Numerical Simulation of Kinetic Effects in Ionospheric Plasma

Bengt Eliasson

March 22, 2001

Abstract

In this thesis, we study numerically the one-dimensional non-relativistic Vlasov equation for a plasma consisting of electrons and infinitely heavy ions. This partial differential equation describes the evolution of the distribution function of particles in the two-dimensional phase space (x, v). The Vlasov equation describes, in statistical mechanics terms, the collective dynamics of particles interacting with long-range forces, but neglects the interactions due to short-range "collisional" forces. A space plasma consists of electrically charged particles, and therefore the most important long-range forces acting on a plasma are the Lorentz forces created by electromagnetic fields.

What makes the numerical solution of the Vlasov equation to a challenging task is firstly that the fully three-dimensional problem leads to a partial differential equation in the six-dimensional phase space, plus time, making it even hard to store a discretized solution in the computer's memory. Secondly, the Vlasov equation has a tendency of structuring in velocity space (due to free streaming terms), in which steep gradients are created and problems of calculating the v (velocity) derivative of the function accurately increase with time.

The method used in this thesis is based on the technique of Fourier transforming the Vlasov equation in velocity space and then solving the resulting equation. We have developed a method where the small-scale information in velocity space is removed through an outgoing wave boundary condition in the Fourier transformed velocity space. The position of the boundary in the Fourier transformed variable determines the amount of small-scale information saved in velocity space.

The above numerical method is used to investigate numerically a phenomenon of tunnelling of information through an ionospheric layer, discovered in experiments, and to assess the accuracy of approximate analytic formulae describing plasma wave dispersion. The numerical results are compared with theoretical predictions, and further physical experiments are proposed.

Contents

CHAPTER 1

Introduction

Space plasma research is a relatively new discipline, going back in time about 80 years. Before the "space age," opened up with the advent of the artificial satelites, space was assumed to be essentially a vacuum, whose content of matter was limited to the high energy particles that constitute the cosmic radiation. The discovery of the Earth's ionosphere came from radio wave observations and the recognition that only a reflecting layer composed of electrons and ions could explain the characteristics of the observations [8].

A planet's ionosphere is a partially ionised gas that envelopes the planet, forming an interface between the planet's atmosphere (if it exists) and space. Since the gas of the ionosphere is ionised, it cannot be fully described by the equations of neutral fluid dynamics. Furthermore, it is also often necessary to use models which take the non-equilibrium distribution of particles in velocity space into account.

One example of an ionosphere is the earth's ionosphere [8], which begins at an altitude below 100 km and extends to an altitude of about 1000 km. The gas of the earth's ionosphere is composed mainly of atoms and molecules formed by the elements oxygen and nitrogen. These atoms and molecules are ionised by the radiation from the sun and by impact of energetic particles. The degree of ionisation is of the order 10^{-2} to 10^{-4}, hence the ionosphere is a partially ionised *plasma* [19].

The studies of the earth's ionosphere were made possible by the development of technical equipment for remote sensing, and by the space programme with the associated development of instruments for balloons, rockets, and satellites, which made *in situ* measurements possible. Most of the early research was aimed at explaining the various layers in the ionosphere and their variability with local time, latitude, season, etc. As time passed by, the emphasis of ionospheric research shifted towards understanding the dynamics and plasma physics of ionospheric

phenomena. Following this line towards basic research, researchers about 20 years ago began to use the ionosphere as a test bed for fundamental nonlinear plasma physics experiments, where the plasma was perturbed by injection of electromagnetic waves into the plasma [22]. By these controlled experiments, dynamic and often chaotic processes could be studied and compared with theory. One type of such experiments is mentioned in Chapter 5.

The development of computers and numerical methods has made it possible to test mathematical models of plasma physics, sometimes in a more controlled manner than in physical experiments which are often disturbed by unwanted interferences from radio transmitters and other electrical equipment. Numerical simulations makes it easier to measure and visualise physical quantities, without having to plan the design and operation of sensing equipment as in a physical experiment. By performing the relatively inexpensive numerical experiments, one can test the mathematical models and then predict more carefully which physical experiments should be undertaken.

A mathematical model which describes the non-equilibrium distribution of particles is the Vlasov (or collisionless Boltzmann) equation. It describes the collective dynamics of particles interacting with long-range forces, but neglects any short-range "collisional" forces. For the electrically charged particles of very low mass in a plasma, the most important long-range force is the *Lorentz force*, created by electromagnetic fields. The charged particles may then act as sources of electric net charges and currents, creating self-consistent electromagnetic fields as described by the Maxwell equations.

In this thesis the one-dimensional non-relativistic Vlasov equation is studied numerically. What makes the numerical solution of the Vlasov equation such a challenging task is, firstly, that the fully three-dimensional problem leads to a partial differential equation in the six-dimensional phase space, plus time, making it even hard to store a discretized solution in the computer's memory. Secondly, The Vlasov equation has a tendency of structuring in velocity space (due to the creation of ballistic, or free streaming, particles), in which steep gradients in the velocity direction are created and problems of accurately calculating the v (velocity) derivative of the function increase with time [1].

The numerical method analysed in this thesis is based on the technique of Fourier transforming the Vlasov equation in velocity space and then solving the resulting equation numerically. The small-scale information in velocity space is removed through an outgoing wave boundary condition in the Fourier transformed velocity space. The position of the boundary in the Fourier transformed variable determines the amount of small-scale information saved in velocity space.

The numerical method devised is used to investigate numerically the phenomenon of an apparent tunnelling of plasma waves through an ionospheric layer, discovered

in experiments [7]. The numerical results are compared with theoretical predictions, and further physical experiments are proposed.

In Chapter 2, the numerical method for the one-dimensional Fourier transformed Vlasov equation is described in detail. A large part of the text in this chapter has been submitted to and accepted for publication by Journal of Scientific Computing. In Chapter 3, the parallel implementation of the algorithms is described. The efficiency of the parallelised code is discussed, together with some numerical tests. In Chapter 4, a comparison is made between a theoretical approximation of dispersion laws for electrostatic electron waves, and the numerically obtained dispersion law. In Chapter 5, a mechanism for tunnelling of plasma waves through a Gaussian plasma battier, mimicking an ionospheric layer, is investigated numerically. The numerical tests are compared with theoretical predictions, and physical experiments are proposed in order to verify or falsify the numerical and theoretical results. The numerical algorithm described in Chapter 2 and 3 is implemented in Fortran 90, and the source code is listed in Appendix *A*.

CHAPTER 2

Outflow Boundary Conditions for the Fourier Transformed One-Dimensional Vlasov-Poisson System

2.1 Introduction

Methods of solving numerically the Vlasov equation have been developed for many decades, including methods based on Hermite and Fourier expansions [1, 4] and methods based on the convective structure of the Vlasov equation [3]. Convective schemes have also been developed for the collisional Boltzmann equation [5].

A problem with the Vlasov equation is its tendency of structuring in velocity space (due to the creation of ballistic, or free streaming particles), in which steep gradients of the distribution function are created in the velocity direction, and problems of calculating the v (velocity) derivative of the function accurately increase with time [1].

Due to the sampling (Nyqvist) theorem, the tendency of structuring of the Vlasov equation makes it impossible to represent, after a finite time, all parts of the solution on a uniform grid. If not treated carefully, this problem may eventually lead to the so-called *recurrence phenomenon* where parts of the initial condition artificially re-appear on the numerical grid [3].

In applications, the recurrence phenomenon may in some cases be unimportant if other processes dominate [12], but can introduce significant errors if, for example, the long-time behaviour of a single wave is studied [13].

One method of minimising detremental effects due to the recurrence phenomenon is to have a dense enough grid, so that the interesting physical results have ample time to develop, and then to terminate the simulation before the recurrence phenomenon takes place [13]. Another method is to apply smoothing operators to the

numerical solution so that the finest structures never appear on the numerical grid [3].

The method analysed in the present chapter is related to the second of the above two methods, but instead of direct damping of small-scale information, the small-scale information in velocity space is removed through an outgoing wave boundary condition in the Fourier transformed velocity space. The position of the boundary in the Fourier transformed variable determines the amount of small-scale information saved in velocity space. The objective of the method is thus not to resolve the solution fully but only to a certain degree, and to remove the finest structures of the solution. How much of the small-scale information one needs to save depends strongly on the physical problem.

In Section 2.2.1 the three-dimensional Vlasov-Maxwell system is discussed, together with the Fourier transform technique in velocity space. In Sections 2.2.2–2.2.5 the one-dimensional Vlasov-Poisson system is discussed. These sections contain a justification or motivation for solving numerically the Fourier transformed Vlasov-Poisson system in (x,η,t) space instead of the original system in (x,v,t) space. Well-posed boundary conditions are derived in preparation for the numerical simulation of the Fourier-transformed system. In Section 2.3 the numerical schemes used to approximate the time-dependent solution of the Vlasov-Poisson system are described. In Section 2.4 numerical experiments are presented and compared with known theory and with simulations with other methods. In Section 5.5.3 some conclusions are drawn regarding the usefulness of the method.

2.2 The Vlasov-Maxwell system

2.2.1 The three-dimensional system

The non-relativistic Vlasov equation

$$\frac{\partial f_\alpha}{\partial t} + \mathbf{v}\cdot\boldsymbol{\nabla}_x f_\alpha + \frac{\mathbf{F}_\alpha}{m_\alpha}\cdot\boldsymbol{\nabla}_v f_\alpha = 0 \tag{2.1}$$

where F_α is the *Lorentz force*

$$\mathbf{F}_\alpha = q_\alpha(\mathbf{E}+\mathbf{v}\times\mathbf{B}) \tag{2.2}$$

describes the evolution of the distribution function of electrically charged particles of type α (e.g., "electrons" or "singly ionised oxygen ions"), each particle having the electric charge q_α and mass m_α. One Vlasov equation is needed for each species of particles.

The particles interact via the electromagnetic field. In the absence of external electric and magnetic fields, the charge and current densities act as sources of self-consistent electromagnetic fields according to the Maxwell equations

$$\boldsymbol{\nabla}\cdot\mathbf{E} = \frac{1}{\varepsilon_0}\sum_\alpha q_\alpha n_\alpha \tag{2.3}$$

$$\boldsymbol{\nabla}\cdot\mathbf{B} = 0 \tag{2.4}$$

$$\boldsymbol{\nabla}\times\mathbf{E} = -\frac{\partial\mathbf{B}}{\partial t} \tag{2.5}$$

$$\boldsymbol{\nabla}\times\mathbf{B} = \mu_0\sum_\alpha q_\alpha n_\alpha \mathbf{v}_\alpha + \varepsilon_0\mu_0\frac{\partial\mathbf{E}}{\partial t} \tag{2.6}$$

where the particle number densities n_α and mean velocities $\mathbf{v}_\alpha$ are obtained as moments of the distribution function, as

$$n_\alpha(\mathbf{x},t) = \int_{-\infty}^{\infty} f_\alpha(\mathbf{x},\mathbf{v},t)\,\mathrm{d}^3 v \tag{2.7}$$

and

$$\mathbf{v}_\alpha(\mathbf{x},t) = \frac{1}{n_\alpha(\mathbf{x},t)}\int_{-\infty}^{\infty}\mathbf{v} f_\alpha(\mathbf{x},\mathbf{v},t)\,\mathrm{d}^3 v \tag{2.8}$$

respectively. The Vlasov equation together with the Maxwell and the constitutive equations (2.7) and (2.8) form a closed, coupled system of non-linear partial differntial equations whose analytical solution is known for only a very few special cases.

By using the Fourier transform pair

$$f_\alpha(\mathbf{x},\mathbf{v},t) = \int_{-\infty}^{\infty}\widehat{f_\alpha}(\mathbf{x},\boldsymbol{\eta},t)e^{-\mathrm{i}\boldsymbol{\eta}\mathbf{v}}\,\mathrm{d}^3\eta \tag{2.9}$$

$$\widehat{f_\alpha}(\mathbf{x},\boldsymbol{\eta},t) = \frac{1}{(2\pi)^3}\int_{-\infty}^{\infty} f_\alpha(\mathbf{x},\mathbf{v},t)e^{\mathrm{i}\boldsymbol{\eta}\mathbf{v}}\,\mathrm{d}^3\eta \tag{2.10}$$

the velocity variable $\mathbf{v}$ is transformed into a new variable $\boldsymbol{\eta}$ and the distribution function $f(\mathbf{x},\mathbf{v},t)$ is changed to a new, complex valued, function $\widehat{f}(\mathbf{x},\boldsymbol{\eta},t)$, which obeys the transformed Vlasov equation

$$\frac{\partial\widehat{f_\alpha}}{\partial t} - \mathrm{i}\boldsymbol{\nabla}_x\cdot\boldsymbol{\nabla}_\eta\widehat{f_\alpha} - \mathrm{i}\frac{q_\alpha}{m_\alpha}\left\{\mathbf{E}\cdot\boldsymbol{\eta}\widehat{f_\alpha} + \boldsymbol{\nabla}_\eta\cdot\left[\left(\mathbf{B}\times\boldsymbol{\eta}\right)\widehat{f_\alpha}\right]\right\} = 0 \tag{2.11}$$

The nabla operators $\boldsymbol{\nabla}_x$ and $\boldsymbol{\nabla}_\eta$ denote differentiation with respect to $\mathbf{x}$ and $\boldsymbol{\eta}$, respectively.

Equation (2.11) must be solved together with the Maxwell equations, where the particle number densities and mean velocities are obtained as

$$n_\alpha(\mathbf{x},t) = (2\pi)^3 \widehat{f}_\alpha(\mathbf{x},\mathbf{0},t) \tag{2.12}$$

and

$$\mathbf{v}_\alpha(\mathbf{x},t) = -\mathrm{i}\frac{(2\pi)^3}{n_\alpha(\mathbf{x},t)}\left[\boldsymbol{\nabla}_\eta \widehat{f}_\alpha(\mathbf{x},\boldsymbol{\eta},t)\right]_{\boldsymbol{\eta}=\mathbf{0}} \tag{2.13}$$

respectively. One can note that the integrals over infinite $\mathbf{v}$ space have been converted to evaluations at a single point in $\boldsymbol{\eta}$ space. The factor $(2\pi)^3$ in Equation (2.10), (2.12) and (2.13) is valid for three velocity dimensions. For n velocity dimensions this factor is $(2\pi)^n$.

2.2.2 The one-dimensional Vlasov-Poisson system

In order to explore advantages and disadvantages of the Fourier transformation technique just described, we have chosen to study numerically a simpler case, the one-dimensional Vlasov-Poisson system consisting of electrons and ions, with the ions assumed fixed uniformly in space. These assumptions lead to the system

$$\begin{aligned} &\frac{\partial f}{\partial t} + v\frac{\partial f}{\partial x} - \frac{eE}{m}\frac{\partial f}{\partial v} = 0 \\ &\frac{\partial E(x,t)}{\partial x} = \frac{e}{\varepsilon_0}\left[n_0 - \int_{-\infty}^{\infty} f(x,v,t)\,\mathrm{d}v\right] \end{aligned} \tag{2.14}$$

where $q_\mathrm{e} = -|e|$ is the charge of the electron and n_0 is the neutralising heavy ion density background.

