain parallel to the magnetic field, as can be seen in Figure B.58. It is not obvious how to change the particle injection scheme to avoid this problem.

Probably the biggest problem is the breakdown of the separation of time scales approach. Although it seems to work very well in particle-in-cell simulations, with the boundary integral/treecode method it causes heating of the system and a subsequent negative drift in the number of particles. The effect varies with the ratio of the time step sizes as well as with the details of the structure of one time step, but seems to be present in some form in all cases that do not use a correct 1:1 time stepping. Overcoming this problem is crucial for reaching useful time scales in systems that include both electrons as well as ions. It is not obvious from the present work whether this can be achieved.

None of these problems were studied in a systematic way, as this work entered the use of treecodes with boundary conditions as a new area that needs to explored. No published applications to the problems that motivated this work are known to the author.

5.4 Performance

To provide a general idea of the computational demands of the programmes, let us have a look at some examples of code performance. Specifically, we will look at the time it takes to complete one time step. In the case of the particle-in-cell method, this will be broken down into the Poisson solver contribution, which does not run in parallel, and the contribution of processing particles, which does run in parallel. Only the two-dimensional code will be discussed, as it is substantially more demanding than the one-dimensional one. This is by no means a proper benchmark, but rather a rough estimate of the performance of the programme. The values were obtained by averaging several tens of values from different time steps, to suppress fluctuations a bit, but a more through procedure would be necessary to get really reliable results.

Although direct comparison of the particle-in-cell method and boundary integral/tree-code method is not sensible, as the structure of the solution is different, we can say that particle-in-cell is considerably faster for problems of similar size.

Two different computers were used to obtain most of the results presented in this work. One of them is a desktop computer with 4 GB of RAM and an Intel Core2 Quad CPU with each of the four cores clocked at 2.4 GHz. The Intel C/C++ Compiler was used on this machine. For the sake of simplicity, this system will be referred to as C2Q. The other system is a Sun Fire V40z server. It has a Non-Uniform Memory Architecture with 4 AMD Opteron CPUs, each of them having 4 GB of memory. Due to the memory architecture, a total of 16 GB of RAM is available to the whole system. Each of the Opteron CPUs has two cores clocked at 2.4 GHz. The GNU C/C++ Compiler was used on this machine. This system will be referred to as Sun Fire.

Two sets of data are provided for the particle-in-cell method. Figure 5.1 shows the durations of the whole time step as well as the solver, as this is the part that is not parallel. This measurement was performed with a 500×500 grid and 10^7 particles, which gives 40 particles per cell, a somewhat higher but sensible number. It is interesting to note that four cores are actually slower than two cores on the C2Q. The values are so close that it can be a statistical error in such a simple study, but it can well be a real effect of the architecture of the C2Q system. It is also obvious (and not surprising) that Core2 is significantly faster as for single core performance. Figure 5.2 shows timings only for the solver on both of the systems used.

This measurement can help us assess the possibility of utilizing the current model in three dimensions. Simply put, it would not be very practical. Even a relatively coarse grid of $100 \times 100 \times 100$ points would represent the last data point in Figure 5.2 as for the number of grid points. Moreover, the matrix for a tree-dimensional system is “less sparse”, having two sets of fringes instead of one. While it is technically possible to use the current solver in three dimension, it would be much better to implement a fully parallel solver to be able to treat problem of useful size.

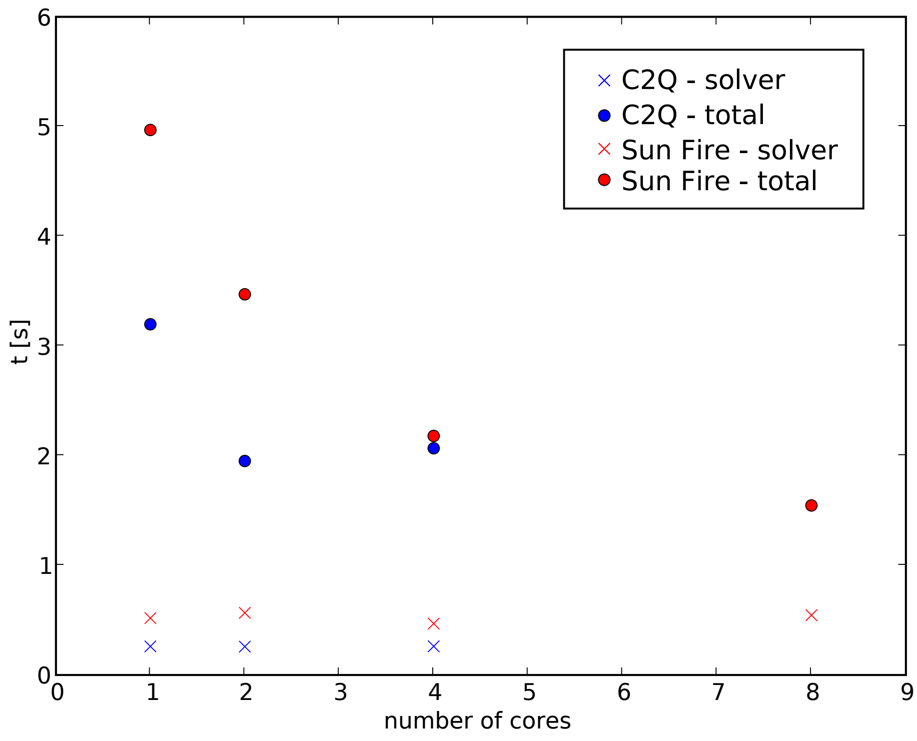


Figure 5.1 Particle-in-cell time step duration as a function of the number of cores used for the two computers that were mostly used for this work. Total time as well as Poisson solver time is shown.

The boundary integral/treecode running time is dominated by the evaluation of the action of the tree on each particle. The timings presented in Figure 5.3 are for 125000 particles of each species. We can expect the time to grow roughly linearly with the number of particle because of the major contribution of the evaluation. For higher numbers of particles, the building of the tree will get gradually more significant. This method was run only on the C2Q system, as a compiler providing OpenMP was not available on the Sun Fire system. The scaling is obviously worse than linear. As the part of the algorithm can be really divided into independent parts, we have to attribute this decrease in per-core performance to the architecture of the Core2 Quad CPU. This is indeed the expected behaviour of such a system that can be observed even for entirely independent runs.

The allowable range of parameters is determined by the length of the whole run, that usually comprises several tens of thousands of time steps. For the purpose of this thesis, runs up to one week long were performed, but longer runs, several weeks perhaps, can still be practical, if the results are worth the computer time.

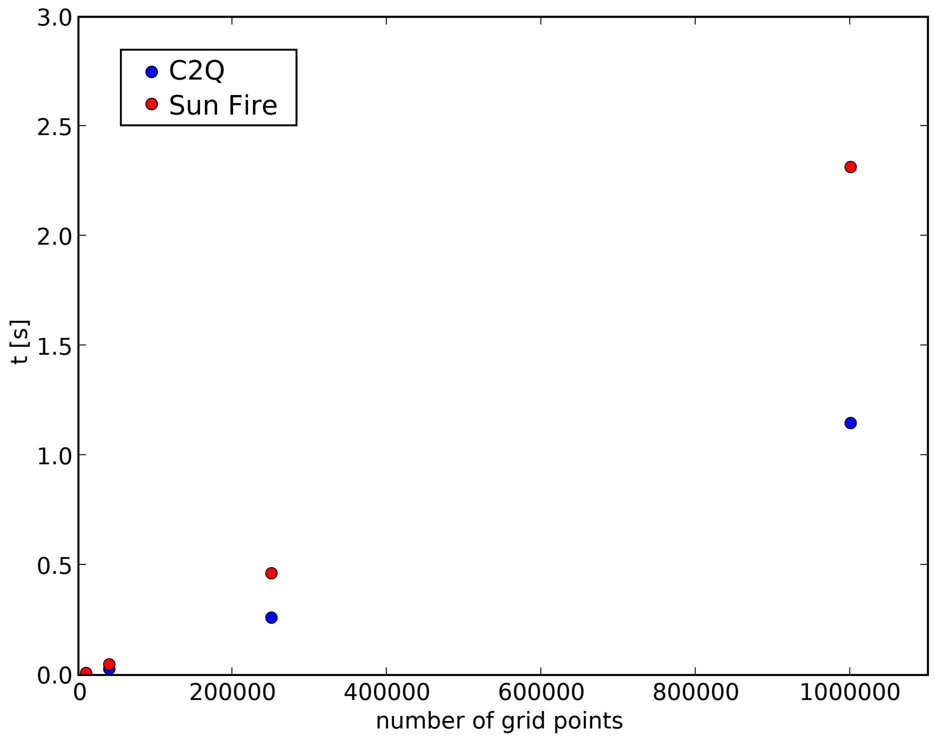


Figure 5.2 Particle-in-cell Poisson solver time step duration as a function of the number of grid points for a two-dimensional grid.

5.5 Methods comparison and discussion

A direct comparison of the two methods used would be difficult, as they represent quite different approaches. We can, however, look at the strengths and weaknesses of both of them.

The particle-in-cell method is a widely used approach to plasma simulations. It works well, is thoroughly explored and has been used in various applications. The present implementation is partially parallelized and there is space for improvement — the Poisson solver. It is obvious from the results obtained in the present work that particle-in-cell has the character of a mean-field approximation. This can be seen from the pair correlation functions, especially in contrast with the treecode results. This approximation certainly influences certain aspects of the results, but it is hard to say how much without a comparison with more precise results. On the other hand, the separation of time scales that is possible thanks to that approximation helps achieve results for problems that would otherwise be problematic or impossible.

The boundary integral/treecode method is not as fast as particle-in-cell, but this is certainly something that could be improved by a better, fully parallel implementation. The better accuracy of this method could be expected based on its formulation and is

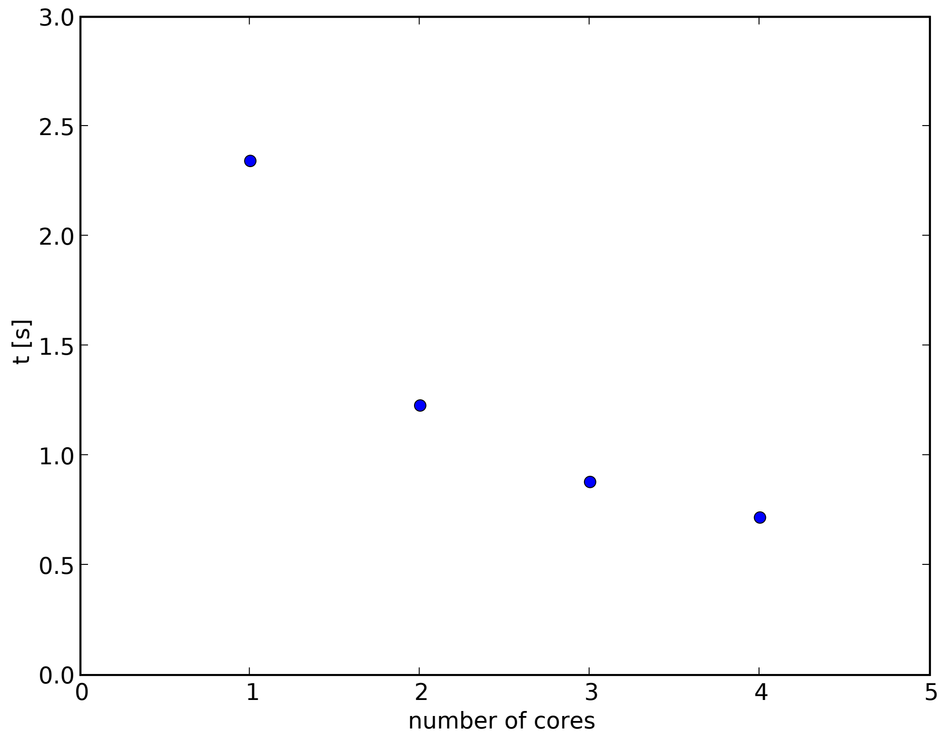


Figure 5.3 Treecode time step durations as a function of the number of cores used on the C2Q system.

well illustrated by the results presented in this chapter. However, there are at least two problems that limit the practical use of this method severely. As has been explained, the separation of time scales does not seem to work well and the injection of particles causes problems in combination with magnetic field. More work is needed to explore the options available to overcome these problems. As this is quite a new method generally and a completely new method for probe diagnostics, it would be premature to draw any definitive conclusions.

The contrast between particle trajectories obtained with the two methods further supports the claim that the particle-in-cell method is a mean-field approximation. The treecode trajectories are certainly closer to the behaviour of real plasma, although there are obviously inaccuracies caused by other features of the model, notably the dimension reduction. Based on the character of the trajectories, we can say that the diffusion across magnetic field is too low in the particle-in-cell model. The sharp edges of the magnetic presheath present in particle-in-cell methods would probably not be present for a more accurate method. The overall character of the trajectories is similar to a system where collisions with neutral particles are present. Therefore, the lack of this behaviour in particle-in-cell is probably hidden, to a certain extent, in low-temperature plasma models that use particle-in-cell and stochastic collisions with neutrals.

CHAPTER

SIX

Conclusions and outlook

Particle plasma modelling is a very wide field of study and there are many problems to be solved. This thesis has focused on a particular area — electrostatics-based simulations. The main tool developed during the work is an implementation of a plasma model that includes two different electrostatic solvers — particle-in-cell and boundary integral/treecode. This programme was used to obtain bulk plasma properties and probe diagnostics data. Substantial differences between the two solvers, as for their capabilities as well as results' properties, were found. Both methods were parallelized, as their computational demands are quite high. The presented results show the capabilities of the methods and suggest areas of possible improvement.

Looking back at the objectives of the thesis, it can be said that they were mostly achieved, although there are a few caveats.

The two-dimensional model was designed successfully and works as expected under the conditions of interest. The particle-in-cell method works without any major problems. The problems that appear in the results and are discussed in the thesis are intrinsic to the method. The boundary integral/treecode method was implemented, but several problems were encountered. The problem with the injection of particles in combination with magnetic field would limit the use of the model to non-magnetized plasma. Moreover, it is possible that it can be solved by redesigning the particle injection scheme. On the other hand, the problem with the use of the separation of time scales makes this part of the model in its current state unsuitable for plasma probe diagnostics simulations of any kind.

The performance of the programme was an important consideration. Serial performance was improved using profiling and both methods were partially parallelized. In the case of the particle-in-cell method, the parallelization using MPI allows the use of the programme on a distributed memory architecture. The current limiting factor is the Poisson solver that is not parallelized, due to the library used. The boundary integral/treecode method is parallelized in a more simple way that did not require any substantial changes to the code.

A variety of results were obtained that illustrate the capabilities and limitations of the models. Because of the problems with the treecode that were mentioned, no probe configurations were computed with the boundary integral/treecode method.

There are several areas open for future work. The particle-in-cell implementation can be extended with other methods, for example a more elaborate way to treat collisions with neutral particles, including inelastic processes. The Poisson solver could be exchanged for a stronger, ideally parallelized, one. This is also the only feasible way to approach problems in three dimensions. The boundary integral/treecode has ample opportunities for more work. The features of this method could be studied in a systematic way for situations where it is known to work. Further, solving the problems encountered and explained in this work would enable the application of this method to a wider range of problems, including probe diagnostics in high-temperature plasma.

APPENDIX

A

Computer code guide

There are two computer programmes enclosed with this thesis — Plasma1D and PES. They are both written in the C++ programming language and were run under the GNU/Linux operating system. The SCons construction tool [SCo] is used to build them, but this can be easily exchanged for some other tool. A makefile for one specific machine is also provided with PES. Porting the code to different operating system should not be a problem, provided that all the libraries are available. Although the code provided with the thesis represents the latest versions at the time of the completion of the thesis, it is recommended to contact the author at `ondrej.marsalek@matfyz.cz` to check if there is a newer version available, if you want to use any of the programmes. Enquiries about the code will be answered at the same address.

A.1 Plasma1D

Plasma1D is the implementation of the one-dimensional model. It is not very complicated and does not require any libraries. A C++ compiler is enough to build the programme.

All the configuration is done in the file `src/config.cpp` and obviously the programme has to be recompiled to reflect any configuration changes. Data are written for every frame, the number of frames and number of time steps per frame can be specified. Phase space densities and some information about each frame are saved. These data can be plotted using the provided programme ProcessPlasma1D or any software capable of displaying grid data.

A.2 PES

PES stands for Plasma ElectroStatics. It is the implementation of the two-dimensional models, both particle-in-cell and boundary integral/treecode. It is larger and more complicated the the one-dimensional code and requires several libraries. Specifically, to build PES the following libraries are needed — GNU Scientific Library, an MPI implementation (for example OpenMPI), a BLAS implementation (for example GotoBLAS) and UMFPACK. All of these are introduced in 4.2.6 "Libraries used" (page 49). A C++ compiler is needed for the build and OpenMP support is required to run the treecode solver in parallel.

Most of the configuration of PES is done in a text file that is in the same directory as the executable. The default filename is `config.txt`, but this can be changed easily in the source code. The choice of the type of geometry has to be done in the source code by creating the appropriate object in `src/main.cpp`. There are classes prepared for a domain without probes, a single circular probe an an array of circular probes. They are created by inheriting from the `BaseDomain` class and additional geometries can be easily created in a similar way. The parameters for probes can be set in the configuration file. A sample configuration file is provided with all the parameters explained.

The particle-in-cell version can be run in parallel using MPI or in serial as a normal programme. The start of the MPI run depends on the MPI implementation used and the architecture of the system. On a shared memory architecture, this will usually be `mpirun -np N ./PES`, where `N` is the requested number of processes. Data input and output takes place only on the master node.

The treecode version will refuse to run when more than one instance is invoked using MPI. If an OpenMP compiler was used, the number of concurrent threads for the solver can be set with the environmental variable `OMP_NUM_THREADS`.

APPENDIX

B

Results plots

All the plots shown here are related to material presented in Chapter 5 "Results and discussion" (page 55). More details can be found in the appropriate sections of that Chapter.

B.1 Particle-in-cell 1D model

Cold two-stream instability

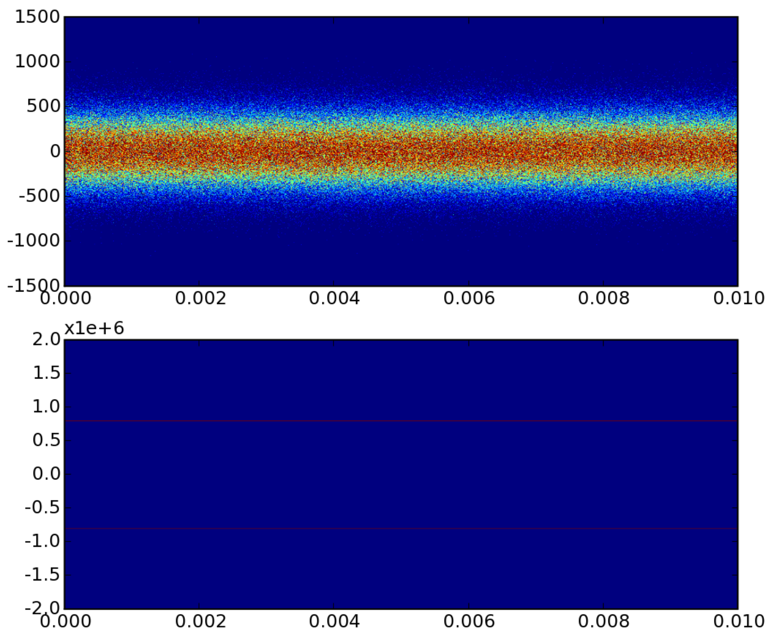


Figure B.1 Cold two-stream instability at $t = 0\,\mathrm{s}$.

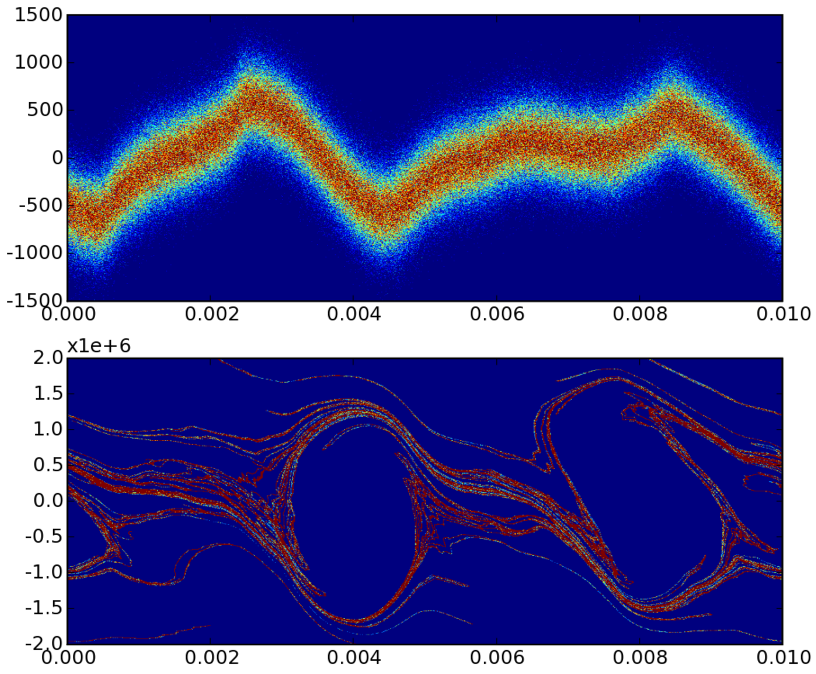


Figure B.2 Cold two-stream instability at $t = 2.3 \times 10^{-8}$ s.

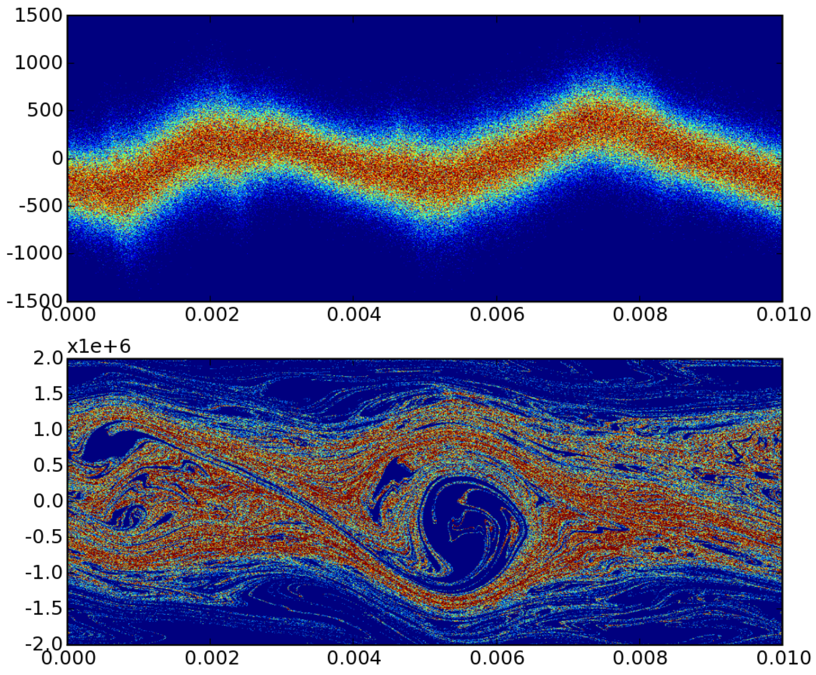


Figure B.3 Cold two-stream instability at $t = 8.425 \times 10^{-8}$ s.

Hot two-stream instability

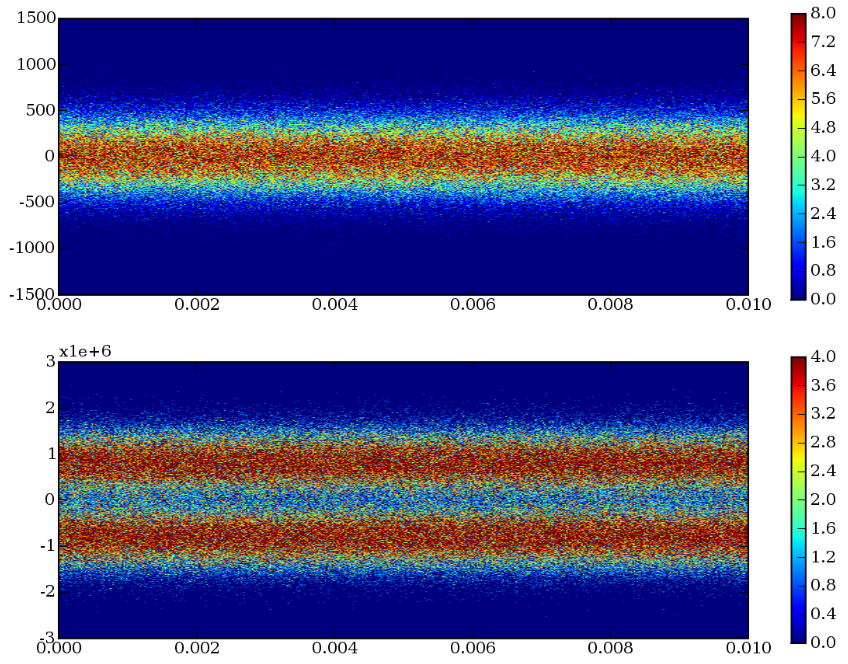


Figure B.4 Hot two-stream instability at $t = 0\,\mathrm{s}$.

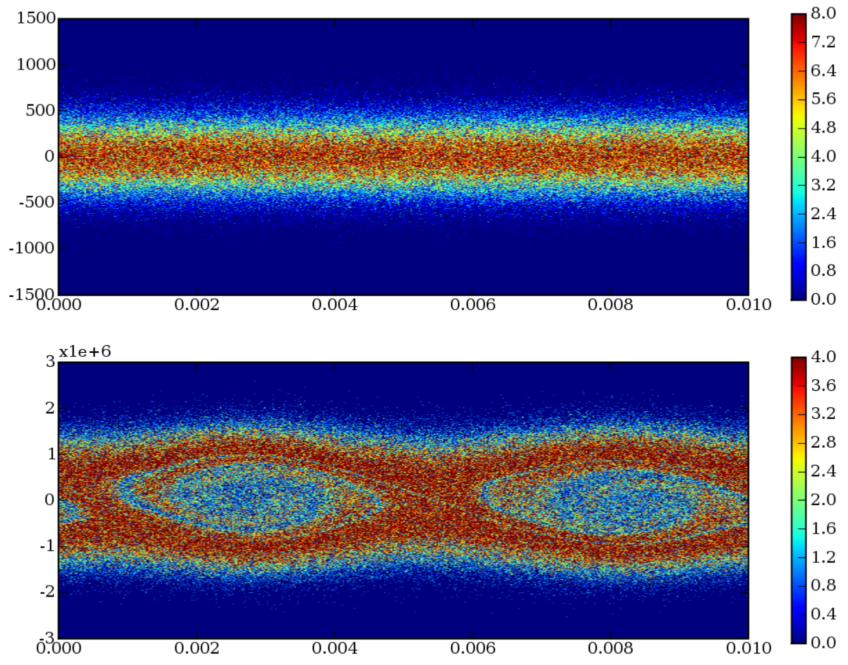


Figure B.5 Hot two-stream instability at $t = 1.9 \times 10^{-8}$ s.

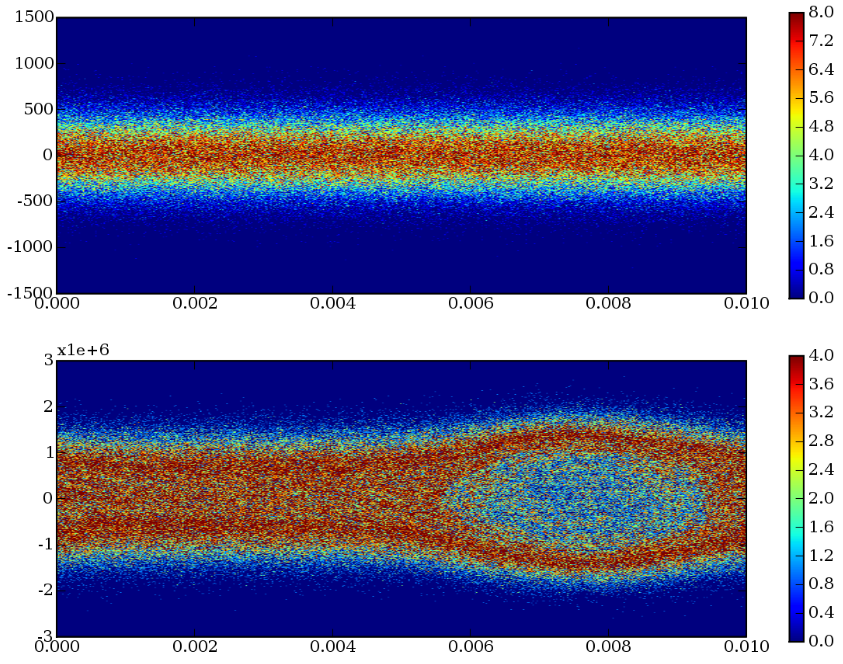


Figure B.6 Hot two-stream instability at $t = 1.13 \times 10^{-7}$ s.

B.2 Particle-in-cell 2D model

B.2.1 Bulk properties

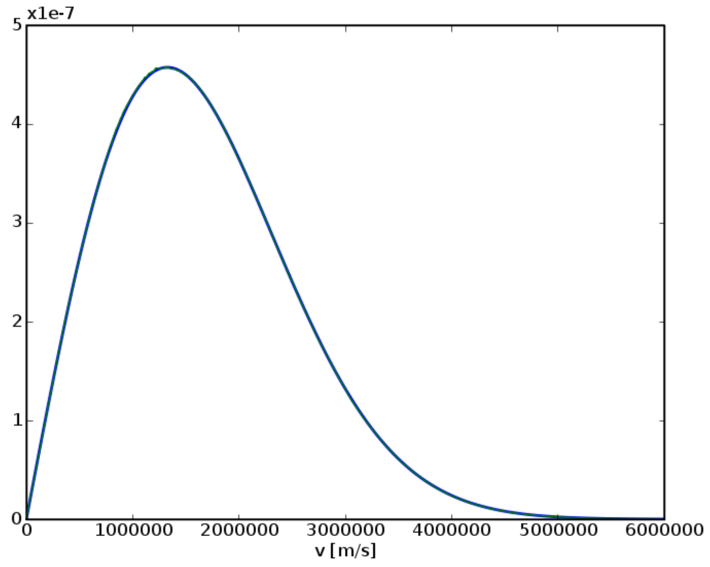


Figure B.7 Averaged velocity distribution together with a theoretical curve for $n = 10^{15}$ m^{-3}.

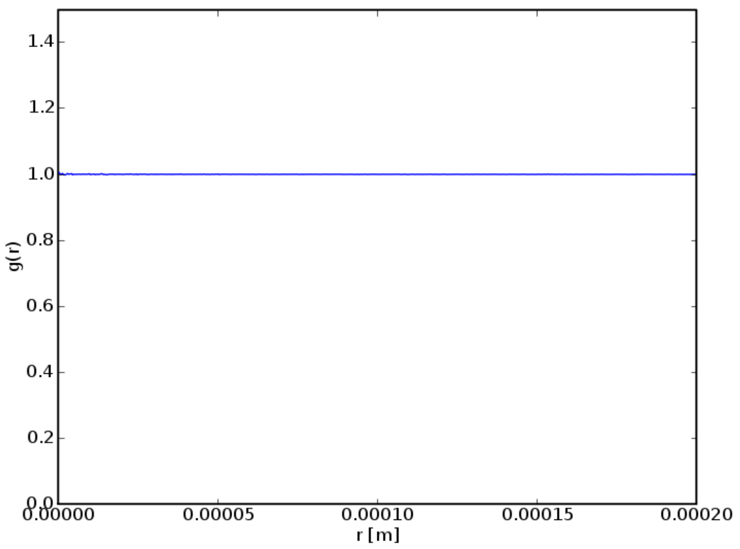


Figure B.8 Electron-ion pair correlation function for $n = 10^{15}$ m^{-3}.

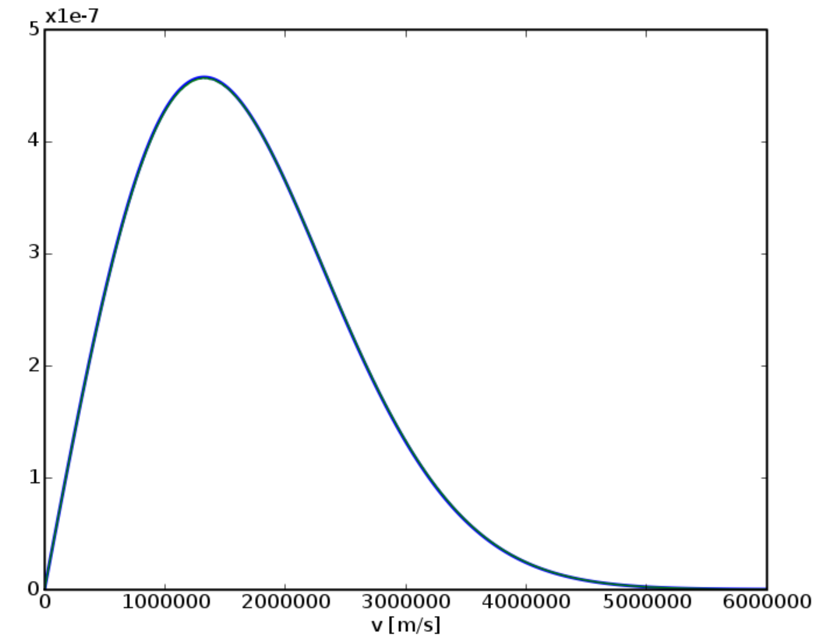


Figure B.9 Averaged velocity distribution together with a theoretical curve for $n = 10^{18}\ \mathrm{m}^{-3}$.