Introducing the Fourier transform pair

$$f(x,v,t) = \int_{-\infty}^{\infty} \widehat{f}(x,\eta,t)\mathrm{e}^{-\mathrm{i}\eta v}\,\mathrm{d}\eta \tag{2.15}$$

$$\widehat{f}(x,\eta,t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} f(x,v,t)e^{\mathrm{i}\eta v}\mathrm{d}v \tag{2.16}$$

equation (2.14) will be transformed into

$$\begin{aligned} &\frac{\partial \widehat{f}}{\partial t} - \mathrm{i}\frac{\partial^2 \widehat{f}}{\partial x \partial \eta} + \mathrm{i}\frac{eE}{m}\eta\widehat{f} = 0 \\ &\frac{\partial E(x,t)}{\partial x} = \frac{e}{\varepsilon_0}\left[n_0 - 2\pi\widehat{f}(x,0,t)\right] \end{aligned} \tag{2.17}$$

Equation (2.17) has been studied analytically, by means of the same Fourier transform technique as the one we use here, by H. Neunzert [15, 16].

The systems (2.14) and (2.17) can be cast into dimensionless form by a scaling of variables: the time t is scaled to the inverse of the plasma frequency $\omega_p^{-1} = \sqrt{\varepsilon_0 m/(n_0 e^2)}$, the velocity v is scaled to the thermal velocity v_{th}; the new variable η is then scaled to the inverse of the thermal velocity, and the spatial variable x is scaled to the Debye length $r_D = v_{th}\omega_p^{-1}$. Finally, the function $\widehat{f}$ is scaled to the background density n_0, the function f is scaled to n_0/v_{th} and the electric field E is scaled to the quantity $v_{th}\sqrt{n_0 m/\varepsilon_0}$. In terms of primed, dimensionless variables, the scaling is

$$t = \omega_p^{-1} t' \tag{2.18}$$

$$v = v_{th} v' \tag{2.19}$$

$$\eta = v_{th}^{-1} \eta' \tag{2.20}$$

$$\widehat{f} = n_0 \widehat{f'} \tag{2.21}$$

$$f = n_0 v_{th}^{-1} f' \tag{2.22}$$

$$E = v_{th}\sqrt{n_0 m/\varepsilon_0}\, E' \tag{2.23}$$

By this scaling of variables, the systems (2.14) and (2.17) attain the dimensionless form, where the primes have been omitted,

$$\frac{\partial f}{\partial t} + v\frac{\partial f}{\partial x} - E\frac{\partial f}{\partial v} = 0 \tag{2.24}$$

$$\frac{\partial E(x,t)}{\partial x} = 1 - \int_{-\infty}^{\infty} f(x,v,t)\,\mathrm{d}v \tag{2.25}$$

and

$$\frac{\partial \widehat{f}}{\partial t} - \mathrm{i}\frac{\partial^2 \widehat{f}}{\partial x \partial \eta} + \mathrm{i}\eta E \widehat{f} = 0 \tag{2.26}$$

$$\frac{\partial E(x,t)}{\partial x} = 1 - 2\pi \widehat{f}(x,0,t) \tag{2.27}$$

respectively.

2.2.3 The problem of structuring in velocity space

Due to the property of conservation of phase memory, the Vlasov-Poisson system in phase (x,v,t) space may develop fine structures in velocity space, since no smearing of the solution occurs. This can be illustrated by a simple example:

By making the somewhat unphysical assumption that the self-consistent electric field is so weak that the Vlasov equation degenerates into the interaction-free model equation

$$\frac{\partial f}{\partial t} + v\frac{\partial f}{\partial x} = 0 \tag{2.28}$$

with the choice of initial condition

$$f(x,v,0) = f_0(x,v) = [1 + A\cos(k_x x)]\, e^{-v^2/2} \tag{2.29}$$

the solution to this initial value problem becomes

$$f(x,v,t) = f_0(x - vt, v) = [1 + A\cos(k_x x - k_x v t)]\, e^{-v^2/2} \tag{2.30}$$

This solution becomes more oscillatory in the velocity direction with increasing time; it will in fact be impossible to represent the solution after a finite time due to the Nyqvist theorem, which states that one needs at least two grid points per wavelength in order to represent a solution on an equidistant grid.

For this simple example it is possible to calculate the time after which the solution will be impossible to represent: Assume that the grid size in v direction is Δv, and that function values are stored for $v = 0, \pm\Delta v, \pm 2\Delta v, \ldots, \pm N_v \Delta v$. The "wavelength" of the function $\cos(k_x x - k_x v t)$ is $\lambda_v = 2\pi/k_x t$ in the velocity direction. The Nyqvist theorem states the condition $\lambda_v/\Delta v > 2$ for storing the solution, which for the problem gives the condition $2\pi/k_x t\Delta v > 2$. This condition only holds for times $t < \pi/k_x\Delta v$. After this time it is impossible to represent the solution on the grid.

The recurrence effect [3] occurs at the time $T_c = 2\pi/k_x\Delta v$, which is the time for the values of the initial condition to re-appear on the numerical grid because of the Nyqvist theorem just described.

2.2.4 Some properties of the Fourier transformed system

In general $f(x,v,t)$ decreases as a Gaussian function $\sim \exp(-\alpha v^2)$ for large values of v. This behaviour guarantees that the inverse Fourier transformed function $\widehat{f}(x,\eta,t)$ is a smooth function in η; it is an analytic function for all complex η and therefore all η-derivatives are well-defined. This is favourable when the η derivative in Equation (2.26) is approximated by a numerical difference approximation.

The difference in behaviour for the Fourier transformed system compared to the untransformed system can be illustrated by the example in the previous section; taking the Fourier transform of the solution (2.30) in velocity space yields

$$\widehat{f}(x,\eta,t) = \frac{1}{\sqrt{2\pi}}\left\{ e^{-\eta^2/2} + \frac{A}{2}\left[\cos(k_x x)\left(e^{-(\eta-k_x t)^2/2} + e^{-(\eta+k_x t)^2/2} \right) + \mathrm{i}\sin(k_x x)\left(e^{-(\eta-k_x t)^2/2} - e^{-(\eta+k_x t)^2/2} \right) \right] \right\} \tag{2.31}$$

This function does not become oscillatory for large times. The $\exp[-(\eta-t)^2/2]$ and $\exp[-(\eta+t)^2/2]$ terms represent smooth wave packets moving away from the origin $\eta = 0$. Instead of becoming more *oscillatory*, the Fourier transformed solution becomes *wider* with increasing time.

Since the original distribution function $f(x,v,t)$ is real-valued, the Fourier transformed function $\widehat{f}(x,\eta,t)$ fulfils the relation

$$\widehat{f}(x,-\eta,t) = \left[\widehat{f}(x,\eta,t)\right]^* \tag{2.32}$$

where $*$ denotes complex conjugation. Therefore it is only necessary to solve the problem for positive η to obtain the solution for all η. For the derivatives one can easily show that the relation

$$\frac{\partial^n \widehat{f}(x,-\eta,t)}{\partial \eta^n} = (-1)^n \left[\frac{\partial^n \widehat{f}(x,\eta,t)}{\partial \eta^n} \right]^* \tag{2.33}$$

holds. Thus for even numbers of derivatives of the function $\widehat{f}$ with respect to η, the real part is even and the imaginary part is odd with respect to η. For odd numbers of derivatives of $\widehat{f}$, the opposite holds.

2.2.5 Invariants of the Vlasov-Poisson system

The one-dimensional system (2.24) with periodic boundary conditions describes a closed, non-dissipative system and has several invariants with respect to time, such as

$$S = \int_0^L \int_{-\infty}^{\infty} f^2(x,v,t)\,\mathrm{d}v\,\mathrm{d}x \tag{2.34}$$

$$N = \int_0^L \int_{-\infty}^{\infty} f(x,v,t)\,\mathrm{d}v\,\mathrm{d}x \tag{2.35}$$

$$p = \int_0^L \int_{-\infty}^{\infty} v f(x,v,t)\,\mathrm{d}v\,\mathrm{d}x \tag{2.36}$$

$$W = \int_0^L \left[\int_{-\infty}^{\infty} \frac{v^2}{2} f(x,v,t)\,\mathrm{d}v + \frac{E^2(x,t)}{2} \right] \mathrm{d}x \tag{2.37}$$

which describe the conservation of the energy norm (S), the total number of electrons (N), total momentum (p) and total energy (W), respectively. Instead of S as defined here, other entropy-like functionals can be used, for example where f^2 is replaced by $f\log(f)$ in the integrand of (2.34), with similar results [13]. The choice used here is convenient because it has its counter-part in the Fourier transformed space via the Parseval formula, see any mathematical handbook.

The corresponding invariants for the Fourier-transformed system (2.26)–(2.27) are:

$$S' = \int_0^L \int_{-\infty}^{\infty} \left| \widehat{f}(x,\eta,t) \right|^2 \mathrm{d}\eta \mathrm{d}x \tag{2.38}$$

$$N = \int_0^L 2\pi \widehat{f}(x,0,t)\,\mathrm{d}x \tag{2.39}$$

$$p = \int_0^L -\mathrm{i}2\pi \left[\frac{\partial \widehat{f}(x,\eta,t)}{\partial \eta} \right]_{\eta=0} \mathrm{d}x \tag{2.40}$$

$$W = \int_0^L \left\{ -\pi \left[\frac{\partial^2 \widehat{f}(x,\eta,t)}{\partial \eta^2} \right]_{\eta=0} + \frac{E^2(x,t)}{2} \right\} \mathrm{d}x \tag{2.41}$$

where a factor $1/2\pi$ has been omitted for S'. In the absence of an analytical "calibration" solution, it is important to check how well a numerical scheme conserves these invariants.

2.2.6 The well-posedness of the continuous problem

The equation system (2.26)–(2.27) is valid for all η on the real axis. In order to simulate numerically the system on an equidistant grid, one must however truncate the solution domain in the η direction, so that, for example, $-\eta_{\max} \le \eta \le \eta_{\max}$. Using the symmetry (2.32), this gives rise to boundaries at $\eta = 0$ and $\eta = \eta_{\max}$.

The boundary $\eta = \eta_{\max}$ must be treated with care so that it does not give rise to reflection of η "waves" or to instabilities. One strategy is to let outgoing waves travel out over the boundary and to give a boundary condition equal to zero for incoming waves. This gives a mathematically well-posed problem.

In order to explore this idea one can study the initial value model problem [cf. (2.28)]

$$\frac{\partial \widehat{f}}{\partial t} - \mathrm{i}\frac{\partial^2 \widehat{f}}{\partial x \partial \eta} = 0 \tag{2.42}$$

$$f(x,\eta,0) = f_0(x,\eta) \tag{2.43}$$

at the boundary $\eta = \eta_{\max}$. Fourier transforming Equation (2.42) in the x direction gives a new differential equation for the unknown function $\widetilde{f}(k_x, \eta, t)$,

$$\frac{\partial \widetilde{f}}{\partial t} + k_x \frac{\partial \widetilde{f}}{\partial \eta} = 0 \tag{2.44}$$

$$\widetilde{f}(k_x, \eta, 0) = \widetilde{f_0}(k_x, \eta) \tag{2.45}$$

The general solution to this equation is

$$\widetilde{f}(k_x, \eta, t) = \widetilde{f_0}(k_x, \eta - k_x t) \tag{2.46}$$

where $\widetilde{f_0}$ is an arbitrary function. The solution (2.46) describes outgoing waves at $\eta = \eta_{\max}$ for $k_x > 0$ and incoming waves for $k_x < 0$.

Assuming the initial condition to be zero at the boundary $\eta = \eta_{\max}$ at the time $t = 0$, a well-posed boundary condition is

$$\begin{cases} \frac{\partial \widetilde{f}}{\partial t} + k_x \frac{\partial \widetilde{f}}{\partial \eta} = 0, & k_x > 0, \eta = \eta_{\max} \\ \frac{\partial \widetilde{f}}{\partial t} = 0, & k_x \le 0, \eta = \eta_{\max} \end{cases} \tag{2.47}$$

which can be expressed as

$$\frac{\partial \widetilde{f}}{\partial t} + H(k_x) k_x \frac{\partial \widetilde{f}}{\partial \eta} = 0, \qquad \eta = \eta_{\max} \tag{2.48}$$

where H is the Heaviside step function

$$H(k_x) = \begin{cases} 1, & k_x > 0 \\ 0, & k_x \le 0 \end{cases} \tag{2.49}$$

The boundary condition (2.47–2.48) allows outgoing waves to pass over the boundary and to be lost, while incoming waves are set to zero. In this way, we avoid nonphysical waves from coming back from the artificial boundary. The loss of the outgoing waves corresponds to the loss of the finest structures in velocity space.

Introducing short notations for the spatial Fourier transform and inverse spatial Fourier transform as

$$\mathrm{F}\phi = \int_{-\infty}^{\infty} \phi(x) e^{-\mathrm{i}k_x x} \mathrm{d}x \tag{2.50}$$

and

$$\mathrm{F}^{-1}\widetilde{\phi} = \frac{1}{2\pi} \int_{-\infty}^{\infty} \widetilde{\phi}(k_x) e^{\mathrm{i}k_x x} \mathrm{d}k_x \tag{2.51}$$

respectively, then inverse Fourier transforming Equation (2.48) with respect to k_x gives the boundary condition for the original problem (2.42) as

$$\frac{\partial \widehat{f}}{\partial t} + \mathrm{F}^{-1} H(k_x) \mathrm{F} \left(-\mathrm{i} \frac{\partial^2 \widehat{f}}{\partial x \partial \eta} \right) = 0, \qquad \eta = \eta_{\max} \tag{2.52}$$

The projection operator $\mathrm{F}^{-1} H(k_x) \mathrm{F}$ projects a function onto the space of functions with only positive Fourier components in the x direction.

Problem (2.26) is treated according to the same idea,

$$\frac{\partial \widehat{f}}{\partial t} + \mathrm{F}^{-1} H(k_x) \mathrm{F} \left(-\mathrm{i} \frac{\partial^2 \widehat{f}}{\partial x \partial \eta} + \mathrm{i} \eta E \widehat{f} \right) = 0, \qquad \eta = \eta_{\max} \tag{2.53}$$

which prevents the $\mathrm{i}\eta E \widehat{f}$ term from producing spurious waves at the boundary.