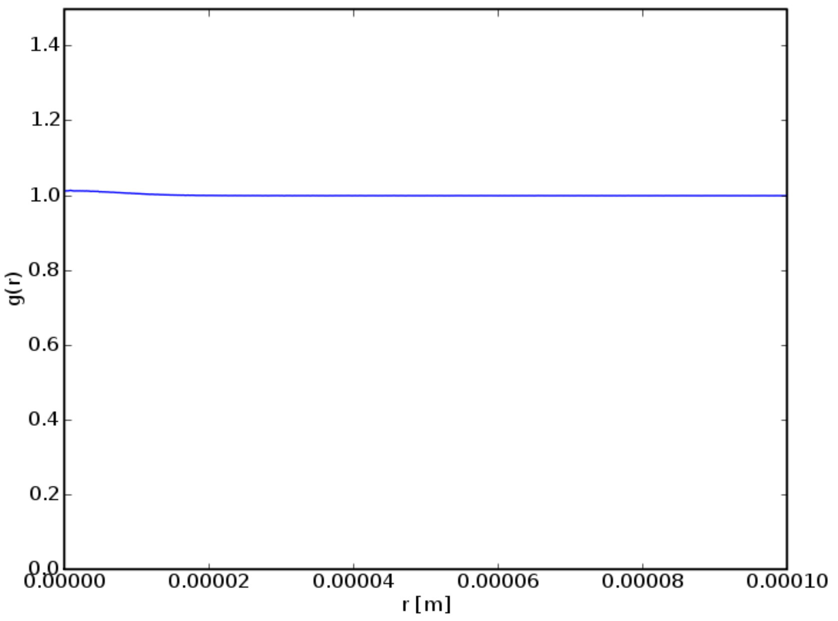


Figure B.10 Electron-ion pair correlation function for $n = 10^{18}\ \mathrm{m}^{-3}$.

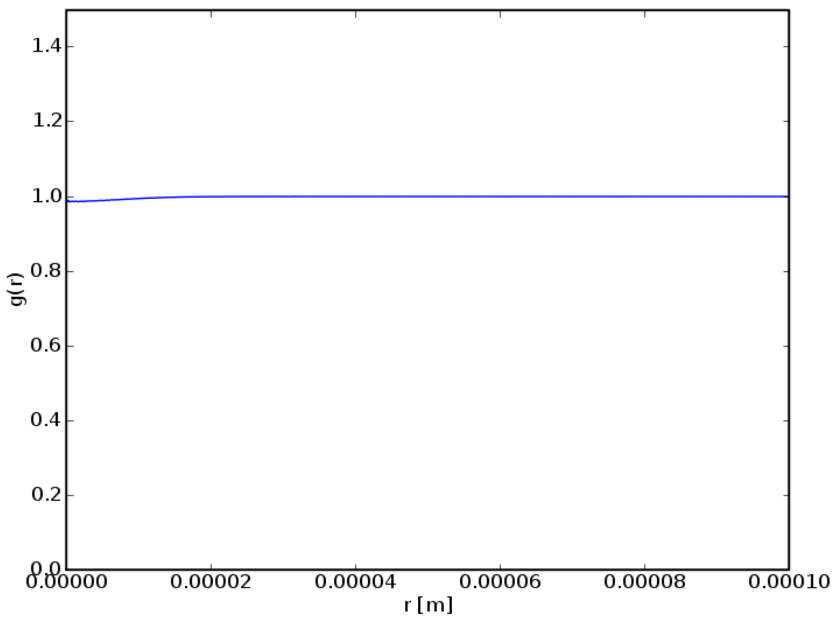


Figure B.11 Electron-electron pair correlation function for $n = 10^{18}$ m^{-3}.

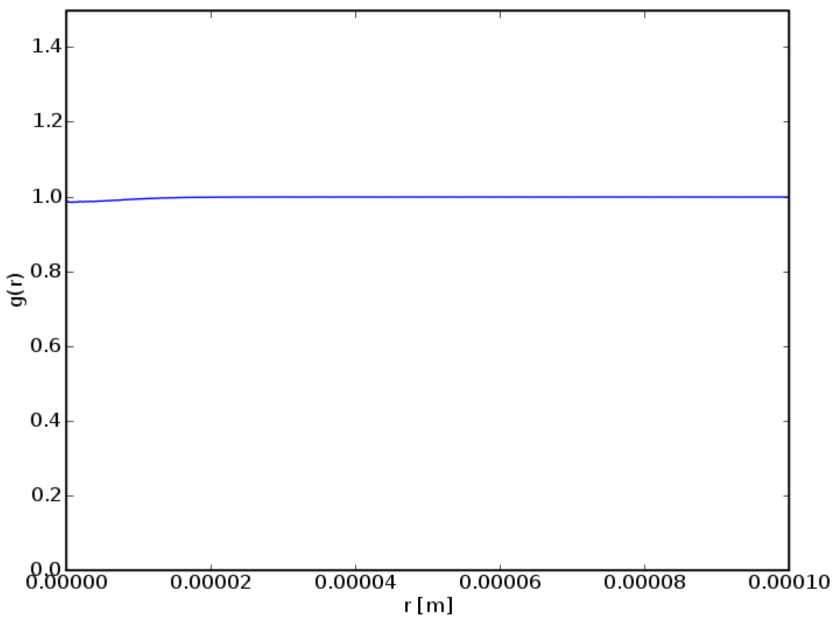


Figure B.12 Ion-ion pair correlation function for $n = 10^{18}$ m^{-3}.

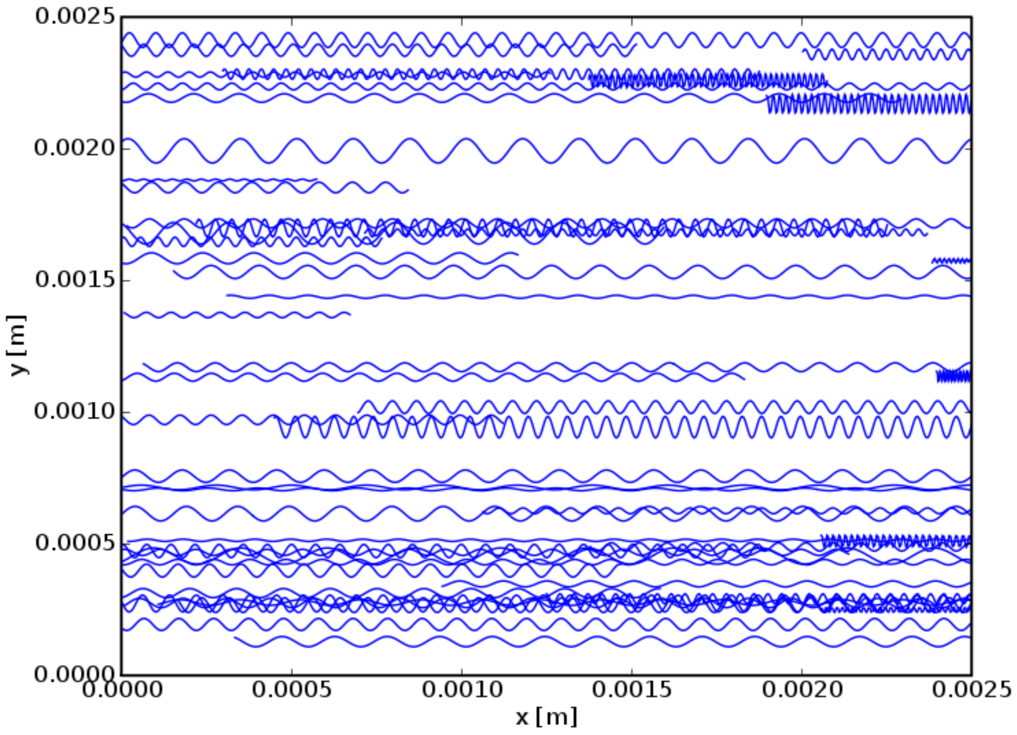


Figure B.13 Randomly chosen electron trajectories for $n = 10^{15}\ \mathrm{m}^{-3}$.

B.2.2 One cylindrical probe

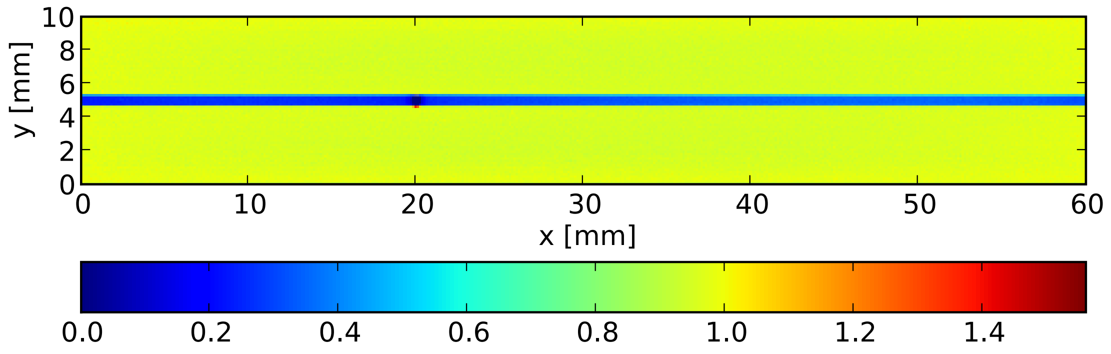


Figure B.14 A single probe at positive potential — electron density.

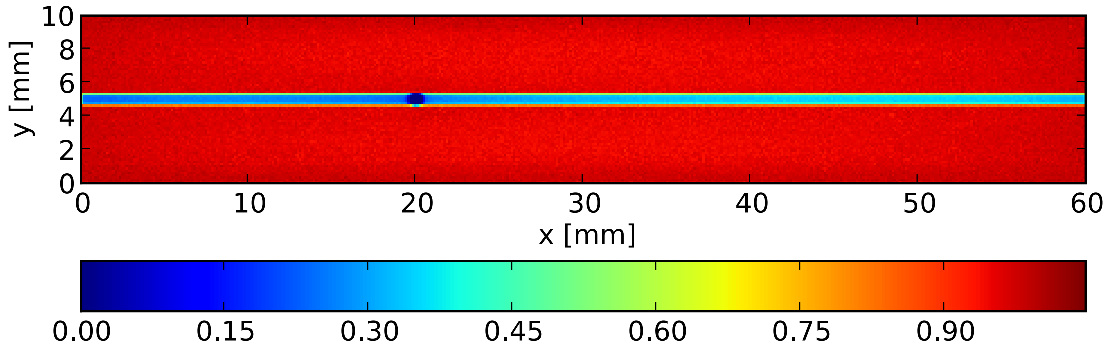


Figure B.15 A single probe at positive potential — ion density.

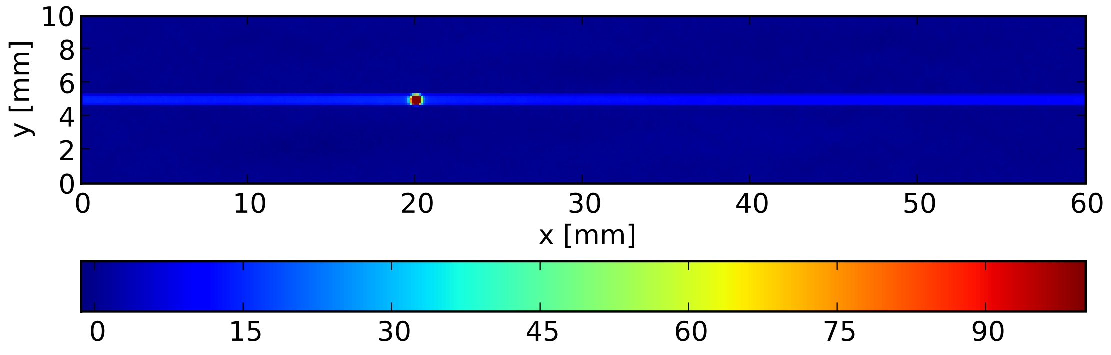


Figure B.16 A single probe at positive potential — electrostatic potential.

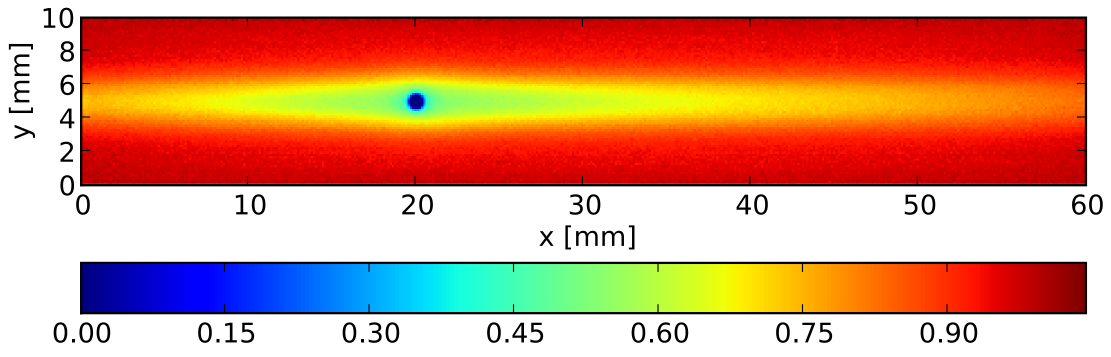


Figure B.17 A single probe at negative potential — electron density.

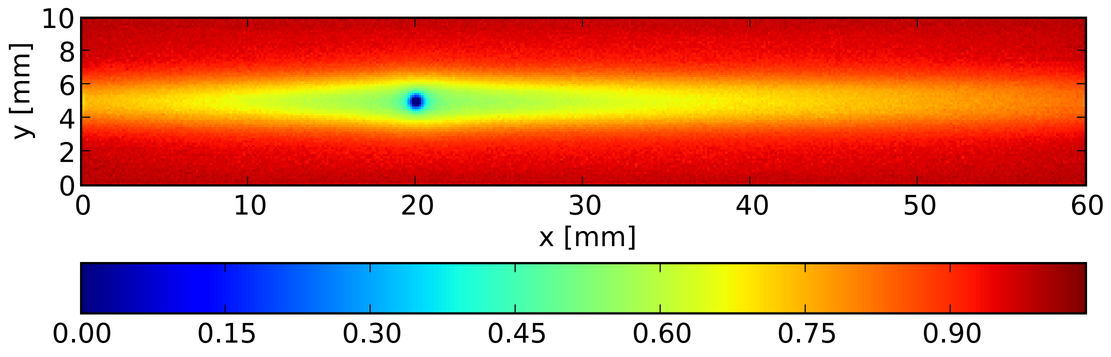


Figure B.18 A single probe at negative potential — ion density.

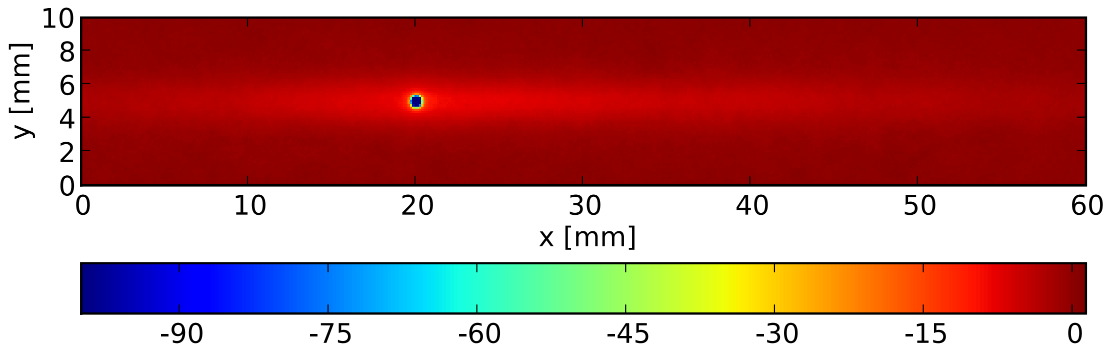


Figure B.19 A single probe at negative potential — electrostatic potential.

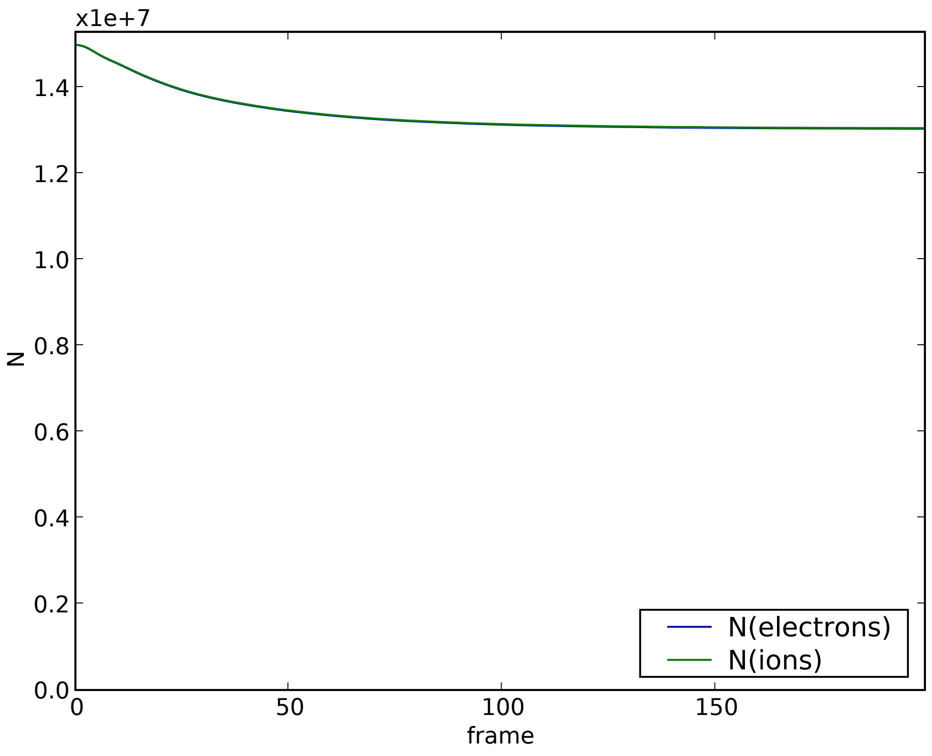


Figure B.20 A single probe at negative potential — number of particles. One frame is 6×10^{-10} s for electrons and 6×10^{-8} s for ions.

B.2.3 One cylindrical probe in argon plasma

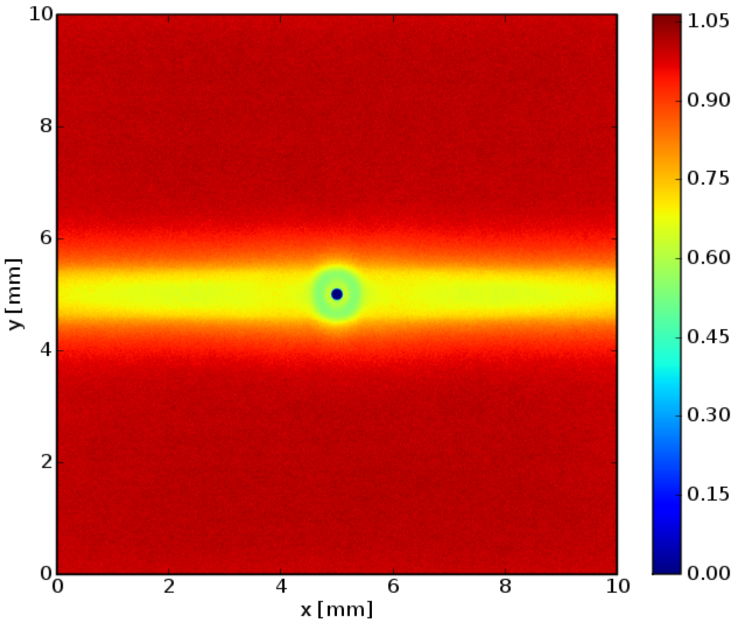


Figure B.21 Low-temperature argon plasma — electron density.

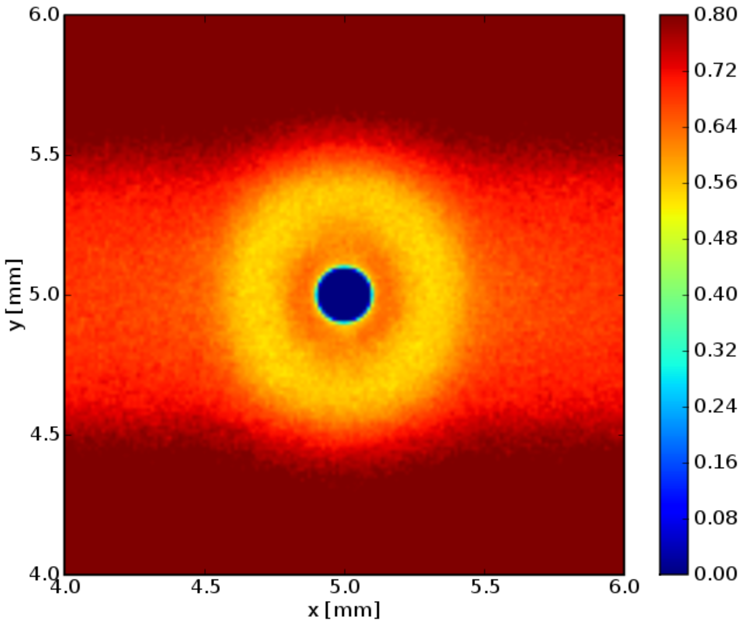


Figure B.22 Low-temperature argon plasma — electron density detail.

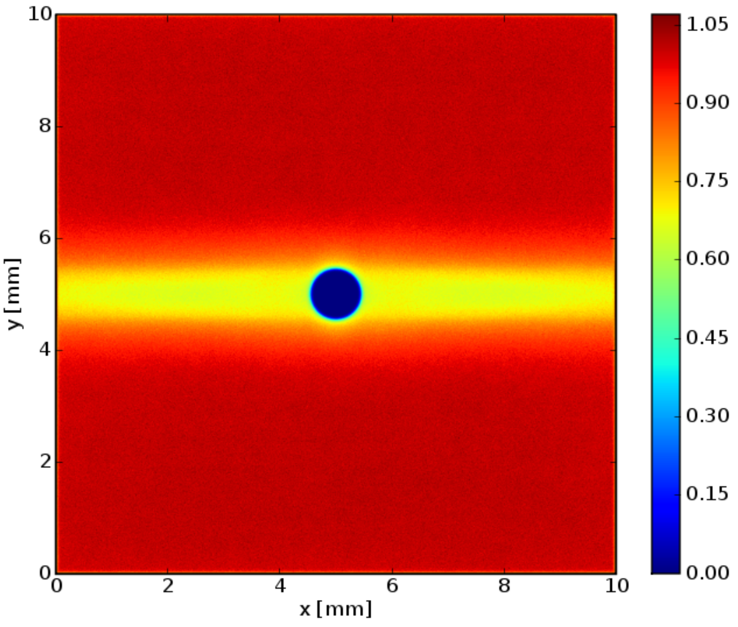


Figure B.23 Low-temperature argon plasma — argon ion density.

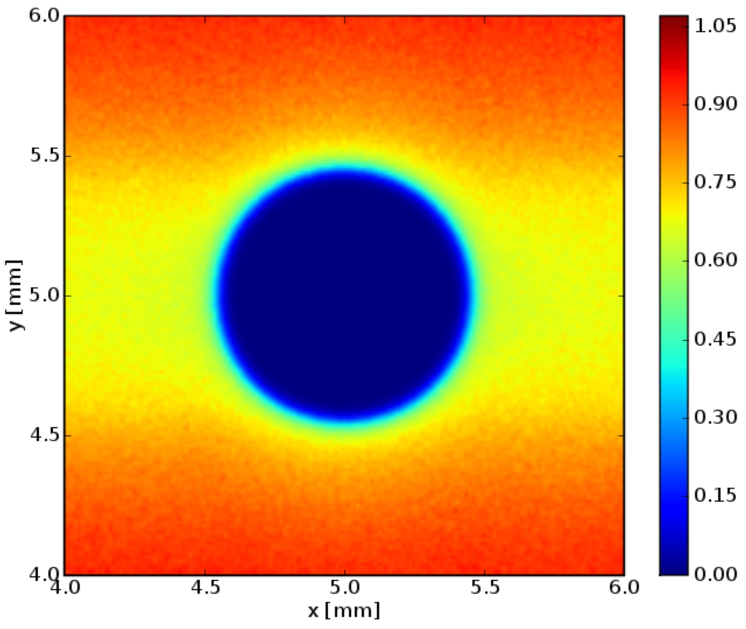


Figure B.24 Low-temperature argon plasma — argon ion density detail.

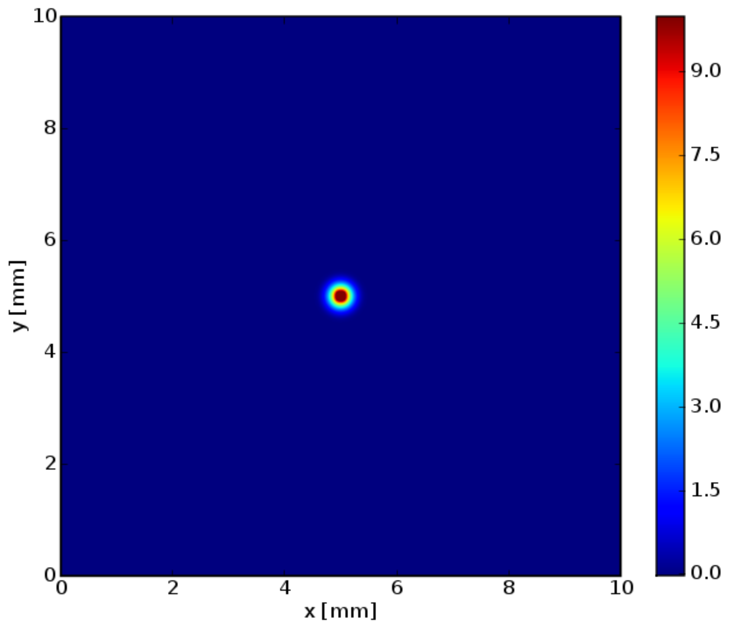


Figure B.25 Low-temperature argon plasma — electrostatic potential.

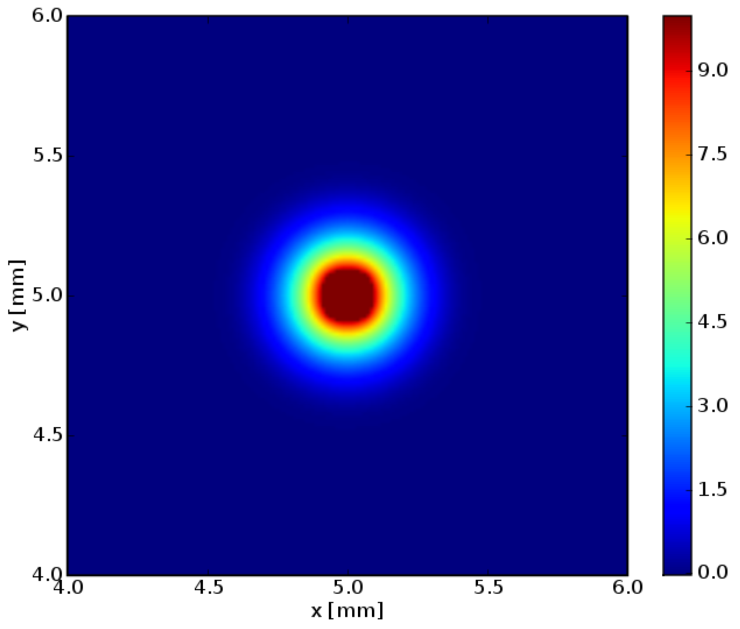


Figure B.26 Low-temperature argon plasma — electrostatic potential detail.

B.2.4 Six cylindrical probes

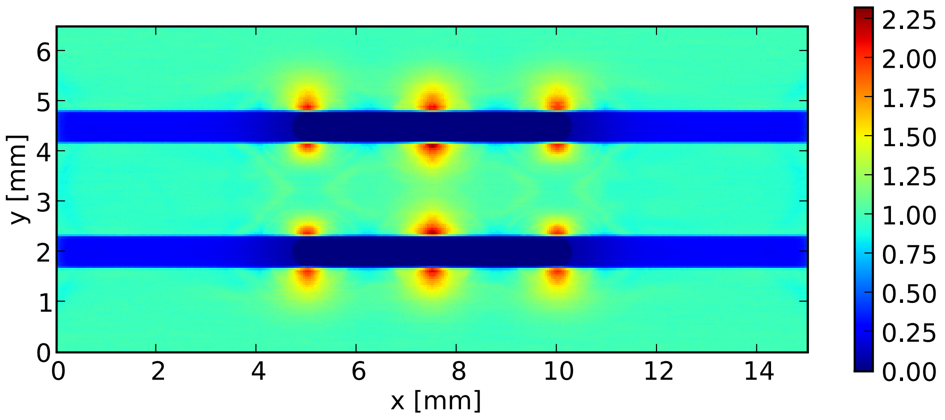


Figure B.27 Probe array, $n = 10^{15}$ m^{-3}, electron density.

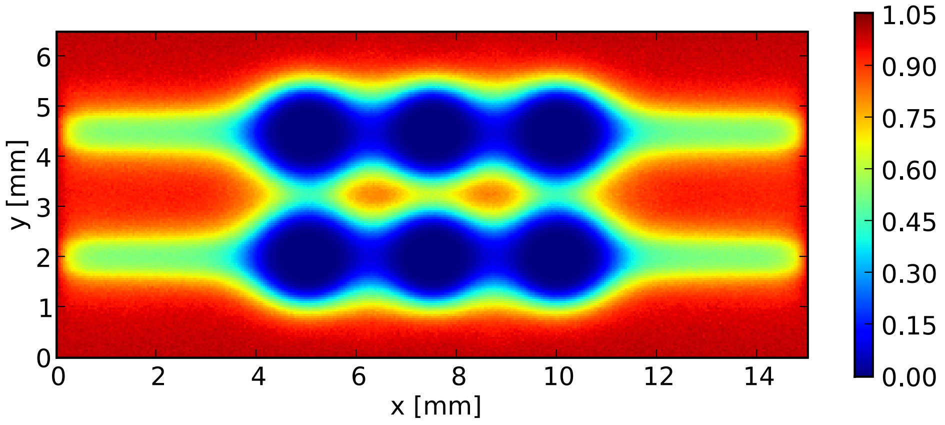


Figure B.28 Probe array, $n = 10^{15}$ m^{-3}, ion density.

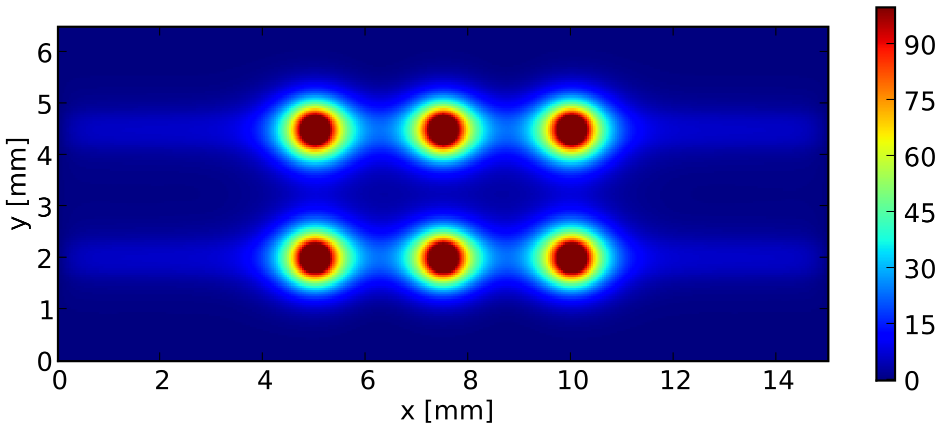


Figure B.29 Probe array, $n = 10^{15}$ m^{-3}, electrostatic potential.

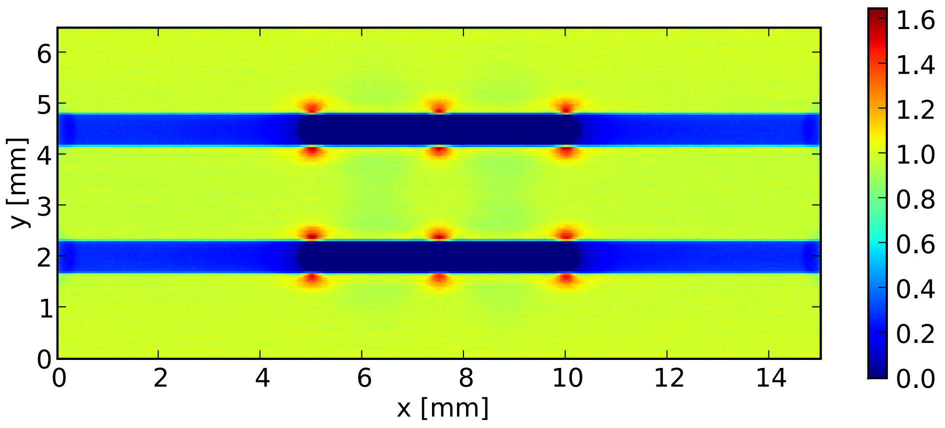


Figure B.30 Probe array, $n = 10^{16}$ m^{-3}, electron density.

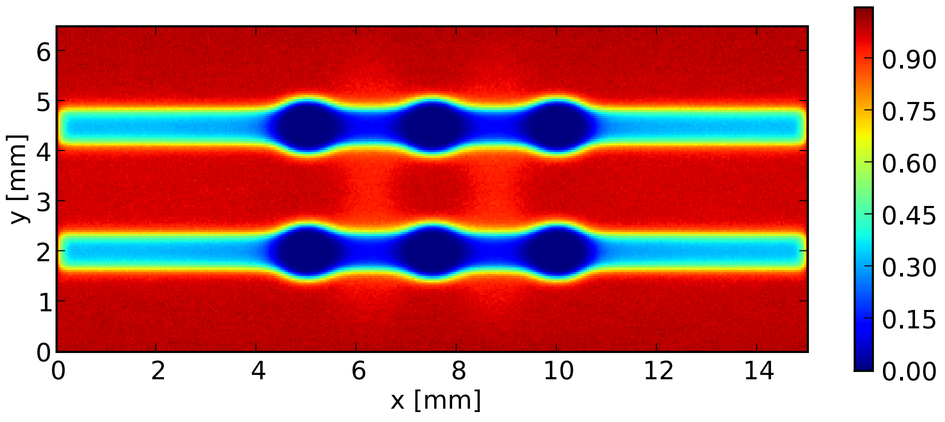


Figure B.31 Probe array, $n = 10^{16}$ m^{-3}, ion density.