The continuous problem is well-posed if the energy norm

$$\left\| \widehat{f} \right\|^2 = \int_{x=0}^{L} \int_{\eta=0}^{\eta_{\max}} \left| \widehat{f} \right|^2 \mathrm{d}\eta \mathrm{d}x = \int_{x=0}^{L} \int_{\eta=0}^{\eta_{\max}} \widehat{f} \widehat{f}^* \mathrm{d}\eta \mathrm{d}x \tag{2.54}$$

of the solution is bounded for all times. In the following, we prove that this norm is monotonically decreasing with time. Taking the time derivative of the norm gives

$$\frac{\mathrm{d} \left\| \widehat{f} \right\|^2}{\mathrm{d}t} = \int_{x=0}^{L} \int_{\eta=0}^{\eta_{\max}} \left(\widehat{f}^* \frac{\partial \widehat{f}}{\partial t} + \widehat{f} \frac{\partial \widehat{f}^*}{\partial t} \right) \mathrm{d}\eta \mathrm{d}x \tag{2.55}$$

and then replacing the time derivatives with the differential equation (2.26) gives

$$\begin{aligned}
\frac{\mathrm{d} \left\| \widehat{f} \right\|^2}{\mathrm{d}t} &= \int_{x=0}^{L} \int_{\eta=0}^{\eta_{\max}} \left[\widehat{f}^* \left(\mathrm{i} \frac{\partial^2 \widehat{f}}{\partial x \partial \eta} - \mathrm{i} \eta E \widehat{f} \right) + \right. \\
&\qquad \left. \widehat{f} \left(-\mathrm{i} \frac{\partial^2 \widehat{f}}{\partial x \partial \eta}^* + \mathrm{i} \eta E^* \widehat{f}^* \right) \right] \mathrm{d}\eta \mathrm{d}x \\
&= \int_{x=0}^{L} \int_{\eta=0}^{\eta_{\max}} \left[\mathrm{i} \left(\widehat{f}^* \frac{\partial^2 \widehat{f}}{\partial x \partial \eta} - \widehat{f} \frac{\partial^2 \widehat{f}}{\partial x \partial \eta}^* \right) + \mathrm{i} \widehat{f} \widehat{f}^* \underbrace{\left(E - E^* \right)}_{=0, \ (E \text{ real})} \right] \mathrm{d}\eta \mathrm{d}x \\
&= \mathrm{i} \int_{x=0}^{L} \int_{\eta=0}^{\eta_{\max}} \left[\frac{\partial}{\partial \eta} \left(\widehat{f}^* \frac{\partial \widehat{f}}{\partial x} \right) - \frac{\partial}{\partial x} \left(f \frac{\partial f^*}{\partial \eta} \right) \right] \mathrm{d}\eta \mathrm{d}x \\
&= \mathrm{i} \int_{x=0}^{L} \left[\widehat{f}^* \frac{\partial \widehat{f}}{\partial x} \right]_{\eta=0}^{\eta_{\max}} \mathrm{d}x - \mathrm{i} \int_{\eta=0}^{\eta_{\max}} \left[\widehat{f} \frac{\partial f^*}{\partial \eta} \right]_{x=0}^{L} \mathrm{d}\eta
\end{aligned} \tag{2.56}$$

where the last term vanishes due to periodic boundary conditions in the x direction, giving

$$\begin{aligned}\frac{\mathrm{d}\left\|\widehat{f}\right\|^2}{\mathrm{d}t} &= \mathrm{i}\int_{x=0}^{L}\left[\widehat{f}^*\frac{\partial\widehat{f}}{\partial x}\right]_{\eta=0}^{\eta_{\max}}\mathrm{d}x \\ &= \mathrm{i}\int_{x=0}^{L}\widehat{f}^*(x,\eta_{\max},t)\,\frac{\partial\widehat{f}}{\partial x}(x,\eta_{\max},t)\,\mathrm{d}x \\ &\quad -\mathrm{i}\int_{x=0}^{L}\widehat{f}^*(x,0,t)\,\frac{\partial\widehat{f}}{\partial x}(x,0,t)\,\mathrm{d}x\end{aligned} \tag{2.57}$$

The last term in (2.57) vanishes because, due to the symmetry (2.32), the imaginary part of the function is zero along the boundary $\eta = 0$:

$$\begin{aligned}-\mathrm{i}\int_{x=0}^{L}\widehat{f}^*(x,0,t)\frac{\partial\widehat{f}}{\partial x}(x,0,t)\mathrm{d}x &= -\mathrm{i}\int_{x=0}^{L}\widehat{f}^{(\mathrm{Re})}(x,0,t)\frac{\partial\widehat{f}^{(\mathrm{Re})}}{\partial x}(x,0,t)\mathrm{d}x \\ &= -\mathrm{i}\left[\frac{1}{2}\widehat{f}^{(\mathrm{Re})}(x,0,t)^2\right]_{x=0}^{L} = 0\end{aligned} \tag{2.58}$$

What remains is

$$\frac{\mathrm{d}\left\|\widehat{f}\right\|^2}{\mathrm{d}t} = \mathrm{i}\int_{x=0}^{L}\widehat{f}^*(x,\eta_{\max},t)\,\frac{\partial\widehat{f}}{\partial x}(x,\eta_{\max},t)\,\mathrm{d}x \tag{2.59}$$

Along the boundary $\eta = \eta_{\max}$, the boundary condition (2.53) is applied. This equation can formally be integrated with respect to time, which gives

$$\begin{aligned}&\widehat{f}(x,\eta_{\max},t) \\ &= \int_{t'=0}^{t}\mathrm{F}^{-1}H(k_x)\mathrm{F}\left[\mathrm{i}\frac{\partial^2\widehat{f}}{\partial x\partial\eta}(x,\eta_{\max},t') - \mathrm{i}\eta E(x,t')\widehat{f}(x,\eta_{\max},t')\right]\mathrm{d}t' \\ &= \mathrm{F}^{-1}H(k_x)\mathrm{F}\int_{t'=0}^{t}\left[\mathrm{i}\frac{\partial^2\widehat{f}}{\partial x\partial\eta}(x,\eta_{\max},t') - \mathrm{i}\eta E(x,t')\widehat{f}(x,\eta_{\max},t')\right]\mathrm{d}t' \\ &= \mathrm{F}^{-1}H(k_x)\mathrm{F}g(x,t)\end{aligned} \tag{2.60}$$

where

$$g(x,t) = \int_{t'=0}^{t}\left[\mathrm{i}\frac{\partial^2\widehat{f}}{\partial x\partial\eta}(x,\eta_{\max},t') - \mathrm{i}\eta E(x,t')\widehat{f}(x,\eta_{\max},t')\right]\mathrm{d}t' \tag{2.61}$$

The expression (2.60) inserted into (2.59) gives

$$\frac{\mathrm{d}\left\|\widehat{f}\right\|^2}{\mathrm{d}t} = \mathrm{i}\int_{x=0}^{L}\left[\mathrm{F}^{-1}H(k_x)\mathrm{F}g(x,t)\right]^*\frac{\partial}{\partial x}\left[\mathrm{F}^{-1}H(k_x)\mathrm{F}g(x,t)\right]\mathrm{d}x \tag{2.62}$$

Due to periodic boundary conditions in the x direction, the function g can be expanded into a Fourier series,

$$g(x,t) = \sum_{\omega=-\infty}^{\infty}\widehat{g}_\omega(t)e^{\mathrm{i}2\pi\omega\frac{x}{L}} \tag{2.63}$$

Taking the Fourier transform of this expression gives

$$\begin{aligned}\mathrm{F}g(x,t) &= \int_{-\infty}^{\infty}e^{-\mathrm{i}k_x x}\sum_{\omega=-\infty}^{\infty}\widehat{g}_\omega(t)e^{\mathrm{i}2\pi\omega\frac{x}{L}}\mathrm{d}x\\ &= \sum_{\omega=-\infty}^{\infty}\widehat{g}_\omega(t)\int_{-\infty}^{\infty}e^{\mathrm{i}\left(\frac{2\pi\omega}{L}-k_x\right)x}\mathrm{d}x\\ &= \sum_{\omega=-\infty}^{\infty}\widehat{g}_\omega(t)2\pi\delta_0\left(\frac{2\pi\omega}{L}-k_x\right)\end{aligned} \tag{2.64}$$

where δ_0 is the Dirac delta measure.

Multiplying this expression by the Heaviside function truncates the infinite sum as

$$\begin{aligned}H(k_x)\mathrm{F}g(x,t) &= \sum_{\omega=-\infty}^{\infty}\widehat{g}_\omega(t)2\pi H(k_x)\delta_0\left(\frac{2\pi\omega}{L}-k_x\right)\\ &= \sum_{\omega=1}^{\infty}\widehat{g}_\omega(t)2\pi\delta_0\left(\frac{2\pi\omega}{L}-k_x\right)\end{aligned} \tag{2.65}$$

since $\omega \le 0$ gives zero contribution to the sum.

Inverse Fourier transforming expression (2.65) gives

$$\begin{aligned}\mathrm{F}^{-1}H(k_x)\mathrm{F}g(x,t) &= \frac{1}{2\pi}\int_{-\infty}^{\infty}\left[H(k_x)\mathrm{F}g(x,t)\right]e^{\mathrm{i}k_x x}\mathrm{d}k_x\\ &= \sum_{\omega=1}^{\infty}\widehat{g}_\omega(t)\int_{-\infty}^{\infty}\delta_0\left(\frac{2\pi\omega}{L}-k_x\right)e^{\mathrm{i}k_x x}\mathrm{d}k_x\\ &= \sum_{\omega=1}^{\infty}\widehat{g}_\omega(t)e^{\mathrm{i}\frac{2\pi\omega}{L}x}\end{aligned} \tag{2.66}$$

which, inserted into (2.62), gives

$$\begin{aligned}\frac{\mathrm{d}\left\|\widehat{f}\right\|^2}{\mathrm{d}t} &= \mathrm{i}\int_{x=0}^{L}\left[\sum_{\omega=1}^{\infty}\widehat{g}_\omega(t)e^{\mathrm{i}\frac{2\pi\omega}{L}x}\right]^*\frac{\partial}{\partial x}\left[\sum_{\omega=1}^{\infty}\widehat{g}_\omega(t)e^{\mathrm{i}\frac{2\pi\omega}{L}x}\right]\mathrm{d}x \\ &= \mathrm{i}\int_{x=0}^{L}\left[\sum_{\omega=1}^{\infty}\widehat{g}^*_\omega(t)e^{-\mathrm{i}\frac{2\pi\omega}{L}x}\right]\left[\sum_{\omega=1}^{\infty}\widehat{g}_\omega(t)\mathrm{i}\frac{2\pi\omega}{L}e^{\mathrm{i}\frac{2\pi\omega}{L}x}\right]\mathrm{d}x \\ &= -\frac{2\pi}{L}\int_{x=0}^{L}\sum_{\omega=1}^{\infty}\widehat{g}^*_\omega(t)\widehat{g}_\omega(t)\omega\mathrm{d}x = -2\pi\sum_{\omega=1}^{\infty}\left|\widehat{g}_\omega(t)\right|^2\omega \le 0\end{aligned} \tag{2.67}$$

Thus we have proved that the energy norm is non-increasing with time, and therefore the continuous problem with the given boundary conditions is well-posed.

2.3 Numerics

2.3.1 Storage of the solution

This section discusses the number of grid points and the amount of data needed for storing the solution.

When storing the distribution function $f(x,v,t)$ on a grid there are two problems to keep in mind.

1. The function is defined for all velocities, but numerically one has to truncate the solution domain at some "high" velocity $v_{\max}$, where the function values have become small enough.

2. The function may contain oscillatory structures in the v direction, and one has to have a fine enough grid to represent these structures.

These two problems have their counterparts in the inverse Fourier transformed variables; a less localised function in v space leads to finer structures in η space, and finer structures in v space leads to a less localised function in η space. To be precise, the two problems are converted to

1. Assuming that the maximum velocity for particles is $v = v_{\max}$, then after Fourier transforming the function $f(x,v,t)$, the quantity $k_{\eta,\max} = v_{\max}$ will be the maximum wave number in η direction, and the minimum "wavelength" will then be $\lambda_{\eta,\min} = 2\pi/k_{\eta,\max} = 2\pi/v_{\max}$. According to the Nyqvist sampling theorem one needs at least two grid points per wavelength to represent the solution, so the grid size condition becomes $\Delta\eta < \lambda_{\eta,\min}/2 = \pi/v_{\max}$.

2. Assuming that the shortest "wavelength" to be resolved in the v direction is $\lambda_{v,\min}$, the highest wave number in the v direction becomes $k_{v,\max} = 2\pi/\lambda_{v,\min}$. After Fourier transformation, this gives a condition on the domain size in the η direction as $\eta_{\max} \geq k_{v,\max} = 2\pi/\lambda_{v,\min}$.

The number of grid points needed to represent the function $f(x,\eta,t)$ on the interval $0 \leq \eta \leq \eta_{\max}$ [for negative η one can use symmetry relation (2.32)] is then

$$N_\eta = \frac{\eta_{\max}}{\Delta\eta} > 2\frac{v_{\max}}{\lambda_{v,\min}} \tag{2.68}$$

For representing the original function $f(x,v,t)$ one needs to store the function values on the domain $-v_{\max} \leq v \leq v_{\max}$, with the grid size $\Delta v < \lambda_{v,\min}/2$ according to the sampling theorem. This gives the number of grid points to be stored in the v direction as

$$N_v = \frac{2v_{\max}}{\Delta v} > 4\frac{v_{\max}}{\lambda_{v,\min}} \tag{2.69}$$

Thus, in order to represent the original function $f(x,v,t)$ one needs to store twice as many grid points as compared to representing the Fourier transformed function $\widehat{f}(x,\eta,t)$. However, the function $\widehat{f}(x,\eta,t)$ is complex valued so the *amount of data* to store is the same for $\widehat{f}(x,\eta,t)$ as for $f(x,v,t)$.

2.3.2 Discretisation

We discretise the problem on a rectangular, equidistant grid with periodic boundary conditions in the x direction. In the η direction the grid starts at $\eta = 0$ and ends at some positive $\eta = \eta_{\max}$.

The approximate function values at the grid points are enumerated such that

$$\widehat{f}(x_i,\eta_j,t_k) \approx \widehat{f}^k_{i,j} \tag{2.70}$$

with

$$x_i = i\Delta x, \quad i = 0, 1, \ldots, N_x - 1 \tag{2.71}$$

$$\eta_j = j\Delta\eta, \quad j = 0, 1, \ldots, N_\eta \tag{2.72}$$

$$t_k = k\Delta t, \quad k = 0, 1, \ldots, N_t \tag{2.73}$$

where

$$\Delta x = \frac{L}{N_x} \tag{2.74}$$

$$\Delta\eta = \frac{\eta_{\max}}{N_\eta} \tag{2.75}$$

$$\Delta t = \frac{t_{\text{end}}}{N_t} \tag{2.76}$$

2.3.3 Pseudo-spectral methods

For a problem which has periodic solutions, it is sometimes efficient to use *pseudo-spectral* methods to calculate linear operators accurately.

The pseudo-spectral methods are based on *trigonometric interpolation* where trigonometric polynomials are interpolated to a function at the grid points. Assume the discretisation with N_x equidistant grid pints as described in Section 2.3.2. The discrete Fourier transform (DFT) pair form the quantities

$$\widehat{\phi}_\omega = \frac{1}{N_x}\sum_{j=0}^{N_x-1}\phi(x_j)\exp\left(-\mathrm{i}2\pi\omega\frac{j}{N_x}\right) \equiv \mathrm{DFT}\phi(x) \tag{2.77}$$

$$\phi_{N_x}(x) = \sum_{-(N_x/2-1)}^{N_x/2}\phi_\omega\exp\left(\mathrm{i}2\pi\frac{x}{L}\right) \equiv \mathrm{DFT}^{-1}\phi_\omega \tag{2.78}$$

where $\phi(x)$ is assumed to be a continuous function which is interpolated at the grid points x_j by the trigonometric interpolating polynomial $\phi_{N_x}(x)$. Thus the function $\phi(x)$ is *approximated* by $\phi_{N_x}(x)$.

A linear operator acting on $\phi(x)$ is approximated by the same linear operator acting on the interpolating polynomial $\phi_{N_x}(x)$. Linear operators which are translationally invariant (does not change with x) give rise to functions of ω which multiply $\widehat{\phi}_\omega$ in Equation (2.78). A general notation is therefore, for such a linear operator P and the corresponding function $p(\omega)$,

$$\mathrm{P}\phi(x) \approx \mathrm{P}\phi_{N_x}(x) = \mathrm{DFT}^{-1}p(\omega)DFT\phi(x) \tag{2.79}$$

The algorithm is: First the DFT is performed on $\phi(x)$, then the result is multiplied by $p(\omega)$, and then the inverse DFT is performed on this result to obtain the approximation.

In the case when $\mathrm{P} = \partial/\partial x$, then $p(\omega) = \mathrm{i}k_x(\omega)$, where $k_x = 2\pi\omega/L$. In the case when P represents an integration operator, then $p(\omega) = 1/\mathrm{i}k_x$ for $\omega \neq 0$ and $p(\omega) = 0$ for $\omega = 0$. The function $p(\omega) = H(k_x(\omega))$, where H is the Heaviside step function, is used for the boundary condition at $\eta = \eta_{\max}$. The corresponding operator P for this case is related to the Hilbert transform.

In what follows, the notation for the numerical approximation of the Fourier transform on the periodic functions will be

$$\mathrm{F}\phi \approx \mathrm{DFT}\phi \tag{2.80}$$

$$\mathrm{F}^{-1}\widehat{\phi} \approx \mathrm{DFT}^{-1}\widehat{\phi} \tag{2.81}$$

The discrete Fourier transforms are efficiently calculated by the fast Fourier transform (FFT) algorithms.

2.3.4 Numerical approximations

The Vlasov-Poisson system (2.26, 2.27), together with the boundary condition (2.53) at $\eta = \eta_{\max}$, is approximated by a semi-discretisation in x and η space. After that, time steps are taken with the fourth-order Runge-Kutta method.

In order to define the semi-discretisation, the equations are rewritten on the form

$$\frac{\partial \widehat{f}}{\partial t} = \mathrm{i}\frac{\partial^2 \widehat{f}}{\partial x \partial \eta} - \mathrm{i}\eta E \widehat{f}, \qquad 0 \le \eta < \eta_{\max}, 0 \le x < L \tag{2.82}$$

$$\frac{\partial E(x,t)}{\partial x} = 1 - 2\pi \widehat{f}(x,0,t) \tag{2.83}$$

$$\frac{\partial \widehat{f}}{\partial t} = \mathrm{F}^{-1} H(k_x) \mathrm{F} \left(\mathrm{i}\frac{\partial^2 \widehat{f}}{\partial x \partial \eta} - \mathrm{i}\eta E \widehat{f} \right), \eta = \eta_{\max}, 0 \le x < L \tag{2.84}$$

$$\widehat{f}(x+L,\eta,t) = \widehat{f}(x,\eta,t) \tag{2.85}$$

Equation (2.83) is solved numerically to obtain E, which is then used to calculate the right-hand sides in Equation (2.82) and (2.84); one can consider E as a *function* of f. The η and x derivatives in Equation (2.82) and (2.84) as well as the operator $\mathrm{F}^{-1} H(k_x) \mathrm{F}$ in Equation (2.84) are calculated numerically; the methods will be described below. By these approximations and after discretisations in x and η directions according to Section 2.3.2, the equation is approximated by the semi-discretisation

$$\frac{\partial \widehat{f}_{i,j}}{\partial t} = P(\widehat{f})_{i,j} \tag{2.86}$$

where P is a grid function representing the numerical approximation of the right-hand sides of Equation (2.82) and (2.84); the function P is a function of all $\widehat{f}_{i,j}$. The unknown $\widehat{f}_{i,j}$ is then discretised also in *time*, and the time-stepping is done with the well-known Runge-Kutta algorithm:

1. $F^{(1)}_{i,j} \leftarrow P(\widehat{f}^k), \quad \forall\, i,j$
2. $F^{(2)}_{i,j} \leftarrow P(\widehat{f}^k + F^{(1)}\Delta t/2), \quad \forall\, i,j$
3. $F^{(3)}_{i,j} \leftarrow P(\widehat{f}^k + F^{(2)}\Delta t/2), \quad \forall\, i,j$
4. $F^{(4)}_{i,j} \leftarrow P(\widehat{f}^k + F^{(3)}\Delta t), \quad \forall\, i,j$
5. $\widehat{f}^{k+1}_{i,j} \leftarrow \widehat{f}^k_{i,j} + \frac{\Delta t}{6}(F^{(1)}_{i,j} + 2F^{(2)}_{i,j} + 2F^{(3)}_{i,j} + F^{(4)}_{i,j}), \quad \forall\, i,j$

The steps needed for obtaining the approximation $P_{i,j}$ are:

1. Calculate the electric field numerically from Equation (2.83).
2. Calculate a numerical approximation of Equation (2.82), for all points *including the points along the boundary* $\eta = \eta_{\max}$.
3. Apply numerically the boundary condition (2.84) for the points along the boundary $\eta = \eta_{\max}$.

The periodic boundary condition (2.85) eliminates in practice the boundary at $x = L$. There is no need to store the function value corresponding to $x = L$, and, when needed, one uses the rule $\widehat{f}(x,\eta,t) = \widehat{f}(x-L,\eta,t)$ for $x \geq L$, and the rule $\widehat{f}(x,\eta,t) = \widehat{f}(x+L,\eta,t)$ for $x < 0$. The corresponding rules for the discrete case are $\widehat{f}_{i,j} = \widehat{f}_{i-N_x,j}$ for $i \geq N_x$ and $\widehat{f}_{i,j} = \widehat{f}_{i+N_x,j}$ for $i < 0$, respectively.

Due to the periodicity in the x direction, a pseudo-spectral method can be used to calculate the x derivatives in the Equations (2.82 – 2.84) accurately, as described in Section 2.3.3

By using the well-known relation for the Fourier transform

$$\frac{\partial \phi}{\partial x} = \mathrm{F}^{-1}\mathrm{F}\frac{\partial \phi}{\partial x} = \mathrm{F}^{-1}\mathrm{i}k_x\mathrm{F}\phi \tag{2.87}$$

the corresponding approximation of the x derivative, used in the discretised case (see Section 2.3.3), is

$$\frac{\partial \phi}{\partial x} \approx \mathrm{DFT}^{-1}\left[\mathrm{i}k_x\mathrm{DFT}(\phi)\right] \tag{2.88}$$

The integration of E is approximated by

$$E \approx \mathrm{DFT}^{-1}\left[\frac{1}{\mathrm{i}k_x}\mathrm{DFT}(1-2\pi\widehat{f}^k_{i,0})\right] \tag{2.89}$$

except for $k_x = 0$. The component corresponding to $k_x = 0$ is set equal to zero.

The numerical approximation of the x derivatives in Equation (2.82) and (2.84) with

$$\frac{\partial^2 \widehat{f}}{\partial x \partial \eta} = \frac{\partial}{\partial \eta}\left(\frac{\partial \widehat{f}}{\partial x}\right) \tag{2.90}$$

is performed as

$$\frac{\partial \widehat{f}}{\partial x} \approx \mathrm{DFT}^{-1}\left[\mathrm{i}k_x\mathrm{DFT}(\widehat{f}^k_{i,j})\right] \tag{2.91}$$

In the η direction, the derivative $v = \frac{\partial \widehat{f}}{\partial \eta}$ is calculated using the classical fourth order Padé scheme [6, 11]. For the inner points, the implicit approximation

$$v_{i,j-1} + 4v_{i,j} + v_{i,j+1} = \frac{3}{\Delta\eta}\left(\widehat{f}_{i,j+1} - \widehat{f}_{i,j-1}\right), \qquad j = 1, 2, \ldots, N_\eta - 1 \tag{2.92}$$

is used. A family of similar schemes exists [11].

At the boundary $\eta = 0$, the symmetry (2.32) and (2.33) is used to apply the same approximation of the derivative at the boundary as for the inner points. The relations $\widehat{f}_{i,-1} = \widehat{f}^*_{i,1}$ and $v_{i,-1} = -v^*_{i,1}$ give

$$-v^*_{i,1} + 4v_{i,0} + v_{i,1} = \frac{3}{\Delta\eta}\left(\widehat{f}_{i,1} - \widehat{f}^*_{i,1}\right) \tag{2.93}$$

or, for the real and imaginary parts,

$$v^{(\mathrm{Re})}_{i,0} = 0 \tag{2.94}$$

$$2v^{(\mathrm{Im})}_{i,0} + v^{(\mathrm{Im})}_{i,1} = \frac{3}{\Delta\eta}\widehat{f}^{(\mathrm{Im})}_{i,1} \tag{2.95}$$

respectively.

At the boundary $\eta = \eta_{\max}$, the one-sided approximation

$$v_{i,N_\eta} + 3v_{i,N_\eta-1} = -\frac{1}{2\Delta\eta}\left(-5\widehat{f}_{i,N_\eta} + 4\widehat{f}_{i,N_\eta-1} + \widehat{f}_{i,N_\eta-2}\right) \tag{2.96}$$

is used. This gives a truncation error of order $\Delta\eta^3$ at the boundary.

The equations (2.92), (2.93) and (2.96) form one tridiagonal equation system for each subscript $i = 0, 1, \ldots, N_x$, each system having N_η complex-valued unknowns. In practice the equation system can be split into systems for the real and imaginary parts separately.

At the boundary $\eta = \eta_{\max}$ the boundary condition (2.84) is applied, with the help of the approximation

$$\mathrm{F}^{-1}H(k_x)\mathrm{F}\phi(x, \eta_{\max}, t) \approx \mathrm{DFT}^{-1}\left[H(k_x)\mathrm{DFT}(\phi^k_{i,N_\eta})\right] \tag{2.97}$$

where $\phi(x, \eta_{\max}, t)$ is the right-hand side of (2.82) along the boundary $\eta = \eta_{\max}$ and ϕ^k_{i,N_η} its discrete approximation.

In order to reduce aliasing effects in the x direction, a sixth-order dissipative term is added to Equation (2.26), which changes into

$$\frac{\partial \widehat{f}}{\partial t} - \mathrm{i}\frac{\partial^2 \widehat{f}}{\partial x \partial \eta} + \mathrm{i}\eta E\widehat{f} - \delta(\Delta x)^4 \frac{\partial^6 \widehat{f}}{\partial x^6} = 0 \tag{2.98}$$

where the real constant δ is chosen to some small positive number. The sixth derivative is approximated with a centred second-order approximation.

2.3.5 Stability analysis

When solving the reduced model problem

$$\frac{\partial \widehat{f}}{\partial t} - \mathrm{i}\frac{\partial^2 \widehat{f}}{\partial x \partial \eta} = 0 \tag{2.99}$$

with an explicit scheme, the stability region is

$$\Delta t < \frac{\rho}{K_x K_\eta} \tag{2.100}$$

where $\rho = \sqrt{8}$ for the explicit Runge-Kutta scheme, and K_x and K_η are the maximum values of the approximations of wavenumbers produced by the numerical scheme in x and η direction, respectively.

In the x direction the spectral method gives a maximum value of the approximated wave number equal to

$$K_x = \frac{\pi}{\Delta x} \tag{2.101}$$

In the η direction the Padé scheme, applied to a continuous function, is

$$v(\eta - \Delta\eta) + 4v(\eta) + v(\eta + \Delta\eta) = \frac{3}{\Delta\eta}\left[\widehat{f}(\eta + \Delta\eta) - \widehat{f}(\eta - \Delta\eta)\right] \tag{2.102}$$

which, with $v(\eta) = \widetilde{v}\exp(\mathrm{i}k_\eta\eta)$ and $\widehat{f}(\eta) = \widetilde{f}\exp(\mathrm{i}k_\eta\eta)$, gives

$$\widetilde{v} = \mathrm{i}\frac{3}{\Delta\eta}\frac{\sin(k_\eta\Delta\eta)}{\left[2 + \cos(k_\eta\Delta\eta)\right]}\widetilde{f} \tag{2.103}$$

Hence, the maximum value of the approximated wavenumber in the η direction is

$$K_\eta = \max_{0 \le k_\eta\Delta\eta \le \pi}\left|\frac{3}{\Delta\eta}\frac{\sin(k_\eta\Delta\eta)}{\left[2 + \cos(k_\eta\Delta\eta)\right]}\right| = \frac{\sqrt{3}}{\Delta\eta} \tag{2.104}$$

where the maximum is obtained for $k_\eta\Delta\eta = 2\pi/3$.

Inserting the expressions for ρ, K_x and K_η into (2.100) then gives the choice of Δt as

$$\Delta t < \frac{\sqrt{8}}{\sqrt{3}\pi}\Delta x\Delta\eta \approx 0.52\Delta x\Delta\eta \tag{2.105}$$

for stability, disregarding the boundary conditions and the dissipative term. The Courant-Friedrich-Lewy (CFL) condition for stability of explicit schemes for hyperbolic equations can be found in text books on numerical analysis, for example

the text book by Strikwerda [21]. Even if the differential equation treated here is not a hyperbolic equation, it turns out to be convenient to define a generalised *CFL number*, so that the condition (2.105) can be expressed as

$$\Delta t = \mathrm{CFL}\frac{\sqrt{8}}{\sqrt{3}\pi}\Delta x \Delta \eta \tag{2.106}$$

where the positive CFL number

$$\mathrm{CFL} < 1 \tag{2.107}$$

for stability.

For the full problem, including the nonlinearity, dissipative term and boundaries, some numerical tests have shown that CFL = 0.8 gives stability while CFL = 0.9 gives instability; see also section 2.4 for the CFL numbers used there.

2.3.6 The conservation of particles

It is easily shown that the numerical scheme, including the Runge-Kutta time marching, conserves exactly the total number of particles (2.39), approximated by the formula

$$N = 2\pi \sum_{i=0}^{N_x-1} \widehat{f}^k_{i,0}\Delta x \tag{2.108}$$

The sum only picks up the zeroth Fourier component $\widehat{f}^k_{i,0}$, corresponding to $k_x = 0$, and that component is left unchanged since it vanishes in the term containing the x derivative in Equation (2.26) with the approximation (2.91). Along the boundary $\eta = 0$ the last term in (2.26) also vanishes. This result has been verified in the numerical experiments where it has been found that the number of particles are conserved by the numerical scheme up to the precision of the computer.