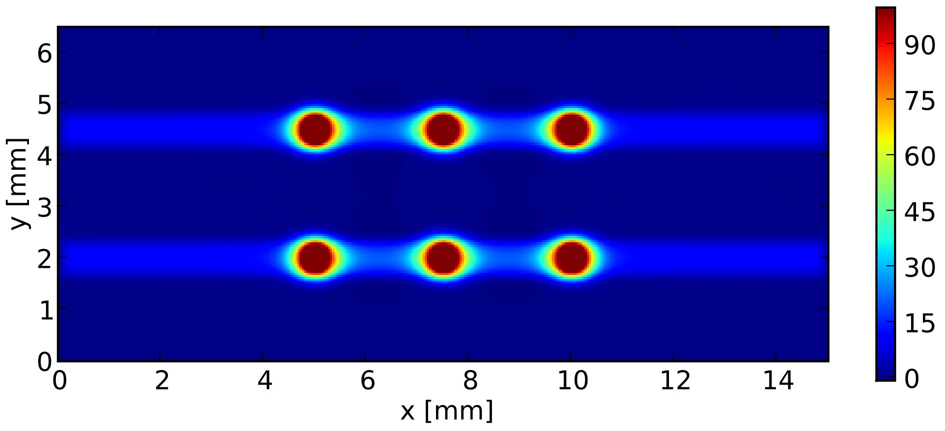


Figure B.32 Probe array, $n = 10^{16}$ m^{-3}, electrostatic potential.

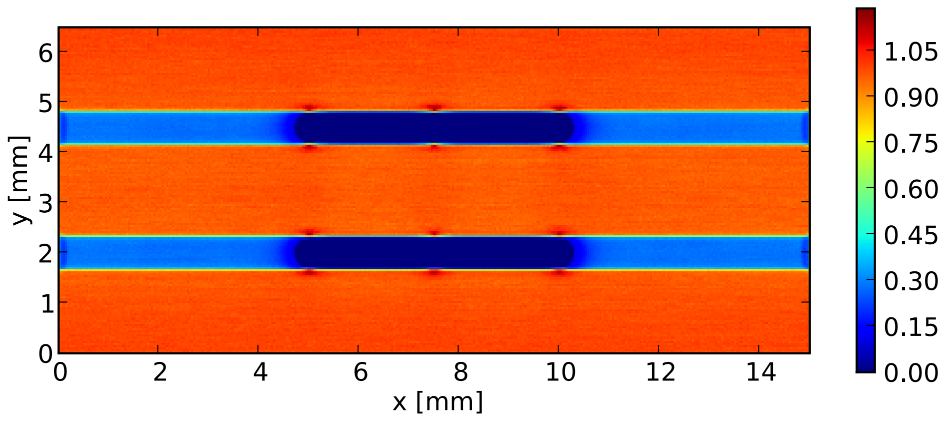


Figure B.33 Probe array, $n = 10^{17}$ m^{-3}, electron density.

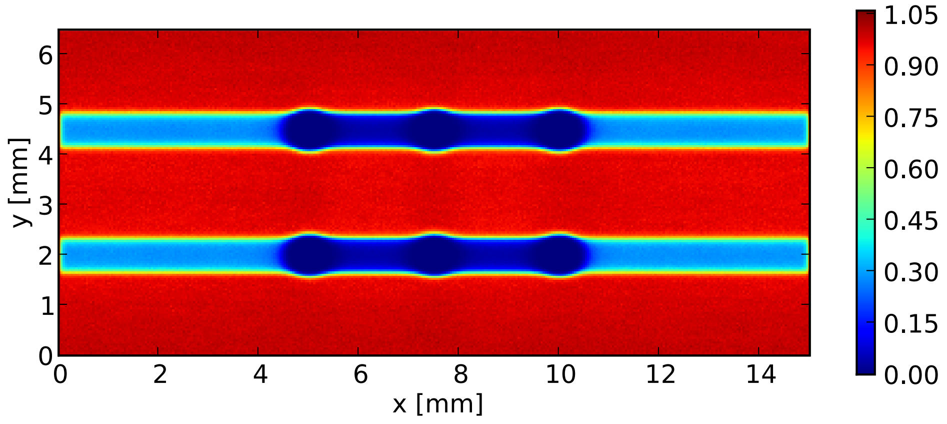


Figure B.34 Probe array, $n = 10^{17}$ m^{-3}, ion density.

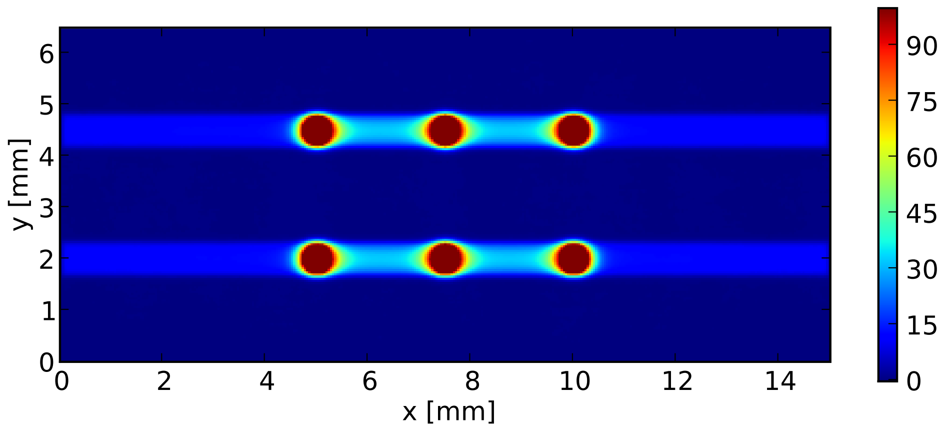


Figure B.35 Probe array, $n = 10^{17}$ m^{-3}, electrostatic potential.

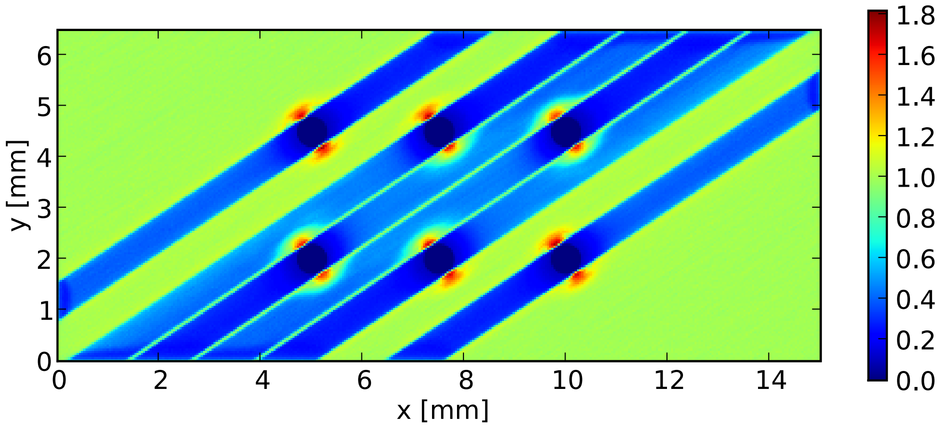


Figure B.36 Probe array, skew magnetic field, electron density.

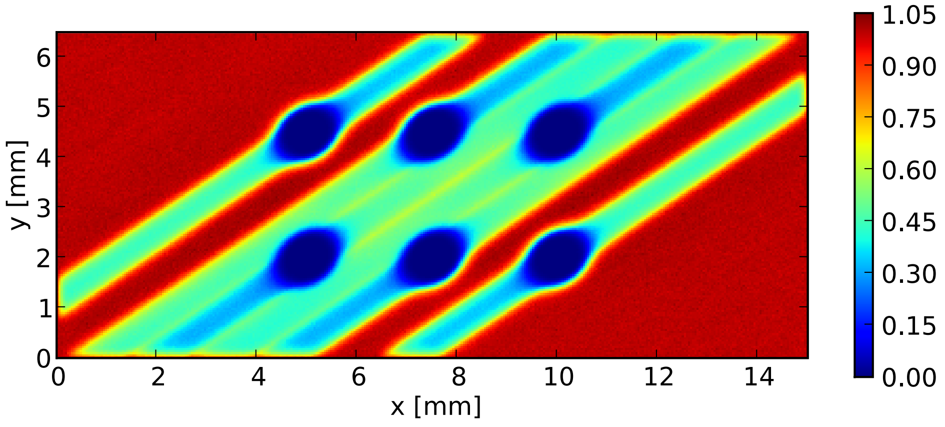


Figure B.37 Probe array, skew magnetic field, ion density.

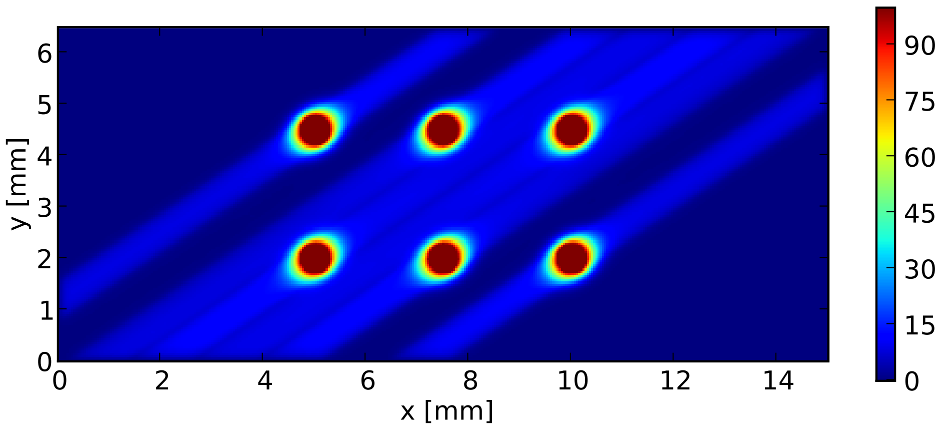


Figure B.38 Probe array, skew magnetic field, electrostatic potential.

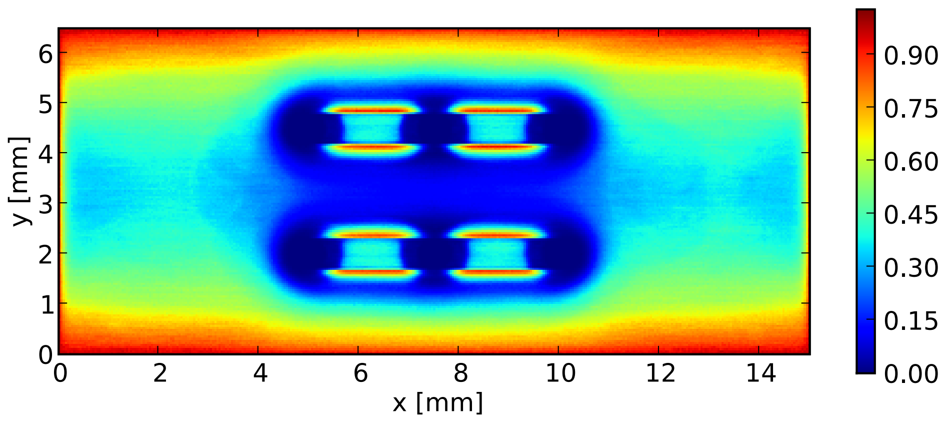


Figure B.39 Probe array, domain too small, $n = 10^{16}$ m^{-3}, electron density.

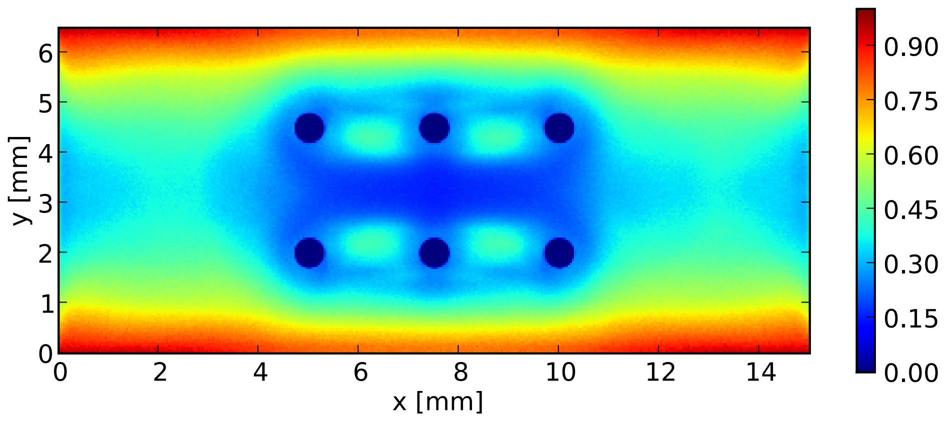


Figure B.40 Probe array, domain too small, $n = 10^{16}$ m^{-3}, ion density.

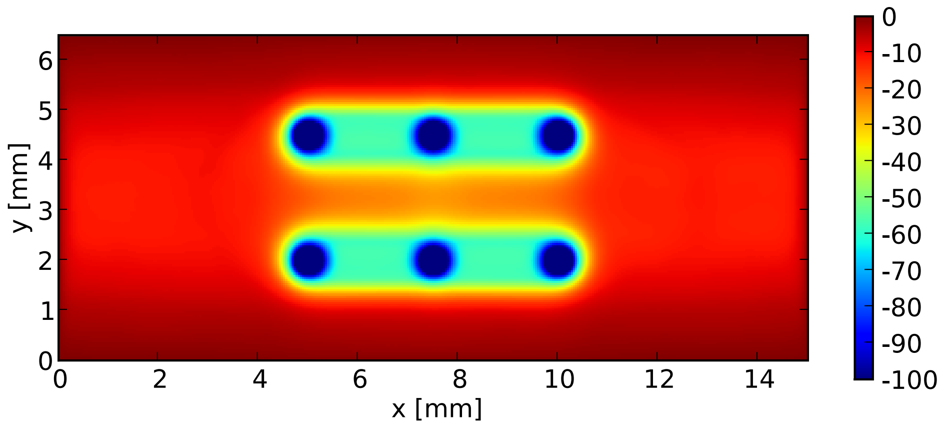


Figure B.41 Probe array, domain too small, $n = 10^{16}$ m^{-3}, electrostatic potential.

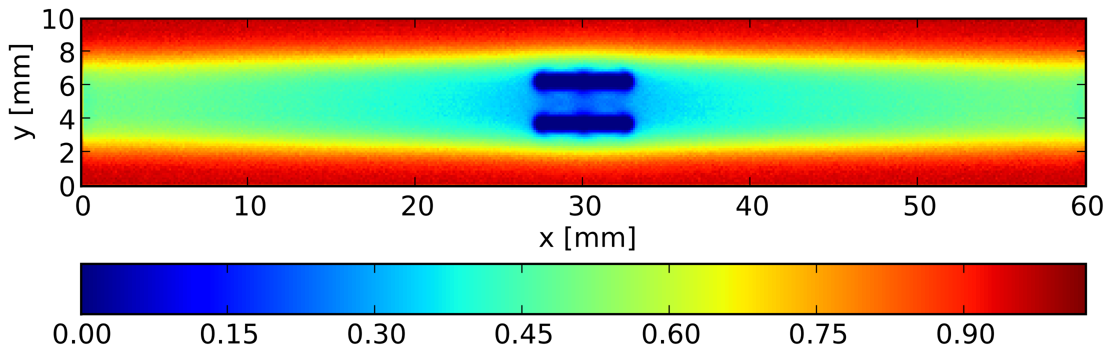


Figure B.42 Probe array, negative potential, $n = 10^{17}$ m^{-3}, electron density.

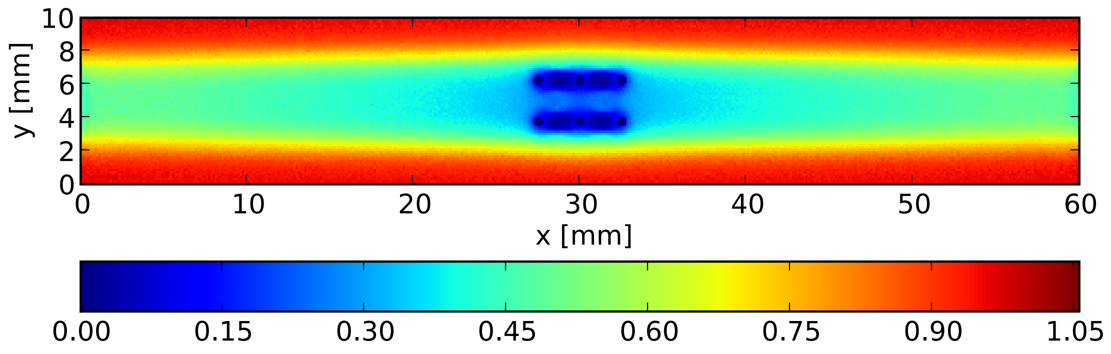


Figure B.43 Probe array, negative potential, $n = 10^{17}$ m^{-3}, ion density.

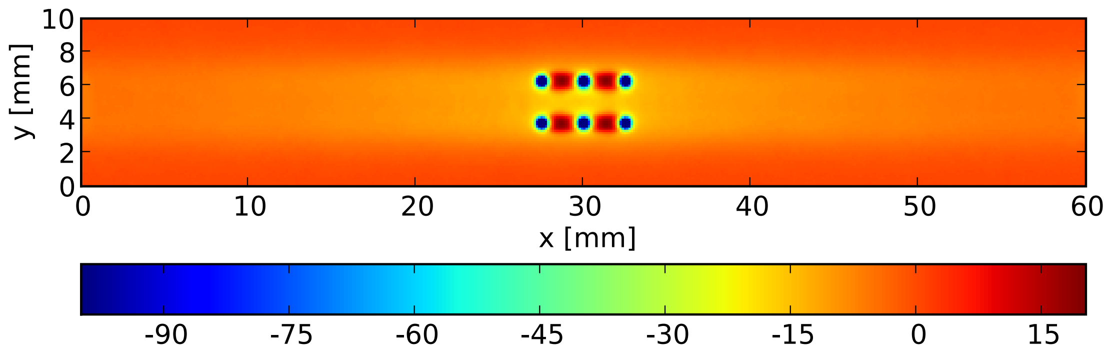


Figure B.44 Probe array, negative potential, $n = 10^{17}$ m^{-3}, electrostatic potential.