2.4 Numerical results

2.4.1 Reflections at the boundaries

In order to verify that waves are absorbed by the boundary at $\eta = \eta_{\max}$, and that thereby the recurrence phenomenon is reduced, a numerical experiment was carried out. The simulation domain was chosen to be $0 \le x \le 4\pi$, $0 \le \eta \le 20$. The number of grid points in the x direction was $N_x = 100$. The initial condition was chosen according to Equation (2.114) below with the amplitude $A = 0.0002$ and

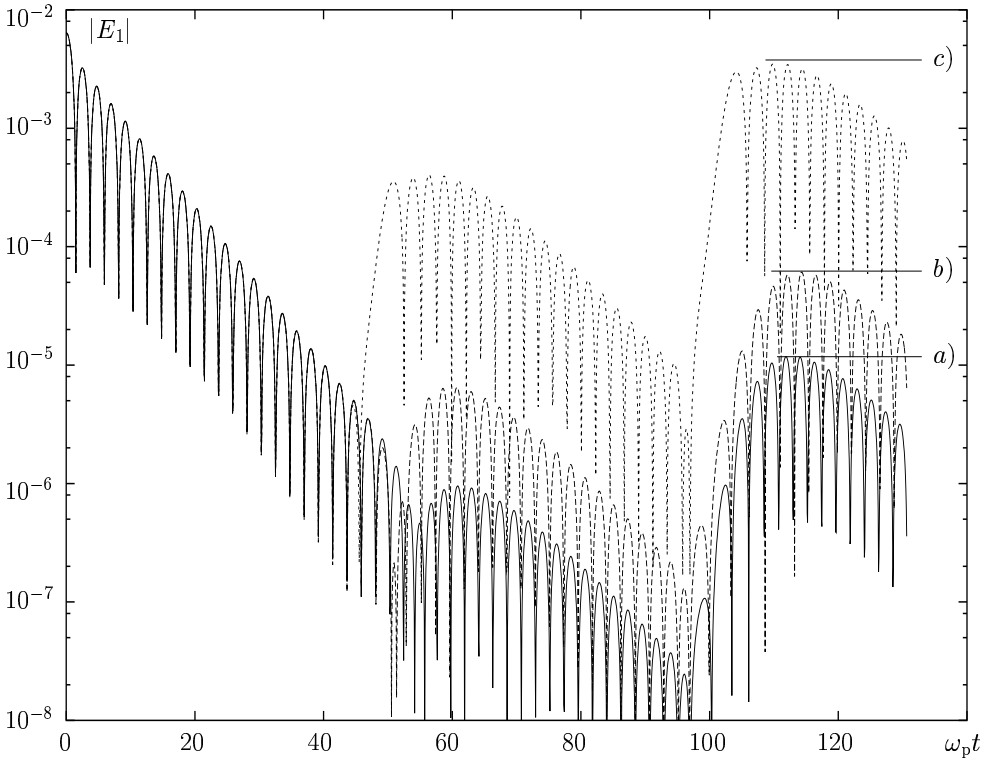


Figure 2.1: Reflections of waves against the boundary $\eta = \eta_{\max}$. The time development of the first spatial harmonic E_1 of the electric field.

the wavenumber $k_x = 0.5$; this choice assured that the wave was linearly damped according to known theory [20]. Three numerical experiments can be seen in Fig. 2.1, which displays the time development of the first spatial harmonic of the electric field. Curve *a*) shows a simulation with the outgoing wave boundary condition and with the grid size $\Delta\eta = 2/15$, and in curve *b*) the grid has been made coarser, $\Delta\eta = 2/10$. As a reference, curve *c*) shows a simulation with the commonly used [1, 4] Dirichlet-type of boundary condition $\widehat{f}(x, \eta_{\max}, t) = 0$ on the finer grid $\Delta\eta = 2/15$.

As can be seen in Fig. 2.1, the solutions are initially exponentially damped, and at about $t = 50$ one can see some smaller reflections from the boundary, followed by a much stronger reflection at $t = 100$. The solution on the finest grid *a*) with outgoing wave boundary conditions shows a reflected wave with the amplitude about $1/1000$ of the amplitude of the initial condition at $t = 0$, while the solution on the somewhat coarser grid *b*) shows a reflected wave with the ampli-

tude somewhat more than $1/100$ of the initial amplitude. With the Dirichlet-type boundary condition c), the reflected wave is of the same order in amplitude as the initial amplitude.

As should be apparent from this numerical investigation, the outgoing wave boundary condition prevents, to a large extent, waves from returning back and ruining the calculations, while the simple Dirichlet-type boundary condition leads to an almost total reflection of waves. These reflected waves lead to a similar detrimental effect as the recurrence phenomenon in real velocity space.

2.4.2 Nonlinear Landau damping

In order to ascertain that the numerical scheme reproduces some known non-linear effects, tests with larger initial amplitudes of the waves were carried out.

A simulation was performed with the initial condition according to, in terms of the original (x,v) variables,

$$f(x,v,0) = (1 + A\cos(k_x x))\, f_0(x,v) \tag{2.109}$$

where f_0 was chosen as

$$f_0(x,v) = \frac{1}{\sqrt{2\pi}} \exp\left\{ -\frac{1}{2}\left[v - \frac{\omega}{k_x} A\cos(k_x x) \right]^2 \right\} \tag{2.110}$$

This is an approximation of a sinusoidal wave moving in the rightward direction. The approximate dispersion relation for Langmuir waves yields $\omega = \sqrt{1+3k_x^2}$. In the inverse Fourier transformed variables, the initial condition is converted into

$$\widehat{f}(x,\eta,0) = [1 + A\cos(k_x x)]\, \widehat{f_0}(x,\eta) \tag{2.111}$$

where

$$\widehat{f_0}(x,\eta) = \frac{1}{2\pi} \exp\left[\mathrm{i}\frac{\omega}{k_x} A\cos(k_x x)\eta \right] \exp\left(-\frac{1}{2}\eta^2 \right) \tag{2.112}$$

which is the initial condition used in the simulation.

The wave number $k_x = 0.25$ and amplitude $A = 0.15$ were chosen. The simulation domain was set to $0 \le x \le 24\pi$, $0 \le \eta \le 30$ with $N_x = 300$, $N_\eta = 150$ and $\Delta t = 0.00875$ (CFL ≈ 0.33). The numerical dissipation was set to $\delta = 0.002$. In order to visualise the solution, it was Fourier transformed back to the real-valued function $f(x,v,t)$ and plotted in Fig. 2.2. One can see the process of electrons getting trapped and starting to oscillate in the potential wells of the wave. As expected,

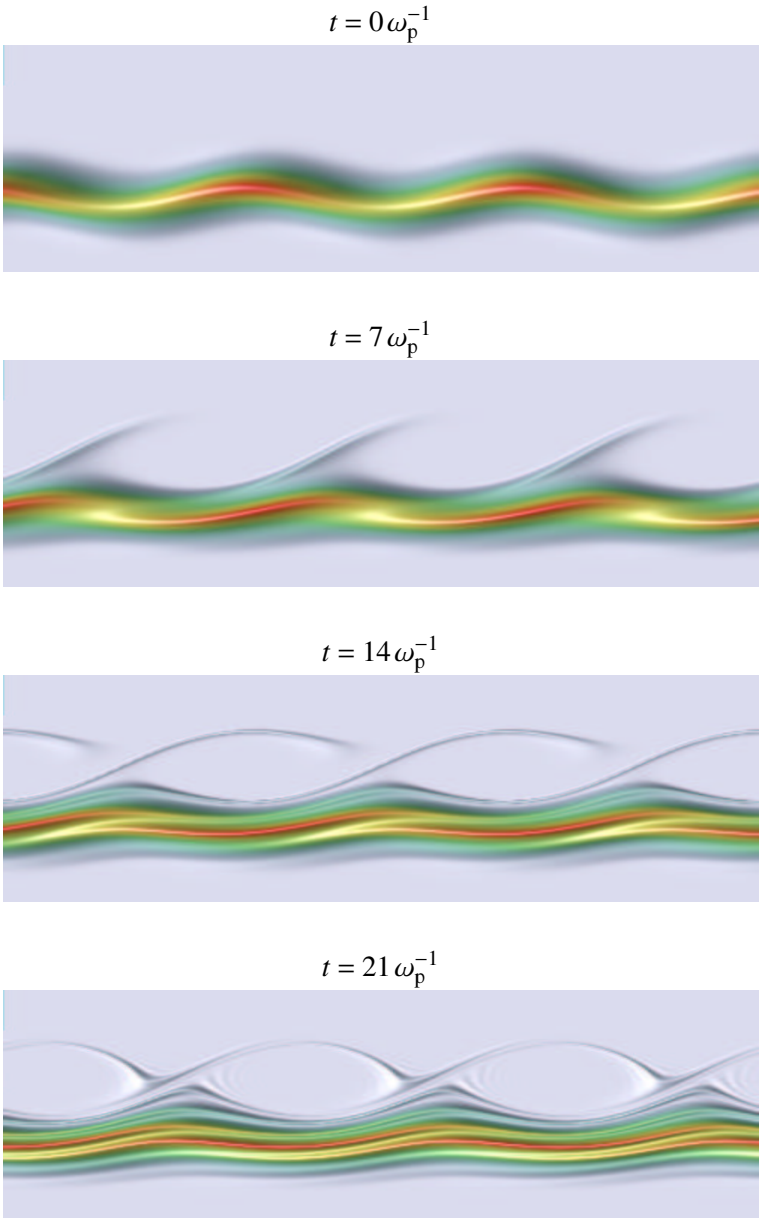


Figure 2.2: The development of an electrostatic wave in phase space (x, v) at four different times. One can see particles getting trapped in the potential wells of the wave.

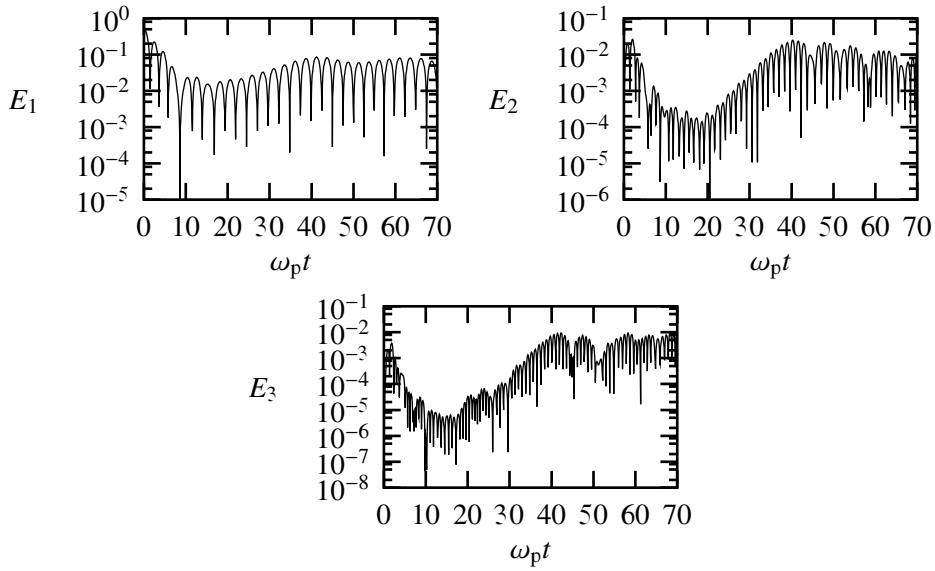


Figure 2.3: The first three spatial harmonics of the electric field.

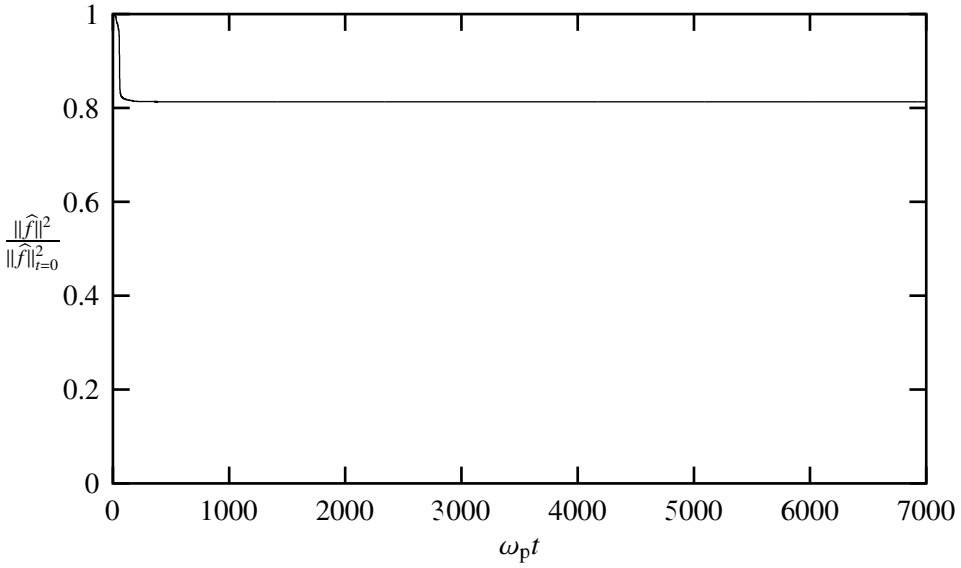


Figure 2.4: The relative change of the squared norm.

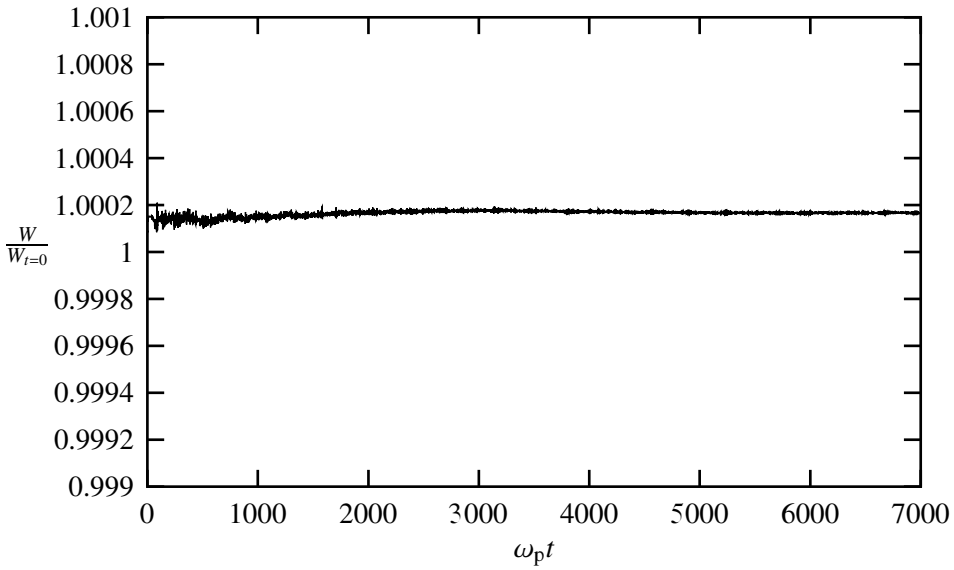


Figure 2.5: The relative change of the total energy.