B.3 Boundary integral/treecode 2D model

B.3.1 Poisson solver

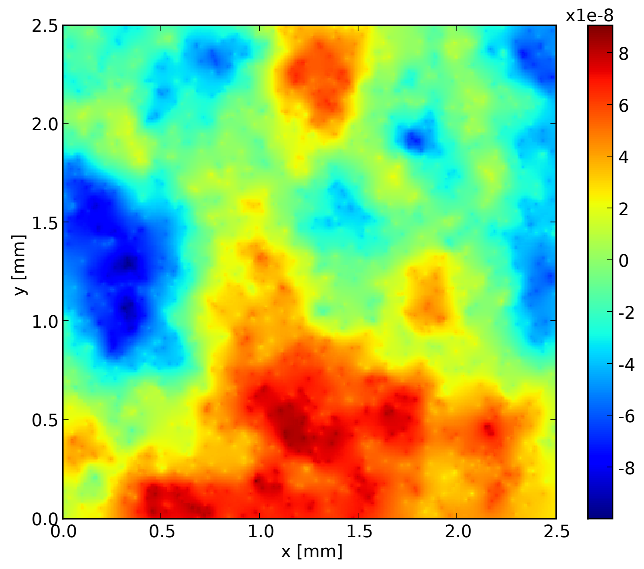


Figure B.45 The particular solution for a BIT solver test case.

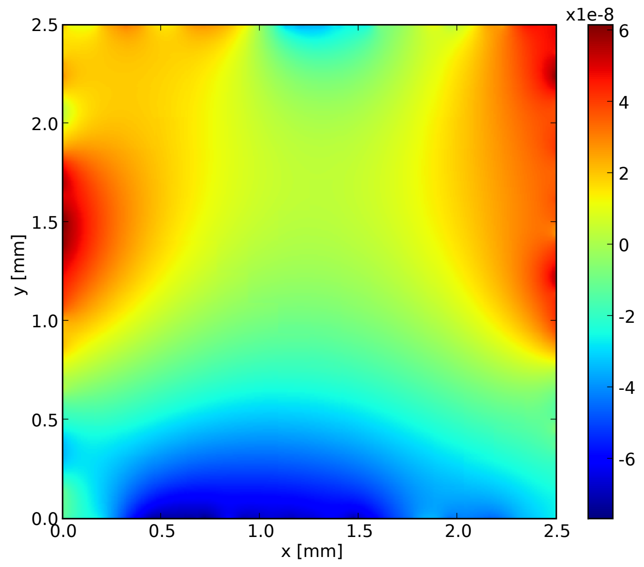


Figure B.46 The homogeneous solution for a BIT solver test case.

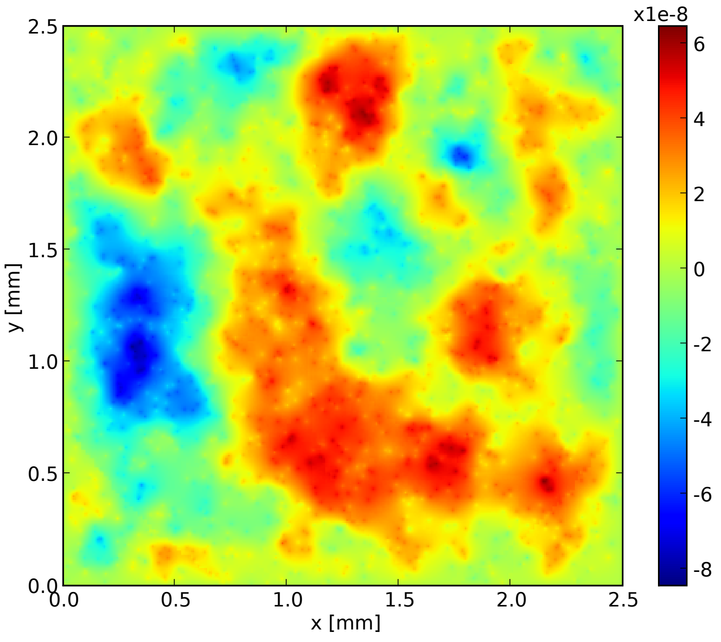


Figure B.47 The full solution for a BIT solver test case.

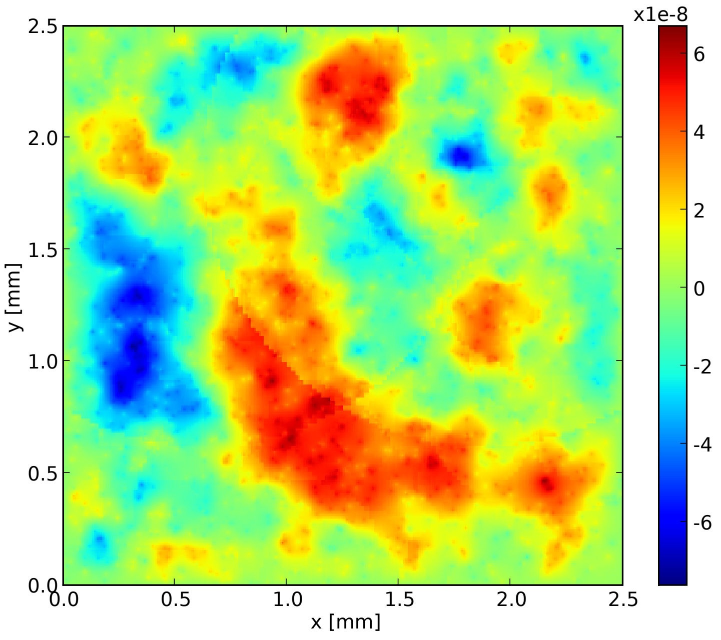


Figure B.48 BIT solver test case — $\theta = 1$.

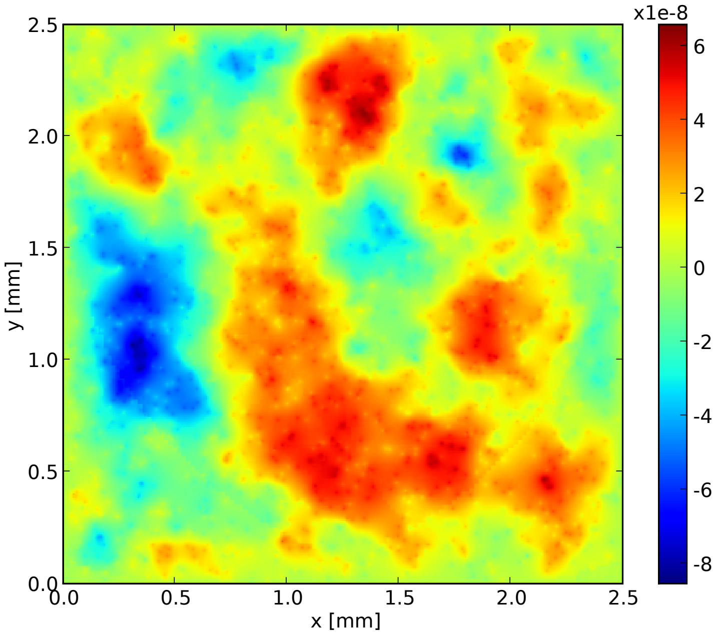


Figure B.49 BIT test case — $\theta = 1.5$.

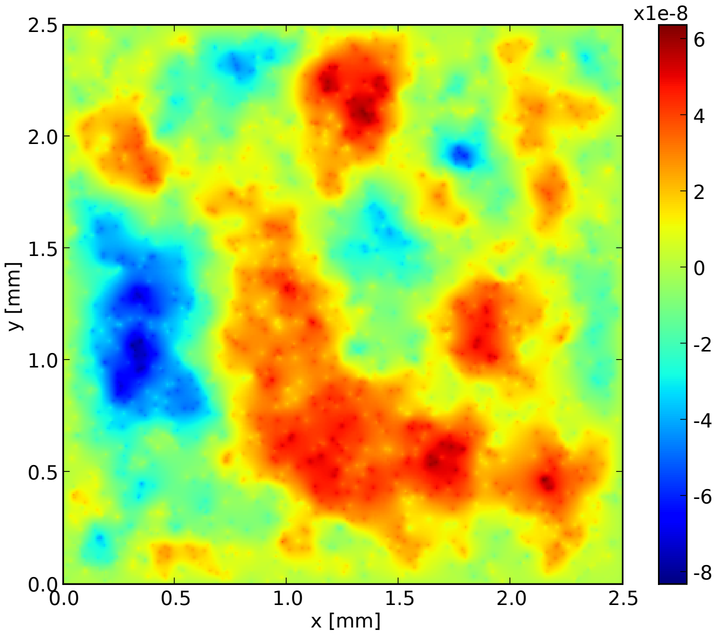


Figure B.50 BIT solver test case — $\theta = 2$.

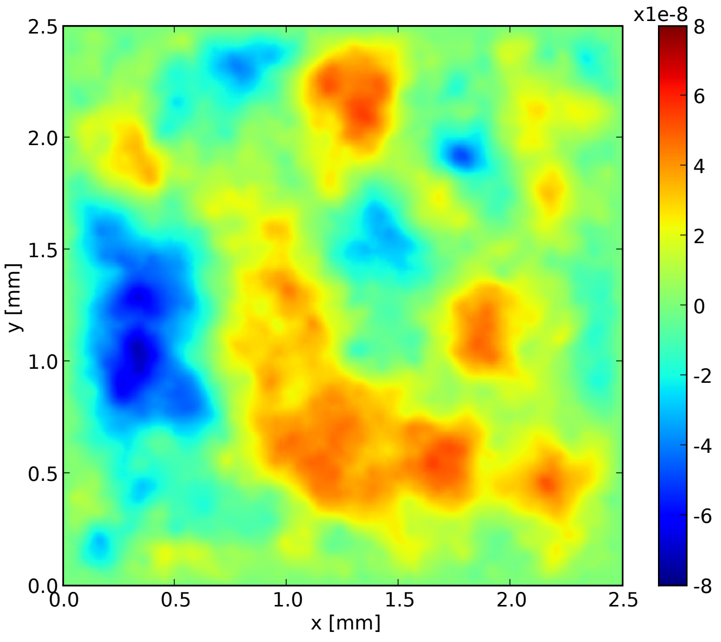


Figure B.51 BIT solver test case — $\delta = 2 \times 10^{-5}$ m.

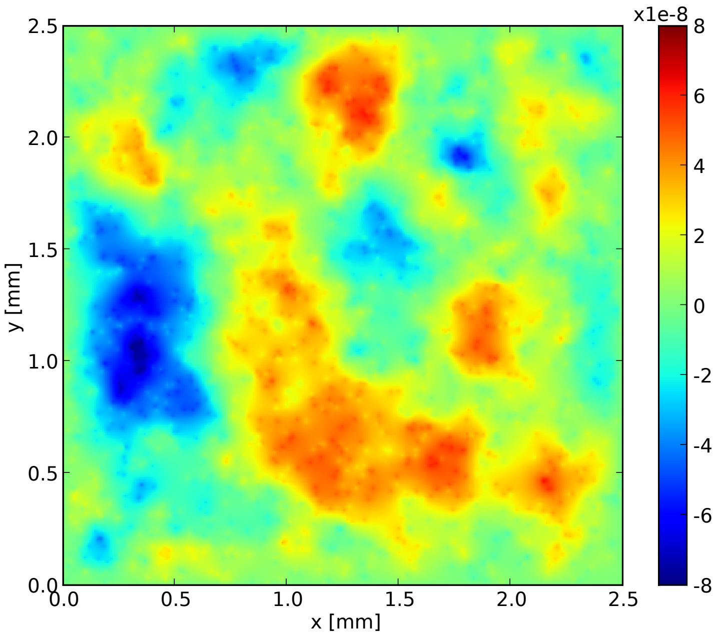


Figure B.52 BIT solver test case — $\delta = 10^{-6}$ m.

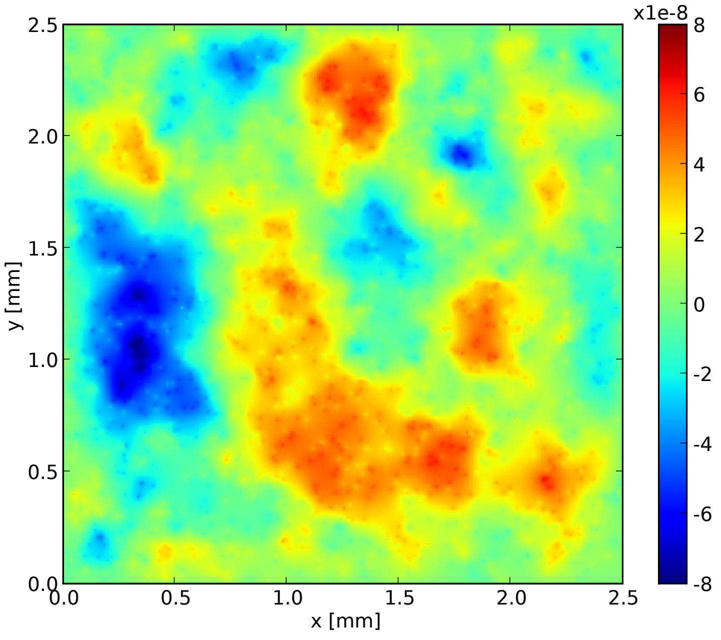


Figure B.53 BIT solver test case — $\delta = 10^{-12}$ m.

B.3.2 Bulk properties

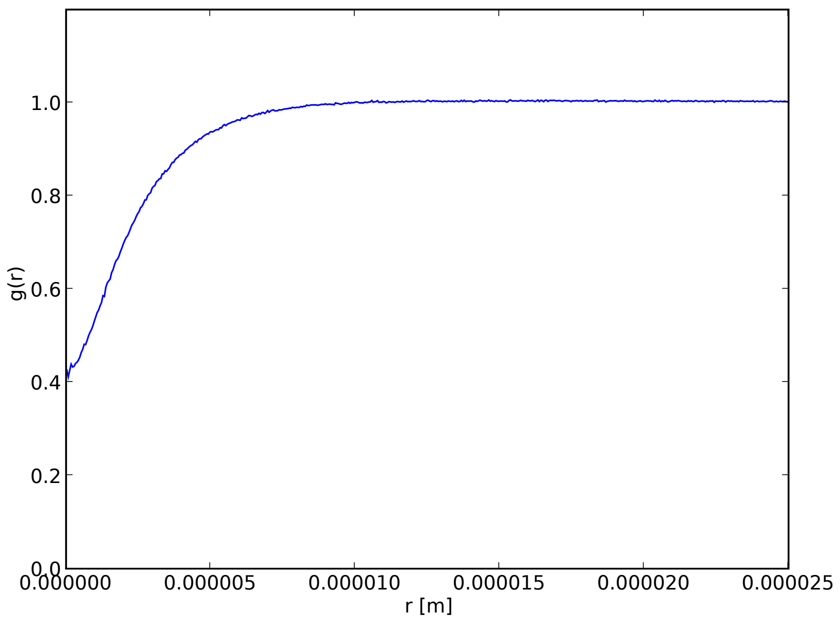


Figure B.54 Electron-electron pair correlation function for $n = 10^{15}$ m^{-3}.

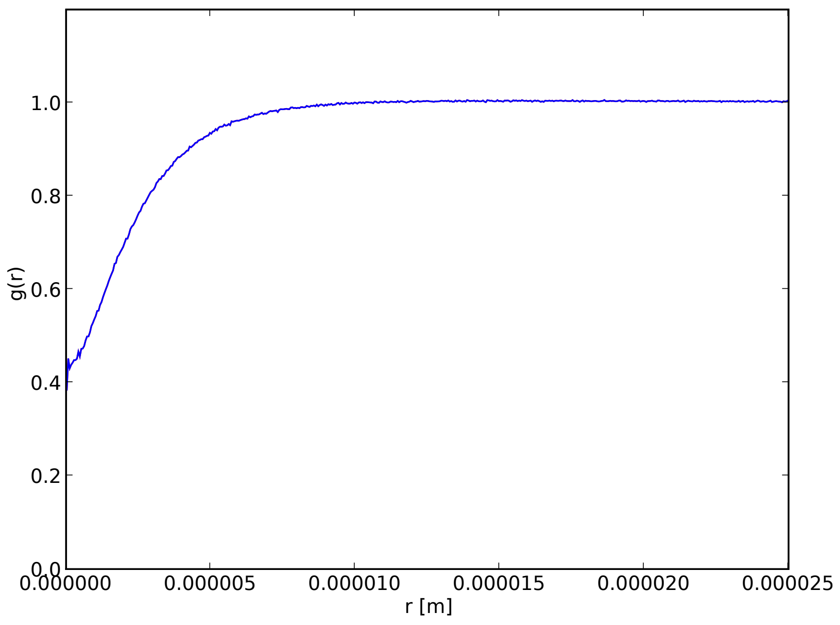


Figure B.55 Ion-ion pair correlation function for $n = 10^{15}$ m^{-3}.