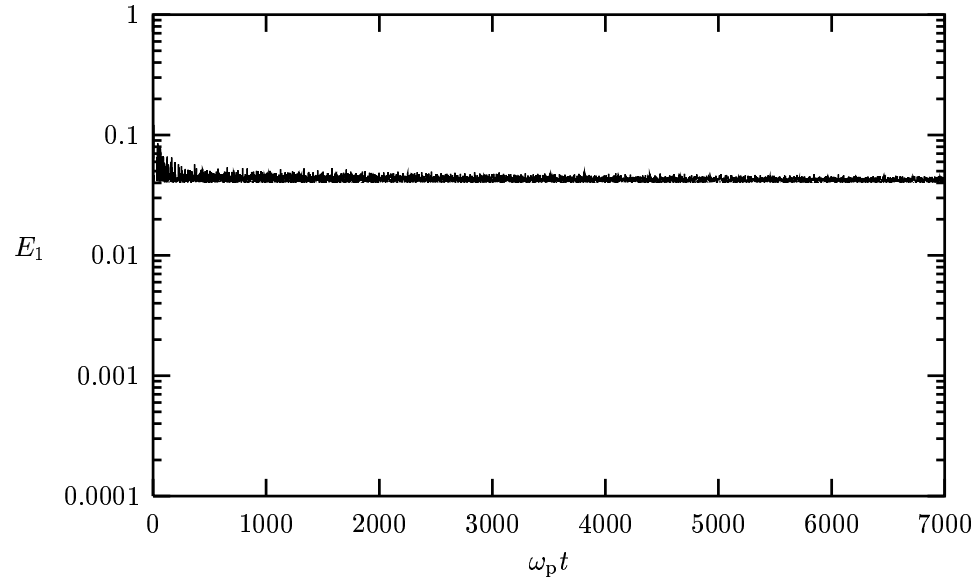


Figure 2.6: The long-term behaviour of the amplitude of the first spatial mode E_1 of the electric field (Function values below 0.04 have been removed).

the solution becomes increasingly oscillatory in the velocity (v) direction, due to the ballistic terms.

Another numerical experiment on nonlinear Landau damping was carried out, with the initial condition chosen to

$$f(x,v,0) = [1 + A\cos(k_x x)]\, f_0(v) \tag{2.113}$$

where $A = 0.5$, $k_x = 0.5$ and $f_0(v) = (2\pi)^{-1/2}\exp(-v^2/2)$; this is identical to one of the experiments carried out by Cheng and Knorr [3].

In the inverse Fourier transformed variables the initial condition becomes

$$\widehat{f}(x,\eta,0) = [1 + A\cos(k_x x)]\, \widehat{f}_0(\eta) \tag{2.114}$$

with $\widehat{f}_0(\eta) = (2\pi)^{-1}\exp(-\eta^2/2)$. The simulation domain was chosen as $0 \le x \le 4\pi$ and $0 \le \eta \le 30$ with $N_x = N_\eta = 100$, and the time domain was chosen as $0 \le t \le 70$ with $N_t = 5000$ (CFL ≈ 0.71). The numerical dissipation was set to $\delta = 0.001$. The amplitudes of the first three spatial components of the electric field were plotted against time in Fig. 2.3. One can note a strong exponential damping of the amplitudes from $t = 0$ to $t \approx 10$, in agreement with linear Landau theory. From $t \approx 20$ to $t \approx 40$ the modes grow exponentially, whereafter they oscillate around equilibria as the Landau damping enters the nonlinear regime [20]. These results are in excellent agreement with those obtained by Cheng and Knorr. These authors have made a deeper analysis of the results [3].

In order to test the long-term properties of our numerical method, a longer simulation was performed where the time domain was changed to $0 \le t \le 7000$ and $N_t = 500{,}000$, and the other parameters were kept unchanged. No numerical instability could be detected in the simulation. The squared energy norm (2.54) was numerically approximated by using a sum representation of the double integral and its value, relative to its initial value, was plotted against the time in Fig. 2.4. Initially it decreases from unity down to an equilibrium state at about 0.8130 after which it exhibits very small fluctuations. This decrease of the norm is due to waves passing over the boundary $\eta = \eta_{\max}$.

The time development of the total energy (2.41) is shown in Fig. 2.5. As can be seen, the energy is almost entirely conserved. In order to calculate the second derivative in the formula for the energy, a centred sixth-order scheme was used together with the symmetry (2.32).

The behaviour of the first spatial mode of the electric field is shown in Fig. 2.6. The electric field is initially damped from the value 0.5 down to somewhat below 0.05 where the damping almost vanishes. In a numerical long-time experiment carried out by Manfredi [13], this general behaviour of the solution could also be observed, for an almost similar problem.

CHAPTER 3

Parallel implementation of the Vlasov code

3.1 Introduction

In this chapter, the parallel implementation of the numerical algorithm is discussed, with emphasis on the methods used to optimise the code and on the parallel algorithms used. A numerical test is performed on two computer systems in order to check the portability and speedup of the parallel code.

3.2 Program structure

The algorithms for solving the one-dimensional Vlasov equation is implemented in Fortran 90. It consists of a main program and a number of subroutines called by the main program and by each other. The program has a hierarchy as shown in Figure 3.1; as can be seen in the figure, the names of the subroutine are object-like. Even if Fortran 90 is not fully object-oriented, it has turned out to be more convenient to think of the routines as *objects* instead of *subroutines* and *functions*. The numerical objects lower down in the program hierarchy are called by those higher up. Objects that use MPI commands for communication are marked with "MPI" in the figure. The source code is listed in Appendix (*A*).

The code is parallelised using MPI (Message Passing Interface), which makes the code highly portable; the code has been run on Hewlett-Packard and Sun parallel computers (described below) and also on the IBM SP2 system at the Royal Institute of Technology (KTH).

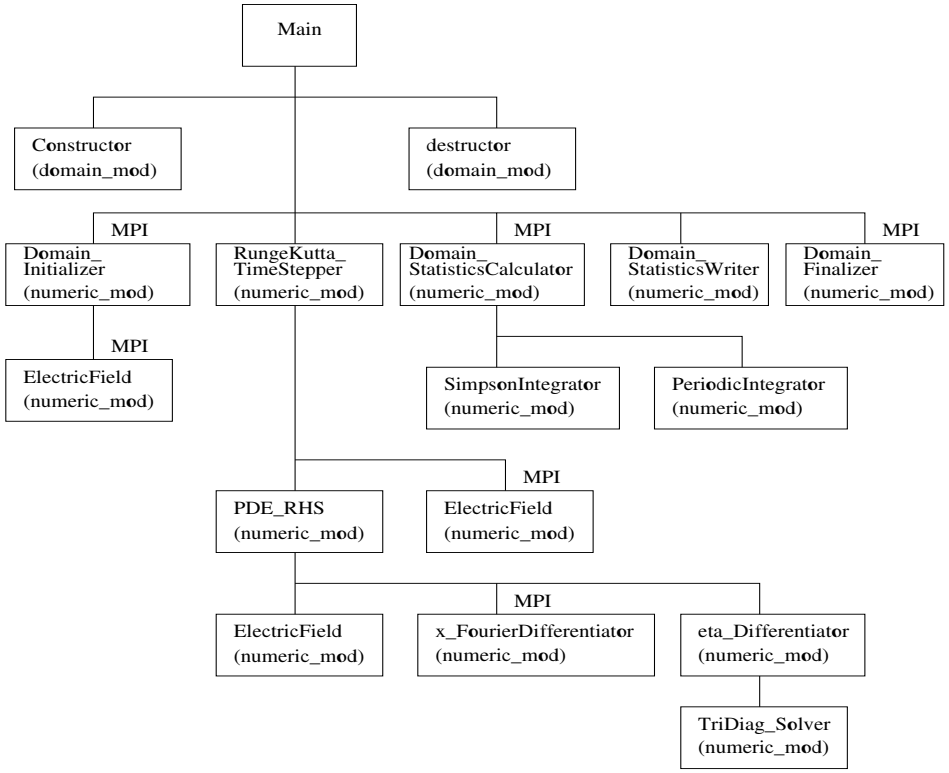


Figure 3.1: The program hierarchy

3.3 Time consuming subroutines

The most time-consuming subroutines could be identified with the simple profiling tool `gprof`. A short description of these subroutines is presented in the following list:

pde_rhs: Calculates the function values in the Runge-Kutta algorithm.

RungeKutta_TimeStepper: Performs a Runge-Kutta time step. It contains a number of loops which perform few arithmetic operations but accesses memory where large two-dimensional arrays are stored.

TriDiag_Solver: Solves a small tri-diagonal linear equation system. This routine performs only a few arithmetic operations but is called many times.

eta_Differentiator: Approximates the η derivative with a difference

approximation, where a tri-diagonal system has to be solved; for this task the *TriDiag_Solver* routine is called.

x_FourierDifferentiator: Approximates the x derivative with a spectral method. This is done by using the fast Fourier transform, supplied by a standard package.

The code was compiled on a single-processor machine with the `+O2` optimisation flag. The problem size was $N_x \times N_\eta = 100 \times 64$ and $N_t = 800$. The *gprof* utility gave the following results:

Subroutine	Seconds	Milliseconds/call	Calls
pde_rhs	11.34	3.54	3200
RungeKutta_TimeStepper	8.41	10.51	800
TriDiag_Solver	4.41	0.01	652800
eta_Differentiator	3.79	0.59	6400
x_FourierDifferentiator	1.89	0.59	3200

The *time* command gave the total user time 37.5 seconds for this problem.

3.4 Code optimisation before parallelisation

The single-processor code was investigated in order to make optimisations according to "the most common hints," that is, the most simple optimisations without any elaborate tricks:

pde_rhs: One division and one multiplication by constants in a loop, were replaced by one multiplication by a constant; this halved the execution time. Some loops were split into two, which gave some improvement. After these changes the time used by the routine decreased from 11.34 to 5.00 seconds.

RungeKutta_TimeStepper: This routine updates large arrays with new values. The first change was to replace the index notation supported by Fortran 90 by explicit loops over the arrays; this gave some improvement. Instead of updating two arrays in one loop, the loop was split into two loops updating one array each; this also gave some improvement. The third change was to replace the dynamic arrays, created with the `ALLOCATE` command, with static arrays, since it was not necessary for the array sizes to be changed during run-time. This gave a very large improvement of the speed. With these changes, the time used by this routine decreased from 8.41 to 0.96 seconds.

TriDiag_Solver: No changes were made.

eta_Differentiator: One division and one multiplication by constants in a loop, were replaced by with one multiplication by a constant. After this change, the time used by the routine decreased from 3.79 to 1.96 seconds.

x_FourierDifferentiator: No changes were made.

The code was again compiled with the `+O2` optimisation flag with the problem size $N_x \times N_\eta = 100 \times 64$ and $N_t = 800$, with the following results:

Subroutine	Seconds	Milliseconds/call	Calls
pde_rhs	5.00	1.83	3200
RungeKutta_TimeStepper	0.96	1.70	800
TriDiag_Solver	4.66	0.01	652800
eta_Differentiator	1.96	0.31	6400
x_FourierDifferentiator	1.94	0.61	3200

The *time* command now gave the total user time 23.6 seconds, to be compared with 37.5 seconds for the original code.

3.5 Parallelisation and partitioning of data

The computation domain was partitioned in the x direction. By this partitioning of data the following subroutines could be run in parallel; in fact these routines were not changed at all compared with the single processor code:

Subroutine
RungeKutta_TimeStepper
TriDiag_Solver
eta_Differentiator

The `pde_rhs` subroutine contains a treatment of a boundary condition which requires a forward and backward FFT on one vector in the x direction. This operation is performed serially by *Processor 0*; the communication is performed with the MPI commands `MPI_GATHER` and `MPI_SCATTER`.

The subroutine `ElectricField` that calculates the Electric field does so by a pseudo-spectral method which requires a forward and backward FFT on a vector plus a multiplication by a constant. This is performed serially by *Processor 0* and the communication is performed with the MPI commands `MPI_GATHER` and `MPI_SCATTER`.

A subroutine that performs statistics of the data needs to communicate partial sums to a total sum. The partial sums are sent to *Processor 0* with the `MPI_REDUCE` command.

The subroutine `x_FourierDifferentiator` calculates the x derivative on the whole domain. It does so by using a pseudo-spectral method. It requires a forward and backward FFT of N_η vectors, each of length N_x, and a multiplication with a constant. Instead of parallelising the FFT (or using a parallelised FFT package), the work is partitioned by first sending one vector to each processor, then letting each processor, in parallel, do the forward and inverse FFT and multiplication by a constant, and then distribute back the results, and then to repeat this procedure until all N_η vectors are processed. The communication is performed with the MPI commands `MPI_GATHER` and `MPI_SCATTER`. This part of the code puts the heaviest load on the communication.

3.6 Performance model

In order to make a precise performance model one should know how the computer systems are constructed and how communication between processors is performed in detail. The performance model presented here is, however, very simple and does not take into account the differences in computer systems; it only contains the very basic elements of a performance model. Therefore one can at best expect the real computation times and speedups to follow the model qualitatively, not quantitatively. For a more detailed discussion about performance models for different computer designs, see for example the book by Kumar et al. [10].

A simple model for the total execution time T for a problem is

$$T = T_{\text{comp}} + T_{\text{msg}} + T_{\text{idle}} \tag{3.1}$$

where the computation time is

$$T_{\text{comp}} = \tau_{\text{f}} N_{\text{f}} \tag{3.2}$$

Here τ_{f} is the time to perform one floating point operation and N_{f} is the number of floating point operations performed per processor.

The messaging time is

$$T_{\text{msg}} = \tau_{\text{s}} N_{\text{s}} + \tau_{\text{b}} L_{\text{p}} \tag{3.3}$$

where τ_{s} is the startup time or latency for a message, N_{s} is the number of messages to send per processor, τ_{b} is the time to send one byte and L_{p} is the total number of bytes per processor to be sent.

The idle time T_{idle} is the time the processor has to wait when another processor is doing work. This can happen due to bad load balance or when a purely serial part of an algorithm is processed by a processor.

With N_t, N_η and N_x being the number of grid points in the time, η and x directions, respectively, the problem size is

$$M_{\text{tot}} = 2N_tN_\eta N_x \tag{3.4}$$

where the factor 2 comes from the fact that the solution is complex valued. The problem size per processor is therefore

$$M_{\text{p}} = \frac{2N_tN_\eta N_x}{N_{\text{p}}} \tag{3.5}$$

where N_p is the number of processors.

The total number of floating point operations per processor is approximately

$$N_{\text{f}} = 25M_{\text{p}} \tag{3.6}$$

that is, on each data there are 25 arithmetic operations per time step.

The total amount of data to send per processor is approximately

$$L_{\text{p}} = 2M_{\text{p}}\left(1 - \frac{1}{N_{\text{p}}}\right) \tag{3.7}$$

This expression is for the *x_FourierDifferentiator* routine where the `MPI_GATHER` and `MPI_SCATTER` routines are used.

The total number of messages is approximately

$$N_{\text{s}} = (10 + N_\eta)N_t(N_{\text{p}} - 1) \tag{3.8}$$

Altogether, this gives the total time to solve the problem as

$$T = \frac{M_{\text{tot}}}{N_{\text{p}}}\left[25\tau_{\text{f}} + 2\tau_{\text{b}}\left(1 - \frac{1}{N_{\text{p}}}\right)\right] + \tau_{\text{s}}(10 + N_\eta)N_t(N_{\text{p}} - 1) + T_{\text{idle}} \tag{3.9}$$

This model predicts that the total execution time $T \to \tau_{\text{s}}(10 + N_\eta)N_t(N_{\text{p}} - 1)$, that is, the latency time τ_{s} will lead to a dominating and growing term in the limit $N_{\text{p}} \to \infty$.