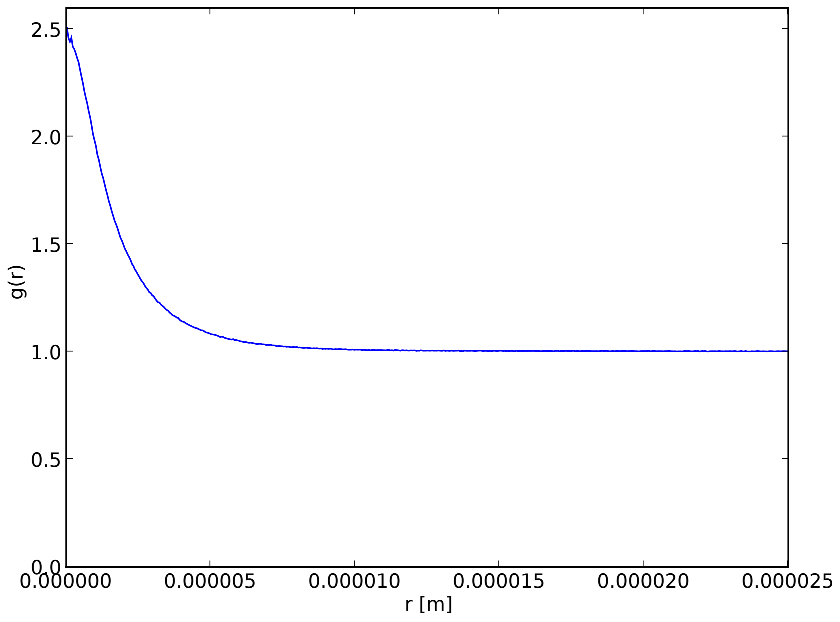


Figure B.56 Electron-ion pair correlation function for $n = 10^{15}$ m^{-3}.

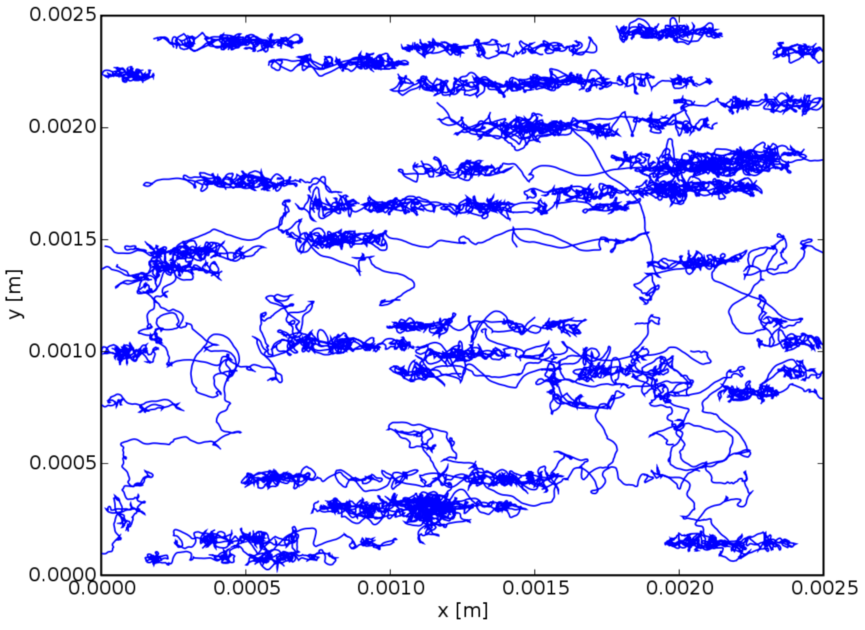


Figure B.57 Randomly chosen electron trajectories for $n = 10^{15}$ m^{-3}.

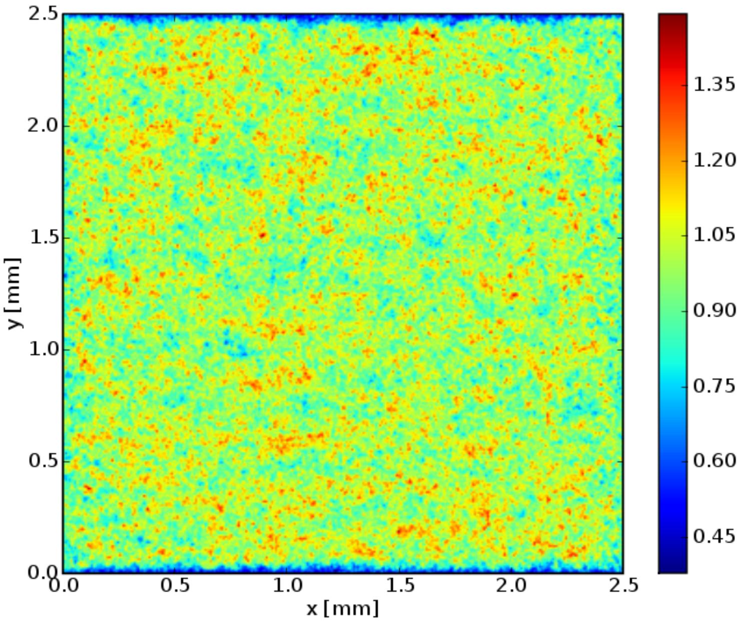


Figure B.58 Electron density for boundary integral/treecode with magnetic field.

APPENDIX

C

Numerical methods

In this appendix we will review some of the standard numerical methods needed in this thesis. First, we will look at numerical differentiation and then we will use it to solve the Poisson's equation on a regular grid.

C.1 Numerical differentiation

We start with the standard Taylor series of a function $f(x)$ around the point a

$$f(x) = \sum_{n=0}^{\infty} \left. \frac{\mathrm{d}^n f(y)}{\mathrm{d}y^n} \right|_{y=a} \frac{(x-a)^n}{n!}. \tag{C.1}$$

We choose $a = x$, expand $f(x+h)$ and truncate the series while adding an error term to obtain

$$f(x) = \sum_{n=0}^{N} \left. \frac{\mathrm{d}^n f(y)}{\mathrm{d}y^n} \right|_{y=a} \frac{(x-a)^n}{n!} + \mathcal{O}(h^{N+1}), \tag{C.2}$$

where $\mathcal{O}(\varepsilon)$ is the usual big O notation for a quantity of the order of ε.

Further, using simply f' to denote the derivative of f we write several terms for the function values at $x \pm h$ explicitly as

$$f(x \pm h) = f(x) \pm h f'(x) + \frac{h^2}{2} f''(x) \pm \frac{h^3}{6} f^{(3)}(x) + \mathcal{O}(h^4). \tag{C.3}$$

We can use this expression and similar ones to write numerical derivatives of data on a regularly spaced grid with node distance h. The general idea is to solve equation (C.3) for the desired derivative at point x. Using several equations, we can systematically eliminate higher terms to obtain a higher degree of accuracy.

The most simple example is just expressing $f'(x)$ from (C.3) to get

$$f'(x) = \frac{f(x+h) - f(x)}{h} + \mathcal{O}(h^2). \tag{C.4}$$

If we have also $f(x-h)$ available, we see that

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} + \mathcal{O}(h^3). \tag{C.5}$$

With the use of the expansions of $f(x \pm 2h)$ we can get a higher accuracy formula for the first derivative in the form

$$f'(x) = \frac{f(x-2h) - 8f(x-h) + 8f(x+h) - f(x+2h)}{12h} + \mathcal{O}(h^5), \tag{C.6}$$

provided that the values of $f(x \pm 2h)$ are available.

This can be potentially extended to any derivative at any desired accuracy, provided that the necessary values are available. For example, the second derivative, using the usual three-point stencil, would be

$$f''(x) = \frac{f(x-h) - 2f(x) + f(x+h)}{h^2} + \mathcal{O}(h^4). \tag{C.7}$$

One sided derivatives can generally be obtained in a way similar to (C.4).

C.2 Poisson's equation on a grid

In this section we will formulate the Poisson's equation on a regular rectangular grid and see that its solution reduces to a sparse linear system. This system can then be treated with suitable numerical methods to obtain the solution of the original equation. Such an approach is called the finite difference method, the details of which can be found for example in [PTVF07].

C.2.1 Basic formulation

The Laplace operator in Cartesian coordinates in dimension d is

$$\Delta = \sum_{i=1}^{d} \frac{\partial^2}{\partial x_i^2}. \tag{C.8}$$

The second partial derivatives can be represented on a grid using the procedure mentioned in the previous section. We will use the expression (C.7), but generally others can be used as well.

We will go through all the details in one dimension and then show how it can be generalized to two and three dimensions.

Let us have a grid of evenly spaced points separated by distance h

$$\{x_k = hk : k = 0 \dots M\} \tag{C.9}$$

and an unknown function $u(x_k) \equiv u_k$ at these points. The Poisson's equation for u_k on the grid, using the above-mentioned differentiation scheme, with the right-hand side b_k is represented by a system of linear equations

$$\frac{u_{k-1} - 2u_k + u_{k+1}}{h^2} = b_k. \tag{C.10}$$

C.2.2 Boundary conditions

The above formula must be modified for $k = 0$ and $k = M$ in a way that depends on the required boundary condition. For periodic boundary conditions, the equations will simply "wrap around" so that

$$\begin{aligned} \frac{u_M - 2u_0 + u_1}{h^2} &= b_0, \\ \frac{u_{M-1} - 2u_M + u_0}{h^2} &= b_M. \end{aligned} \tag{C.11}$$

There remains freedom in the overall shift of all the values of u_k by the same value. The matrix notation of the problem would be

$$\frac{1}{h^2} \begin{bmatrix} -2 & 1 & 0 & \cdots & \cdots & 0 & 1 \\ 1 & -2 & 1 & 0 & \cdots & \cdots & 0 \\ 0 & 1 & -2 & 1 & 0 & \cdots & 0 \\ \vdots & & & \ddots & & & \vdots \\ \vdots & & & & \ddots & & \vdots \\ \vdots & & & & & \ddots & \vdots \\ 1 & 0 & \cdots & \cdots & 0 & 1 & -2 \end{bmatrix} \cdot \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ u_N \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ b_N \end{bmatrix}. \tag{C.12}$$

For Dirichlet boundary conditions requiring the values u_L and u_R at the endpoints we have simply

$$\begin{aligned} u_0 &= u_L, \\ u_M &= u_R. \end{aligned} \tag{C.13}$$

These equations can be explicitly included in the resulting matrix, represented by a 1 on the diagonal and the corresponding value on the right-hand side as in the expression

$$\frac{1}{h^2}\begin{bmatrix} 1 & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ 1 & -2 & 1 & 0 & \cdots & \cdots & 0 \\ 0 & 1 & -2 & 1 & 0 & \cdots & 0 \\ \vdots & & & \ddots & & & \vdots \\ \vdots & & & & \ddots & & \vdots \\ 0 & \cdots & \cdots & 0 & 1 & -2 & 1 \\ 0 & \cdots & \cdots & \cdots & \cdots & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ \vdots \\ \vdots \\ u_{N-1} \\ u_N \end{bmatrix} = \begin{bmatrix} b_L \\ b_1 \\ \vdots \\ \vdots \\ \vdots \\ b_{N-1} \\ b_R \end{bmatrix}. \tag{C.14}$$

Finally, for von Neumann boundary conditions (in 1D)

$$\left.\frac{\mathrm{d}u(x)}{\mathrm{d}x}\right|_{x=0+} = u_{DL}, \qquad \left.\frac{\mathrm{d}u(x)}{\mathrm{d}x}\right|_{x=L-} = u_{DR} \tag{C.15}$$

we get the equations

$$\frac{u_1 - u_0}{h} = u_{DL}, \qquad \frac{u_N - u_{N-1}}{h} = u_{DR} \tag{C.16}$$

and the matrix expression

$$\frac{1}{h^2}\begin{bmatrix} -h & h & 0 & \cdots & \cdots & \cdots & 0 \\ 1 & -2 & 1 & 0 & \cdots & \cdots & 0 \\ 0 & 1 & -2 & 1 & 0 & \cdots & 0 \\ \vdots & & & \ddots & & & \vdots \\ \vdots & & & & \ddots & & \vdots \\ 0 & \cdots & \cdots & 0 & 1 & -2 & 1 \\ 0 & \cdots & \cdots & \cdots & 0 & -h & h \end{bmatrix} \cdot \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ \vdots \\ \vdots \\ u_{N-1} \\ u_N \end{bmatrix} = \begin{bmatrix} b_{DL} \\ b_1 \\ \vdots \\ \vdots \\ \vdots \\ b_{N-1} \\ b_{DR} \end{bmatrix}. \tag{C.17}$$

A Newton boundary condition could be imposed in a similar way. Mixed boundary conditions can be used as well. The above equations represent an almost tridiagonal linear system for the unknown values of u_k.

C.2.3 Generalization to 2D and 3D

To generalize this problem to two or three dimensions, we need to order the grid points so that they form a single vector. This ordering is arbitrary but obviously must be consistent. On a two-dimensional rectangular grid

$$\{[x_k = hk, y_l = kl] : \, k = 0 \ldots M, \, l = 0 \ldots N\} \tag{C.18}$$

we could choose to go along rows of points of constant y and increasing x, stacking such blocks to create a single vector of all the values. Here we keep the grid spacing h the same for all dimension, but generally there can be an independent value for each dimension. An example of the resulting matrix is in displayed Figure C.1.

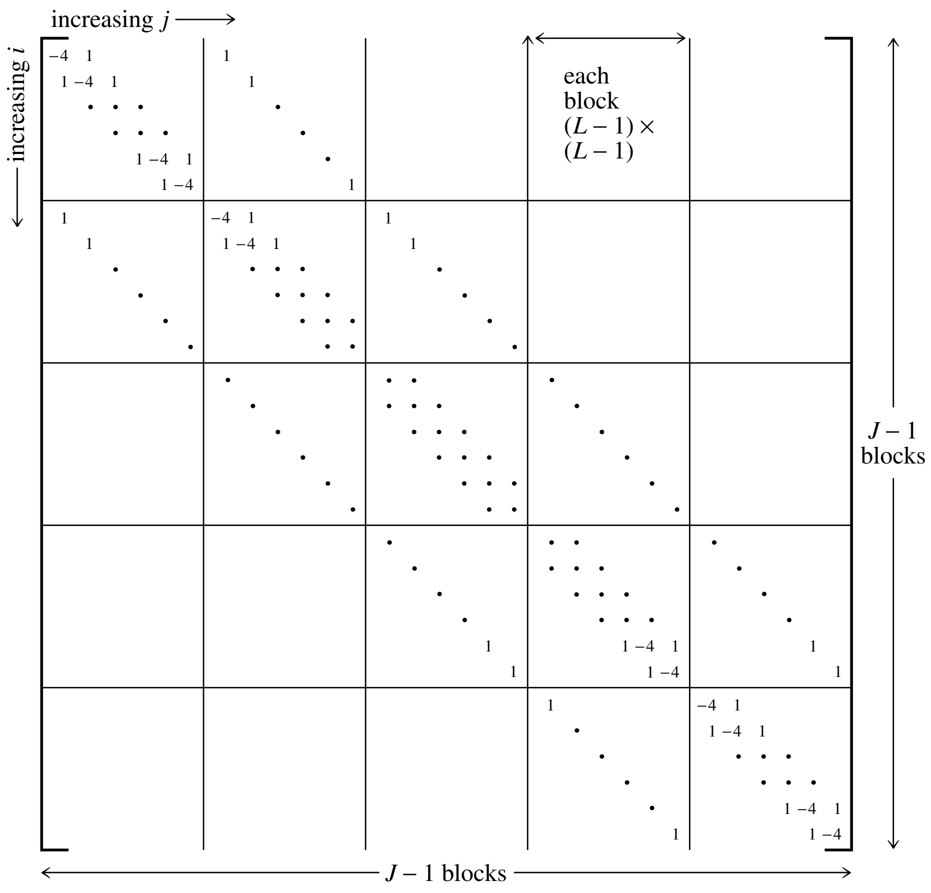


Figure C.1 An example of a Poisson's equation matrix in 2D, reprinted from [PTVF07]. The notation is different from the one in the text. The important thing to note are the additional fringes that correspond to terms connecting grid points across rows in the ordering. The matrix is such that a Dirichlet boundary condition is imposed, if the correct terms are included on the right-hand side.

Generally speaking, we have to solve the system of equations

$$\mathbb{A}\cdot\mathbf{u}=\mathbf{b}', \tag{C.19}$$

where $\mathbf{b}'$ is a modified right-hand side constructed from the original right-hand side and some additional terms related to boundary conditions, as explained above. This system of equations is sparse and suitable methods have to be used, if one wants to treat large problems with decent computational efficiency.

C.3 Systems of linear algebraic equations

Solving a system of linear algebraic equations is perhaps the most common problem in numerical mathematics. It is a very large area of interest and we will only touch it by reviewing the material needed for the present work. The interested reader is referred to [PTVF07] for a more detailed treatment of the subject.

We are interested in the solution of the system

$$\mathbb{A} \cdot \mathbf{x} = \mathbf{b}, \tag{C.20}$$

where $\mathbb{A}$ is a square matrix. By "solution" we mean obtaining the unknown values of $\mathbf{x}$ from the known values of the right-hand side $\mathbf{b}$ for a known matrix $\mathbb{A}$. We will also keep in mind that the problem, to which we would like to apply the solution in this thesis, usually involves solving the same problem many times for the same matrix but different right-hand side vectors. Any problems related to the possible singularity of the matrix will not be discussed here.