The *relative speedup* in this model is

$$\begin{aligned} S &= \frac{T(1)}{T(N_{\text{p}})} \\ &= \frac{M_{\text{tot}}(25\tau_{\text{f}}) + T_{\text{idle}}}{\frac{M_{\text{tot}}}{N_{\text{p}}}\left[25\tau_{\text{f}} + 2\tau_{\text{b}}\left(1 - \frac{1}{N_{\text{p}}}\right)\right] + \tau_{\text{s}}(10 + N_\eta)N_t(N_{\text{p}} - 1) + T_{\text{idle}}} \end{aligned} \tag{3.10}$$

Its general behaviour is first an increase with increasing N_{p} and then a decrease as the latency term $\tau_{\text{s}}(10 + N_\eta)N_t(N_{\text{p}} - 1)$ becomes dominating in the denominator.

3.7 Numerical experiments and results

Identical codes were compiled on two different computer systems:

1. The Sun "Albireo" system at the Information Technology department at Uppsala University. It consists of two Sun Ultra Enterprise servers, each having 16 UltraSparcII (250 MHz) processors. The system has 4 GB RAM.

   The optimisation flag `-fast` was used for the compiler.

2. The Hewlett Packard "Zeipel" system at the Astronomy department at Uppsala University. It consists of an HP 9000/V25000 computer having 12 PA RISC (440 MHz) processors. The system has 2 GB RAM.

   The optimisation flags `+O2` and `+Omultiprocessor` was used in the compilation.

The problem size was varied in the x direction, while the size $N_\eta = 65$ and the number of time steps $N_t = 800$ were kept constant.

The following Tables show the time consumption measured in *minutes:seconds* on the two different computer systems used, as a function of the number of gridpoints N_x and the number of processors N_p. The time is the "real" time measured using the Unix `time` command.

The "Albireo" system:

	$N_x = 100$	$N_x = 200$	$N_x = 400$	$N_x = 800$	$N_x = 1600$
$N_\mathrm{p} = 1$	40	1:12	2:35	5:40	13:57
$N_\mathrm{p} = 2$	29	50	1:36	3:13	6:54
$N_\mathrm{p} = 4$	24	45	1:04	1:59	3:42
$N_\mathrm{p} = 5$	23	31	54	1:38	3:07
$N_\mathrm{p} = 8$	–	34	47	1:20	2:29
$N_\mathrm{p} = 10$	32	43	57	1:27	2:42
$N_\mathrm{p} = 20$	1:04	1:03	1:21	1:52	3:02

The "Zeipel" system:

	$N_x = 100$	$N_x = 200$	$N_x = 400$	$N_x = 800$	$N_x = 1600$
$N_\mathrm{p} = 1$	15	40	1:56	4:44	9:56
$N_\mathrm{p} = 2$	16	29	58	2:32	5:45
$N_\mathrm{p} = 4$	13	21	41	1:24	3:08
$N_\mathrm{p} = 5$	13	20	36	1:08	2:36
$N_\mathrm{p} = 8$	–	18	29	59	1:59
$N_\mathrm{p} = 10$	14	18	28	54	1:41

3.8 Comparison between the performance model and experiment

The numerical experiment showed that both computer systems show an almost linear speedup for few processors and for large problems. The general behaviour for the smaller problems are then that the speedups reach maximums for some N_p and then it decreases again; this behaviour is predicted by the theoretical formula (3.10) for the speedup.

CHAPTER 4

Linear dispersion laws and Landau damping

4.1 Introduction

In order to verify that the computer code developed produces accurate results, a numerical experiment was carried out to verify that the numerical algorithm reproduces the linear dispersion law for electrostatic waves according to known theory.

The dispersion law determines the relation between the angular frequency ω and the wave vector k_x for a wave on the form $\exp(\mathrm{i}(k_x x - \omega t))$. This law is a *macroscopic* law, as opposed to the *microscopic* full solution to the Vlasov equation; the calculation of the electric field involves taking an integral of $f(x,v,t)$ over all v, loosing the information of the exact behaviour of f in v direction. If k_x is real valued then ω may be real valued for undamped waves, or ω may be complex valued with a positive imaginary part for waves which grow exponentially with time or a negative imaginary part for exponentially decaying waves.

4.2 Approximate theoretical dispersion law

As is well known, linear electrostatic plasma waves may be damped even though there are no collisions that would normally dissipate energy from the wave. This is due to the Landau damping effect which reduces wave amplitudes through phase mixing [2]. Groups of electrons will become more and more uncorrelated with other groups of electrons, resulting in a diminishing of the collective wave electric field. In the macroscopic dispersion law, the microscopic origin of the Landau damping will be lost and one will only see the manifestation of it through an exponential damping of the wave amplitudes. Since the total energy of the system of electrons is constant, the potential energy of the electric field is converted into kinetic energy of the electrons. The *temperature* of the electrons is then raised,

even though no collisions have occurred.

No expression in terms of elementary functions exist for the dispersion law for electrostatic waves in a Maxwellian Vlasov plasma, but one often uses various analytic approximations of the dispersion law. For a Maxwellian plasma, with the zeroth order electron distribution proportional to $\exp(-v^2/2v_{\rm th}^2)$ and for waves with a higher phase velocity than thermal velocity, $\omega/k_x > v_{\rm th}$, the following approximation of the angular frequency

$$\omega = \omega_{\rm R} + {\rm i}\omega_{\rm I} \tag{4.1}$$

as a function of the wave vector k_x is often used:

$$\omega_{\rm R} = \omega_{\rm p}\sqrt{1+3\frac{k_x^2}{k_{\rm D}^2}} \tag{4.2}$$

$$\omega_{\rm I} = -\sqrt{\frac{\pi}{8}}\frac{k_{\rm D}^3}{k_x^3}\omega_{\rm p}\exp\left[-\frac{1}{2}\left(\frac{k_{\rm D}^2}{k_x^2}+3\right)\right] \tag{4.3}$$

where the Debye wave number is defined as $k_{\rm D} = \omega_{\rm p}/v_{\rm th}$. In units scaled into dimensionless form according to Section 2.2.2, the approximation becomes

$$\omega_{\rm R} = \sqrt{1+3k_x^2} \tag{4.4}$$

$$\omega_{\rm I} = -\sqrt{\frac{\pi}{8}}\frac{1}{k_x^3}\exp\left[-\frac{1}{2}\left(\frac{1}{k_x^2}+3\right)\right] \tag{4.5}$$

where $\omega_{\rm R}$ and $\omega_{\rm I}$ have been scaled to the plasma frequency $\omega_{\rm p}$ and k_x has been scaled to $k_{\rm D}$. One can note that for $k_x \ll 1$, the damping $\omega_{\rm I}$ becomes very small, while for $k_x \to 1$, the damping becomes larger and finally the approximation breaks down.

4.3 The numerical experiment

A numerical experiment was set up with the following parameters: $N_\eta = 128$, $\eta_{\rm max} = 15.0$ and $N_x = 50$. The simulation domain covered one wavelength in the x direction, $L = 2\pi/k_x$, where the wave number k_x was varied. The step-size in x direction thus varied with k_x as $\Delta x = L/N_x = 2\pi/(kN_x)$. For numerical stability the CFL number (see Section 2.3.5) was chosen to 0.8.

For the initial condition, a similar setup as in Section 2.4.2 was chosen. In terms of the original (x,v) variables, the initial function was

$$f(x,v,0) = [1 + A\cos(k_x x)]\, f_0(x,v) \tag{4.6}$$

k_x/k_D	ω_R/ω_p (Theor.)	ω_R/ω_p (Num.)	ω_I/ω_p (Theor.)	ω_I/ω_p (Num.)
0.10	1.0149	1.0152	-2.70×10^{-20}	—
0.15	1.0332	1.0326	-9.25×10^{-9}	—
0.20	1.0583	1.0639	-6.51×10^{-5}	—
0.25	1.0897	1.1059	-3.00×10^{-3}	-2.16×10^{-3}
0.30	1.1269	1.1600	-2.00×10^{-2}	-1.26×10^{-2}
0.35	1.1694	1.2210	-5.50×10^{-2}	-3.44×10^{-2}
0.40	1.2166	1.2852	-9.60×10^{-2}	-6.62×10^{-2}
0.45	1.2679	1.3494	-1.30×10^{-1}	-1.05×10^{-1}
0.50	1.3229	1.4200	-1.51×10^{-1}	-1.55×10^{-1}

Table 4.1: A comparison between the theoretical (Theor.) and numerical (Num.) relation between the angular frequency ω and the wave number k.

with

$$f_0(x,v) = \frac{1}{\sqrt{2\pi}} \exp\left\{ -\frac{1}{2}\left[v - \frac{\omega}{k_x} A\cos(k_x x)\right]^2 \right\} \tag{4.7}$$

which is an approximation of a sinusoidal wave moving in the rightward direction. The approximate dispersion relation for Langmuir waves yields $\omega = \sqrt{1+3k_x^2}$. In the inverse Fourier transformed variables, the initial condition is converted into

$$\widehat{f}(x,\eta,0) = [1 + A\cos(k_x x)]\,\widehat{f_0}(x,\eta) \tag{4.8}$$

where

$$\widehat{f_0}(x,\eta) = \frac{1}{2\pi} \exp\left[\mathrm{i}\frac{\omega}{k_x} A\cos(k_x x)\eta \right] \exp\left(-\frac{1}{2}\eta^2 \right) \tag{4.9}$$

This is the initial condition used in the simulation.

The amplitude of the wave was chosen to a small value, $A = 0.0002$, to assure that the wave was nearly linear. The time T_{10} for the electric field at the point $x = 0$ to pass through 10 periods in time was measured, by measuring the time elapsed between the first time for the electric field to change sign from negative to positive values and the 11th time to change sign. The real part of the angular frequency was then obtained as as $\omega_I = 2\pi \times 10/T_{10}$ (in dimensionless units). The damping was calculated by measuring the amplitude of the electric field, $|E_0|$, at the time T_0 for the first positive maximum and the amplitude for the maximum of the electric field, $|E_{10}|$, at the time $T_0 + T_{10}$. The damping rate was then obtained as $\omega_I = -\log(|E_{10}|/|E_0|)/T_{10}$.

4.4 Numerical results

The theoretical and numerical values of ω are listed in Table 4.1. For the smallest values of k_x, no damping could be measured in the simulations. The agreement is reasonably good for both the real and imaginary parts of the frequency. For $k_x \to 0$, the approximate law becomes exact, and in Table 4.1 one can see that the numerical and theoretical values of ω converge to each other for small k_x.

CHAPTER 5

Kinetic tunnelling through an ionospheric layer

5.1 Introduction

Three types of propagating waves can exist in an unmagnetised plasma. The most well-known type is the *electromagnetic wave*, which can also propagate in vacuum as radio waves and light. In the plasma, waves can also propagate as *electron plasma waves*, known as *Langmuir waves*, and as *ion acoustic waves*. These waves are of the same compression and rarefaction type as ordinary sound waves in air, but where the interaction is electromagnetic rather than mechanical. A *magnetised plasma* supports a multitude of wave types. The plasma surrounding the earth is magnetised by the *geomagnetic field* and is called the *ionosphere*.

At each height the ionospheric plasma possesses a resonance frequency, called the *plasma frequency*, whose value depends on the number of free electrons in the plasma. Radio waves that are sent vertically into this plasma are reflected back at the height where the frequency of the wave matches the local plasma frequency; see Figure 5.1. This property of the ionosphere is used for short wave radio communication over large distances. Radio waves are also used for probing the near-earth plasma.

During RF (Radio Frequency) ionospheric experiments, when a powerful radio wave is injected vertically into the overhead overdense ionospheric plasma, the level of electrostatic (Langmuir) electron waves and ion acoustic waves are strongly enhanced near the RF reflection point where the RF matches the local plasma frequency. The ionosphere will not only passively reflect the wave but will also "deform" the radio wave and give rise to radio waves on other frequencies than the frequency of the incident wave. This re-radiation mechanism was discovered in 1981 by Thidé et al. [22] and is now known as Stimulated Electromagnetic Emission (SEE).

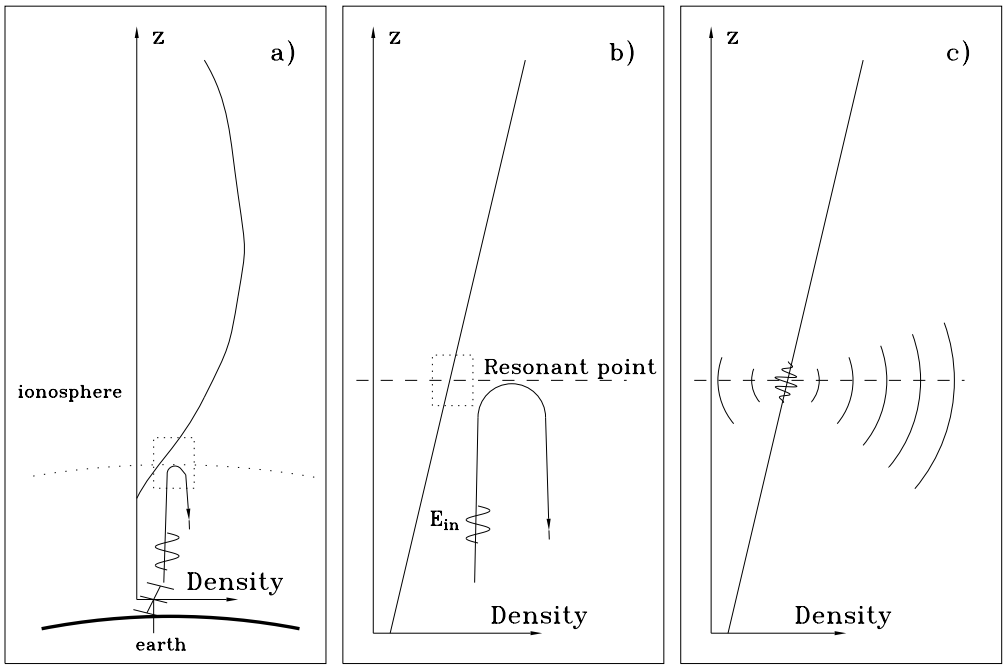


Figure 5.1: a) An electromagnetic wave is reflected from the ionosphere. b) The wave refracts at the resonant point. A weak electric field reaches the resonant point. c) The resonant point starts radiating within a range of frequencies around the original one.

Sometimes a weaker enhancement of Langmuir waves is observed at a higher altitude, on the far side of the ionospheric layer, where again the RF matches the local plasma frequency. This phenomenon has been ascribed by Isham et al. [7] and Mishin et al. [14] to a coupling between the injected electromagnetic O mode and the Z mode, where the Z mode propagates from the lower matching height to the higher matching height where it is strong enough to re-generate Langmuir and ion-acoustic waves.