There are various categories of methods to solve (C.20). We will deal only with iterative and direct methods here. Among others, there is the option of using the Fast Fourier Transform, but its applicability to the problems in this thesis is limited, so we will not discuss it here. The limitations are related mainly to possible geometries and the practical advantages of this method in comparison to others are questionable [PLH07].

C.3.1 Iterative methods

Iterative methods approach to the solution from an initial guess in a sequence of vectors that are closer and closer to the true solution. The iteration is stopped at a desired accuracy, usually determined by the change of two or several consecutive solutions.

Jacobi method

The basic iterative method is the Jacobi method, which uses the matrix written as a sum

$$\mathbb{A} = \mathbb{L} + \mathbb{D} + \mathbb{U} \tag{C.21}$$

of the lower triangle, the upper triangle and the diagonal. The iteration (iterations indexed by r) then proceeds according to the formula

$$\mathbf{x}^{(r+1)} = \mathbb{D}^{-1} \left[\mathbf{b} - (\mathbb{L} + \mathbb{U}) \cdot \mathbf{x}^{(r)} \right]. \tag{C.22}$$

Gauss-Seidel method

The Gauss-Seidel method uses the same iteration procedure as the Jacobi method, but uses values as they become available, working on the same data all the time, instead of creating a copy. This increases the convergence speed. Because of the modification, the iteration formula becomes

$$\mathbf{x}^{(r+1)} = (\mathbb{L} + \mathbb{D})^{-1} \left(\mathbf{b} - \mathbb{U} \cdot \mathbf{x}^{(r)}\right). \tag{C.23}$$

Both the Jacobi and Gauss-Seidel methods are quite slow converging and therefore stronger methods are usually used for practical problems. There are some differences in the convergence of these methods and it is not possible to say that one of them converges in a more general case. For matrices arising from finite differencing there should not be any convergence problems.

Successive overrelaxation

The Gauss-Seidel method can be further improved by combining the old and new values so that

$$\mathbf{x}^{(r+1)} = (1-\omega)\mathbf{x}^{(r)} + \omega\mathbf{x}^{GS}, \tag{C.24}$$

where $\mathbf{x}^{GS}$ is obtained from $\mathbf{x}^{(r)}$ using a single Gauss-Seidel step. The value of the parameter ω influences the speed of convergence significantly. We will not go into the details of determining the correct value of ω for a specific problem, as this method is not used in the present work.

C.3.2 Direct methods

Direct methods calculate the solution by a procedure that does not involve any intermediate solutions and the results is exact (that is, if we do not consider truncation errors caused by finite floating point precision of computers). Some methods include steps to prepare or transform the original matrix in such a way, that the calculation of a solution for a given vector $\mathbf{b}$ is then faster. This is very useful for problems involving a constant matrix and a lot of right-hand sides, as the preparation is done only once.

Gauss-Jordan elimination

The basic idea of the Gauss-Jordan elimination method is to add and subtract suitable multiples of rows to other rows so that the coefficients at certain elements are eliminated and in the end there is only one non-zero value in each row. This is done in a systematic way.

This approach can be well applied to Poisson's equation on a grid in one dimension, as the matrix is completely or almost tridiagonal. The elimination is then straightforward. This particular case is sometimes called the Thomas algorithm.

LU decomposition

One of the methods that perform some preparation steps on the matrix $\mathbb{A}$ is the LU decomposition.

The first step is to find a lower triangular matrix $\mathbb{L}$ with zeros on the diagonal and an upper triangular matrix $\mathbb{U}$ such that

$$\mathbb{A} = \mathbb{L} \cdot \mathbb{U}. \tag{C.25}$$

The solution can then be found easily in two steps. We find $\mathbf{y}$ such that

$$\mathbb{L} \cdot \mathbf{y} = \mathbf{b} \tag{C.26}$$

and use it to find $\mathbf{x}$ from

$$\mathbb{U} \cdot \mathbf{x} = \mathbf{y}. \tag{C.27}$$

At first sight, we have to solve two systems instead of one. The substantial advantage of these is, however, that they are both simple. They are solved by forward substitution and backsubstitution. Each of the systems contains one equation that already represents the value of an unknown. This can be used to solve the next equation and so on.

There are various ways to obtain the decomposition and the specific choice depends on the problem at hand. In practice a library is usually used, as there are high-quality implementations available.

Iterative improvement

Because of the finite precision of the computer representation of floating-point numbers, roundoff error accumulation can degrade the accuracy of the result of a direct method of solution. This problem gets worse for matrices that are close to singular. We can correct it by the application of the iterative improvement of the solution.

Let us take $\mathbf{x}$ as the exact solution of the original system

$$\mathbb{A} \cdot \mathbf{x} = \mathbf{b}. \tag{C.28}$$

After the use of a direct method we have a solution that is slightly wrong and corresponds to a different right-hand side:

$$\mathbb{A} \cdot (\mathbf{x} + \delta\mathbf{x}) = \mathbf{b} + \delta\mathbf{b}. \tag{C.29}$$

By simple manipulation of the above equations, we obtain an equation for the difference

$\delta\mathbf{x}$

$$\mathbb{A} \cdot \delta\mathbf{x} = \mathbb{A} \cdot (\mathbf{x} + \delta\mathbf{x}) - \mathbf{b}, \tag{C.30}$$

where the right-hand side is known.

This process can be applied several times, if necessary.

C.3.3 Sparse systems

Certain problems generate matrices that have a substantial proportion of their elements equal to zero. In these cases, using a straightforward representation by storing all the values in the memory of a computer would be wasteful and often also quite impossible for reasonably sized problems.

For these matrices, sparse storage formats are used. These store only some information about the positions of non-zero elements and their values. This means that any algorithms working on such matrices must be modified to be able to work with them. A discussion of any details in this area is, however, beyond the scope of this work.

There are two advantages of using sparse matrices — the possibility of working with larger matrices and more efficient computation. Usually, algorithms implemented in specialized numerical libraries are used in practice. An example of such implementation is the UMFPACK library [Dav04a].

APPENDIX

D

Rotation of a vector in 3D

In this appendix we will derive an expression for rotating a vector in three dimensions by a given angle around an axis given by a unit vector.

Let us denote the vector we want to transfor $\mathbf{x}$, the unit vector representing the axis direction $\mathbf{a}$, the angle of rotation α and the transformed vector $\mathbf{x}'$.

We will now express $\mathbf{x}'$ using $\mathbf{x}$, $\mathbf{a}$ and α. If we construct orthogonal vectors

$$\mathbf{b}_2 = \mathbf{a} \times \mathbf{x}, \tag{D.1}$$

$$\mathbf{b}_1 = \mathbf{b}_2 \times \mathbf{a}, \tag{D.2}$$

we see that $\mathbf{b}_1$ is a projection of $\mathbf{x}$ into the plane perpendicular to $\mathbf{a}$ and going through the origin. Therefore, the expression

$$\cos(\alpha)\mathbf{b}_1 + \sin(\alpha)\mathbf{b}_2 \tag{D.3}$$

represents the projection of $\mathbf{x}'$ into the same plane.

To get $\mathbf{x}'$, we have to shift this plane so that it passes through the endpoint of $\mathbf{x}$. The correct offset $\mathbf{o}$ can be obtained by subtracting the vector $\mathbf{b}_1$ from $\mathbf{x}$, because it is its projection to the mentioned plane:

$$\mathbf{o} = \mathbf{x} - \mathbf{b}_1. \tag{D.4}$$

Putting all this together, we can write $\mathbf{x}'$ as

$$\begin{aligned} \mathbf{x}' &= \mathbf{x} - \mathbf{b}_1 + \cos(\alpha)\mathbf{b}_1 + \sin(\alpha)\mathbf{b}_2 \\ &= \mathbf{x} + (\cos(\alpha) - 1)\mathbf{b}_1 + \sin(\alpha)\mathbf{b}_2 \\ &= \mathbf{x} + (1 - \cos(\alpha))\mathbf{a} \times (\mathbf{a} \times \mathbf{x}) + \sin(\alpha)\mathbf{a} \times \mathbf{x}. \end{aligned} \tag{D.5}$$

We would like to express $\mathbf{x}'$ as

$$\mathbf{x}' = \mathbb{R}(\mathbf{a}, \alpha) \cdot \mathbf{x}, \tag{D.6}$$

where $\mathbb{R}(\mathbf{a}, \alpha)$ is a matrix representing the rotation with the given parameters. For this, we will use the tilde matrix. Let the symbol $\tilde{a}$ denote a matrix such that

$$\tilde{a}_{ij} = -\varepsilon_{ijk} a_k, \tag{D.7}$$

which gives the matrix elements

$$\tilde{a} = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix}. \tag{D.8}$$

Using this matrix, the cross product can be written as matrix multiplication:

$$\mathbf{a} \times \mathbf{b} = \tilde{a} \cdot \mathbf{b}. \tag{D.9}$$

With the tilde matrix, we can see that we can write $\mathbb{R}$ as

$$\mathbb{R} = \mathbb{1} + (1 - \cos(\alpha))\tilde{a}^2 + \sin(\alpha)\tilde{a}, \tag{D.10}$$

which is the expression we wanted to derive.

BIBLIOGRAPHY

[And80] Hans C. Andersen, *Molecular dynamics simulations at constant pressure and/or temperature*, The Journal of Chemical Physics **72** (1980), no. 4, 2384–2393. 2.2.6

[App85] Andrew W. Appel, *An efficient program for many-body simulation*, SIAM Journal on Scientific and Statistical Computing **6** (1985), no. 1, 85–103. 2.3.8

[AT89] M. P. Allen and D. J. Tildesley, *Computer simulation of liquids*, Oxford University Press, USA, June 1989. 2.2, 2.2.2

[Bee76] D. Beeman, *Some multistep methods for use in molecular dynamics calculations*, Journal of Computational Physics **20** (1976), 130+. 2.2.2

[BH86] Josh Barnes and Piet Hut, *A hierarchical O(N log N) force-calculation algorithm*, Nature **324** (1986), no. 6096, 446–449. 2.3.8

[BL91] C. K. Birdsall and Langdon, *Plasma physics via computer simulation (series on plasma physics)*, Taylor & Francis, January 1991. 2.2.2, 2.3.3, 2.3.4, 2.3.4

[Bro74] Jaromír Brož, *Základy fysikálních měření (II)B (in Czech)*. 2.1.5

[Che74] Francis F. Chen, *Introduction to plasma physics*, Plenum Press, New York, 1974. 2.1

[CKV] A. J. Christlieb, R. Krasny, and J. P. Verboncoeur, *A treecode algorithm for simulating electron dynamics in a Penning-Malmberg trap*, Computer Physics Communications **164**, no. 1-3, 306–310. 2.3.9

[CKV04] ———, *Efficient particle simulation of a virtual cathode using a grid-free treecode poisson solver*, Plasma Science, IEEE Transactions on **32** (2004), no. 2, 384–389. 2.3.9

[CKV$^+$06] A. J. Christlieb, R. Krasny, J. P. Verboncoeur, J. W. Emhoff, and I. D. Boyd, *Grid-free plasma simulation techniques*, Plasma Science, IEEE Transactions on **34** (2006), no. 2, 149–165. 2.3.5, 2.3.9, 4.1.2

[Dav04a] Timothy A. Davis, *Algorithm 832: Umfpack v4.3—an unsymmetric-pattern multifrontal method*, ACM Trans. Math. Softw. **30** (2004), no. 2, 196–199. 4.2.6, C.3.3

[Dav04b] ______, *A column pre-ordering strategy for the unsymmetric-pattern multifrontal method*, ACM Trans. Math. Softw. **30** (2004), no. 2, 165–195. 4.2.6

[DD97] Timothy A. Davis and Iain S. Duff, *An unsymmetric-pattern multifrontal method for sparse lu factorization*, SIAM Journal on Matrix Analysis and Applications **18** (1997), no. 1, 140–158. 4.2.6

[DD99] ______, *A combined unifrontal/multifrontal method for unsymmetric sparse matrices*, ACM Trans. Math. Softw. **25** (1999), no. 1, 1–20. 4.2.6

[DiB94] DiBenedetto, Emmanuele, *Partial differential equations*, Birkhäuser Boston, December 1994. 4.1.2

[EHL80] J. W. Eastwood, R. W. Hockney, and D. N. Lawrence, *P3M3DP-The three-dimensional periodic particle-particle/ particle-mesh program*, Computer Physics Communications **19** (1980), 215–261. 2.3.5

[Ewa21] P. P. Ewald, *Die berechnung optischer und elektrostatischer gitterpotentiale*, Annalen der Physik **369** (1921), no. 3, 253–287. 2.3.6

[FLS63] Richard Feynman, Robert Leighton, and Matthew Sands, *The Feynman Lectures on Physics*, second ed., vol. 1, Addison-Wesley, Boston, 1963. 2.2.2

[FS01] Daan Frenkel and B. Smit, *Understanding molecular simulation (computational science series, vol 1)*, Academic Press, October 2001. 2.2, 2.2.2, 4.1.1

[Got] *http://www.tacc.utexas.edu/resources/software/#blas*. 4.2.6

[GR87] L. Greengard and V. Rokhlin, *A fast algorithm for particle simulations*, Journal of Computational Physics **73** (1987), no. 2, 325–348. 2.3.8

[GSL] *http://www.gnu.org/software/gsl/*. 4.2.6

[HE88] R. W. Hockney and J. W. Eastwood, *Computer simulation using particles*, Taylor & Francis, January 1988. 2.3.3, 2.3.4

[HKVL08] B. Hess, C. Kutzner, D. Vanderspoel, and E. Lindahl, *Gromacs 4: Algorithms for highly efficient, load-balanced, and scalable molecular simulation*, J. Chem. Theory Comput. (2008). 2.2.4

[Hoc70] R. W. Hockney, *Potential calculation and some applications*, Methods Comput. Phys. (1970). 2.2.2

[Hoo85] William G. Hoover, *Canonical dynamics: Equilibrium phase-space distributions*, Physical Review A **31** (1985), no. 3, 1695+. 2.2.6

[Krl05] L. Krlín, *Plasma Theory, lecture at MFF UK, material available from the author*, 2005. 2.1

[Lip07] P. Lipavský, *Condensed Matter Theory II, lecture at MFF UK, material available from the author (in Czech)*, 2007. 2.1.3

[LS24] I. Langmuir and Mott H. Smith, *Langmuir probe technique*, General Electric Rev **27** (1924). 2.1.5

[mat] *http://matplotlib.sourceforge.net/*. 4.2.6

[NKK03] I. Nezbeda, J. Kolafa, and M. Kotrla, *Introduction to computer simulations: Monte-Carlo and molecular dynamics methods (in Czech)*, Karolinum, Prague, 2003. 2.2.2, 2.2, 2.3, 2.2.3, 2.4, 2.2.5

[Ope] *http://www.open-mpi.org/*. 4.2.6

[Pek] Z. Pekárek, *personal communication*. 4.2.4, 4.2.4, 4.2.6

[PLH07] Z. Pekárek, M. Lahuta, and R. Hrach, *Improving performance of multidimensional particle-in-cell codes for modelling of medium pressure plasma*, J. Phys.: Conf. Ser. **63** (2007), no. 1, 012009+. 4.2.4, C.3

[Pra] D. Pražák, *personal communication*. 4.1.2, 4.1.2

[PTVF07] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery, *Numerical recipes: The art of scientific computing*, Cambridge University Press, August 2007. C.2, C.1, C.3

[SABW82] William C. Swope, Hans C. Andersen, Peter H. Berens, and Kent R. Wilson, *A computer simulation method for the calculation of equilibrium constants for the formation of physical clusters of molecules: Application to small water clusters*, The Journal of Chemical Physics **76** (1982), no. 1, 637–649. 2.2.2

[SCo] *http://www.scons.org/*. A

[Sou] Vladimír Souček, *Mathematics for Physicists III, lecture material available at* *http://www.karlin.mff.cuni.cz/~soucek/* *(in Czech)*. 4.1.2

[SSC93] J. M. Sanz-Serna and M. P. Calvo, *Symplectic Numerical Methods for Hamiltonian Problems*, International Journal of Modern Physics C **4** (1993), 385–392. 4.1.1

[TBM92] M. Tuckerman, B. J. Berne, and G. J. Martyna, *Reversible multiple time scale molecular dynamics*, J. Chem. Phys. **97** (1992), 1990–2001. 2.2.2, 4.1.1, 4.1.1, 4.1.1, 4.1.1, 4.2.7

[UMF] *http://www.cise.ufl.edu/research/sparse/umfpack/*. 4.2.6

[Val] *http://www.valgrind.org/*. 4.2.7

[Ver67] Loup Verlet, *Computer "Experiments" on Classical Fluids. I. Thermodynamical Properties of Lennard-Jones Molecules*, Physical Review **159** (1967), no. 1, 98+. 2.2.2

[Ver05] J. P. Verboncoeur, *Particle simulation of plasmas: review and advances*, Plasma Physics and Controlled Fusion **47** (2005), no. 5A, A231–A260. 2.3.4

[Š06] Jiří Šimek, *Rozvoj metod počítačové fyziky pro fyziku plazmatu a fyziku tenkých vrstev (in Czech)*, Ph.D. thesis, 2006. 4.2.4

[Wei] Eric W. Weisstein, *"Green's Identities." From MathWorld—A Wolfram Web Resource.* *http://mathworld.wolfram.com/GreensIdentities.html*. 4.1.2