An alternative mechanism has been pointed out by Revenchuk et al. [17, 18], in which Langmuir wave regeneration on the far side of the ionosphere is occurs via a kinetic plasma effect. The kinetic tunnelling effect is due to the trapping of electrons inside the electrostatic potential well, formed self-consistently in the ionospheric layer.

In this Chapter, the kinetic tunnelling effect is investigated numerically for the Gaussian ionospheric profile that has been investigated theoretically by Revenchuk

et al. [17, 18].

5.2 The self-consistent electrostatic potential

The partially ionised plasma in the ionosphere is produced mainly by solar radiation photoionization of neutral atoms and molecules. The balancing process is a recombination of ions and electrons to form neutrals. As a simple model, one may therefore assume that the steady state can be described by a given profile of immobile ions plus an equal amount of highly mobile electrons. To a good approximation, this model can be used for studying the dynamics of the electrons on time scales much shorter than the actual physical ion time scales.

For a given ion profile one can draw some conclusions regarding the equilibrium density of electrons and the governing electrostatic potential. The force balance of an isothermal electron "fluid" is given by the Navier-Stokes equation as

$$\mathbf{0} = -\frac{\boldsymbol{\nabla} p_\mathrm{e}}{n_\mathrm{e}} + e\boldsymbol{\nabla}\phi \tag{5.1}$$

where n_e is the electron number density, p_e is the electron pressure, ϕ is the electrostatic potential and $-e$ is the electron charge. The force exerted on the electrons by the pressure gradient is balanced by the electric field, which is expressed in terms of the electrostatic potential by the relation

$$\mathbf{E} = -\boldsymbol{\nabla}\phi \tag{5.2}$$

Approximating the electrons with an ideal, isothermal, and isotropic gas, the electron pressure is given by

$$p_\mathrm{e} = k_\mathrm{B} T_\mathrm{e} n_\mathrm{e} \tag{5.3}$$

where k_B is Boltzmann's constant and T_e is the constant electron temperature. The electrostatic potential is obtained from the first of Maxwell's equations (Poisson's equation) as

$$\boldsymbol{\nabla}\cdot\mathbf{E} = -\Delta\phi = \frac{e}{\varepsilon_0}(n_\mathrm{i} - n_\mathrm{e}) \tag{5.4}$$

which expresses the divergence of the electric field in terms of the total charge density.

The Equations (5.1), (5.3) and (5.4) can in principle be solved to obtain n_e, p_e and ϕ. A considerable simplification can be made if the *scale length* L of the

problem is "large." Then one may do the assumption of *quasi-neutrality*, i.e., that it holds approximately that

$$n_\mathrm{e} = n_\mathrm{i} \tag{5.5}$$

If the ion density and electron density would be *exactly* the same, there would not exist any electrostatic potentials or electric fields in the ionosphere, but that is not the case for, i.e., the Earth's ionosphere [8].

The expression (5.3) for the pressure and the approximate expression (5.5) of the electron density inserted into Equation (5.1) then yields

$$0 = -k_\mathrm{B} T_\mathrm{e} \frac{\boldsymbol{\nabla} n_\mathrm{e}}{n_\mathrm{e}} + e \boldsymbol{\nabla} \phi \tag{5.6}$$

which can be integrated to give the potential as

$$\phi = \frac{k_\mathrm{B} T_\mathrm{e}}{e} \log(n_\mathrm{i}) \tag{5.7}$$

plus some arbitrary constant, neglected here. The self-consistent electric field is obtained from the gradient of the potential as

$$\mathbf{E} = -\boldsymbol{\nabla} \phi = -\frac{k_\mathrm{B} T_\mathrm{e}}{e} \boldsymbol{\nabla} \log(n_\mathrm{i}) \tag{5.8}$$

In order to check whether or not the assumption of quasi-neutrality is correct, the expression (5.7) for the potential is inserted into (5.4) which, after some re-ordering of terms, gives

$$\frac{n_\mathrm{e}}{n_\mathrm{i}} = 1 + \frac{\varepsilon_0 k_\mathrm{B} T_\mathrm{e}}{e^2 n_\mathrm{i}} \Delta \log(n_\mathrm{i}) = 1 + r_\mathrm{D}^2 \Delta \log(n_\mathrm{i}) \frac{n_0}{n_\mathrm{i}} \tag{5.9}$$

where the Debye length is

$$r_\mathrm{D} = \sqrt{\frac{\varepsilon_0 k_\mathrm{B} T_\mathrm{e}}{e^2 n_0}} \tag{5.10}$$

At the peak of the F layer of the Earth's ionosphere, the temperature is of the order 1500 K and the electron density is of the order $10^{10}\,\mathrm{m}^{-3}$ [8], which gives the Debye length of the order 2–3 cm, while the scale-length of the ionosphere is obtained by the relation $1/L^2 = \left|\Delta \log(n_\mathrm{i})\right|$; the length-scale L being of the order 10–100 kilometres. These estimates indicate that the term $r_\mathrm{D}^2 \Delta \log(n_\mathrm{i})$ is of the order 10^{-12} to 10^{-14}, which is negligible compared to unity in Equation (5.9) at the vicinity of the peak. It follows that the assumption about quasi-neutrality is correct, and that the expressions for the potential (5.7) and the electric field (5.8) are good estimates.

For a model ionosphere with the ion profile having a Gaussian shape as a function of the altitude z, the results are especially simple. The ion profile is then given as

$$n_{\mathrm{i}} = n_0 \exp\left(-\frac{z^2}{L^2}\right) \tag{5.11}$$

where the maximum density n_0 and the scale length L are constants and the coordinate system is chosen so that the origin is at the peak of the density profile. Then it follows from the expressions for the potential (5.7) and electric field (5.8) that

$$\phi = \frac{k_{\mathrm{B}} T_{\mathrm{e}}}{e}\left(\log(n_0) - \frac{z^2}{L^2}\right) \tag{5.12}$$

and

$$E = 2\frac{k_{\mathrm{B}} T_{\mathrm{e}}}{e}\frac{z}{L^2} \tag{5.13}$$

respectively. This confining electric field is directed downwards for $z < 0$ and upwards for $z > 0$, preventing electrons from escaping from the density profile.

One can note that in the Gaussian-shaped ion profile, the electrons will perform harmonic oscillations in a parabolic potential well. The equation of motion for one electron is, according to Newton's second law,

$$m_{\mathrm{e}}\frac{\mathrm{d}^2 z(t)}{\mathrm{d}t^2} = -eE = -2k_{\mathrm{B}} T_{\mathrm{e}}\frac{z(t)}{L^2} \tag{5.14}$$

This ordinary differential equation gives sinusoidal solutions for the position $z(t)$ of the electrons, which oscillate with the angular frequency

$$\omega_{\mathrm{b}} = \sqrt{2\frac{k_{\mathrm{B}} T_{\mathrm{e}}}{m_{\mathrm{e}}}}\frac{1}{L} = \sqrt{2}\frac{v_{\mathrm{th}}}{L} \tag{5.15}$$

where $v_{\mathrm{th}} = \sqrt{k_{\mathrm{B}} T_{\mathrm{e}}/m_{\mathrm{e}}}$ is the thermal velocity of the electrons. The time it takes for an electron to perform one oscillation in the potential well is

$$T_{\mathrm{b}} = \frac{2\pi}{\omega_{\mathrm{b}}} = \sqrt{2}\pi\frac{L}{v_{\mathrm{th}}} \tag{5.16}$$

From Equation (5.14) it follows that the particle trajectories form ellipses in the (z, v) plane according to

$$\left[\frac{v(t)}{v_{\mathrm{th}}}\right]^2 + 2\left[\frac{z(t)}{L}\right]^2 = C \tag{5.17}$$

where $v(t) = \mathrm{d}z(t)/\mathrm{d}t$, for different constants C.

The assumption of quasi-neutrality, $n_\mathrm{e} = n_\mathrm{i}$, is valid as long as $|\Delta\phi| \ll en_\mathrm{i}/\varepsilon_0$, as can be seen in Equation (5.4). For the Gaussian ion profile, with ϕ given by (5.12) and n_i given by (5.11), this gives the condition

$$\frac{k_\mathrm{B}T_\mathrm{e}}{e}\frac{2}{L^2} \ll \frac{e}{\varepsilon_0}n_0 \exp\left(-\frac{z^2}{L^2}\right) \tag{5.18}$$

or, solving for z,

$$|z| < L\sqrt{\log\left(\frac{L^2}{2r_\mathrm{D}^2}\right)} \tag{5.19}$$

For values taken from the Earth's ionosphere, $r_\mathrm{D} \approx 1\,\mathrm{cm}$ and $L \approx 10\,\mathrm{km}$, the condition becomes $|z| < 5L$. In reality, Earths ionosphere is not very well described by a Gaussian, examples of measured data can be found in the book by Kelliey [8], but the estimation for the Gaussian still gives a hint that the assumption about quasi-neutrality is valid well away from the peak of the ion profile.

5.3 The one-dimensional Vlasov-Poisson system

The dynamics of the electrons in a collision-less plasma is described by the Vlasov equation. It suffices to study the one-dimensional problem if it is assumed that we have a vertically stratified ionosphere, with ions fixed in space, and with collective electron motion only along the gradient of the ion profile. In this case the Vlasov-Maxwell system can be reduced to the one-dimensional Vlasov-Poisson system, similar to the system in Equation (2.14),

$$\begin{aligned}
&\frac{\partial f}{\partial t} + v\frac{\partial f}{\partial z} - \frac{e}{m}[E + E_0(z,t)]\frac{\partial f}{\partial v} = 0 \\
&\frac{\partial E(z,t)}{\partial z} = \frac{e}{\varepsilon_0}\left[n_\mathrm{i} - \int_{-\infty}^{\infty} f(z,v,t)\,\mathrm{d}v\right]
\end{aligned} \tag{5.20}$$

where

$$n_\mathrm{i} = n_0 \exp\left(-\frac{z^2}{L^2}\right) \tag{5.21}$$

is the neutralising heavy ion density background, which is assumed to have a Gaussian shape with respect to the altitude z in this model problem, and $E_0(z,t)$ is an external electric field, resembling the perturbation of the plasma by a Langmuir wave.

As has been shown in Section 5.2, the ion profile has a associated a potential well, which will prevent the electrons from escaping the profile. If the total charge of positively charged ions and negatively charged electrons add up to zero, with all electrons confined to move in the vicinity of the profile, then one can assume that the electric field will vanish at $z = \pm\infty$. In the following, this will be assumed, i.e., that the amount of electrons escaping the profile is negligible.

5.4 The numerical setup

5.4.1 Numerical boundary conditions

Periodic boundary conditions are assumed for the function $\widehat{f}$. This is unphysical, but if the boundaries are far away from the ion profile the densities at the boundaries will be negligible and the problem can be treated as a periodic problem. The periodic boundary conditions makes it possible to calculate the spatial derivative with the pseudo-spectral method, described in Chapter 2.

As a model for the *electric field*, it is assumed that there is no net charge to the right or to the left of the computational domain. If the total electric charge of the whole simulation domain is zero, then the electric field adds up to zero at the left and at the right boundaries.

Integration of the electric field gives an arbitrary time-dependent constant, which can be determined so that the electric field is zero at the boundaries. In practice, the electric field is calculated in two steps:

1. First calculate the electric field with the pseudo-spectral method, described in Chapter 2.

2. Then subtract the value of the electric field at $z = 0$ from the electric field at all points.

By this two-step process, it is possible to calculate the electric field with a pseudo-spectral method, with high accuracy.

5.4.2 The Fourier transformed, dimensionless system

In order to use the method based on the Fourier-transform technique described in Chapter 2, the Vlasov-Poisson system is Fourier transformed analytically and is written in a dimensionless form, in a similar manner as in Section 2.2.2.

By using the Fourier transform pair

$$f(z,v,t) = \int_{-\infty}^{\infty} \widehat{f}(z,\eta,t)\mathrm{e}^{-\mathrm{i}\eta v}\,\mathrm{d}\eta \tag{5.22}$$

$$\widehat{f}(z,\eta,t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} f(z,v,t)e^{\mathrm{i}\eta v}\mathrm{d}v \tag{5.23}$$

equation (5.20) is transformed into

$$\frac{\partial \widehat{f}}{\partial t} - \mathrm{i}\frac{\partial^2 \widehat{f}}{\partial z \partial \eta} + \mathrm{i}\eta\frac{e}{m}[E + E_0(z,t)]\widehat{f} = 0 \tag{5.24}$$

$$\frac{\partial E(z,t)}{\partial z} = \frac{e}{\varepsilon_0}\left[n_0 \exp\left(-\frac{z^2}{L^2}\right) - 2\pi\widehat{f}(z,0,t)\right]$$

In preparation for a numerical simulation, the systems (5.20) and (5.24) are cast into dimensionless form by a scaling of variables (see also Section 2.2.2): the time t is scaled to the inverse of the plasma frequency at the peak of the ion profile, $\omega_{\mathrm{p}}^{-1} = \sqrt{\varepsilon_0 m/(n_0 e^2)}$, the velocity v is scaled to the thermal velocity v_{th}; the new variable η is then scaled to the inverse of the thermal velocity, and the spatial variable z and the scale length L is scaled to the Debye length $r_{\mathrm{D}} = v_{\mathrm{th}}\omega_{\mathrm{p}}^{-1}$. Finally, the function $\widehat{f}$ is scaled to the background density n_0, the function f is scaled to n_0/v_{th} and the electric fields E and E_0 are scaled to the quantity $v_{\mathrm{th}}\sqrt{n_0 m/\varepsilon_0}$. By this scaling of variables, the systems (5.20) and (5.24) attain the dimensionless form

$$\frac{\partial f}{\partial t} + v\frac{\partial f}{\partial z} - [E + E_0(z,t)]\frac{\partial f}{\partial v} = 0 \tag{5.25}$$

$$\frac{\partial E(z,t)}{\partial z} = \exp\left(-\frac{z^2}{L^2}\right) - \int_{-\infty}^{\infty} f(z,v,t)\,\mathrm{d}v \tag{5.26}$$

and

$$\frac{\partial \widehat{f}}{\partial t} - \mathrm{i}\frac{\partial^2 \widehat{f}}{\partial z \partial \eta} + \mathrm{i}\eta[E + E_0(z,t)]\widehat{f} = 0 \tag{5.27}$$

$$\frac{\partial E(z,t)}{\partial z} = \exp\left(-\frac{z^2}{L^2}\right) - 2\pi\widehat{f}(z,0,t) \tag{5.28}$$

respectively.

The externally applied perturbing electric field is assumed to take the form

$$E_0(z,t) = \alpha \exp\left[-\left(\frac{z - z_0}{L_0}\right)^2\right]\sin(\omega t) \tag{5.29}$$

where the constants α, z_0, L_0 and ω are chosen so that the weak perturbation has an angular frequency w less than the plasma frequency ω_p at the peak of the profile, and so that the perturbation is centred at the location z_0 where the local plasma frequency coincides with ω.

5.4.3 Perturbation form

If the perturbing electric field $E_0(x,t)$ is weak, one can assume that the time-dependent solution will deviate very little from the steady-state solution, and it was shown in Section 5.2 that for the steady-state solution, the electron density deviates very little from the ion density due to the quasi-neutrality.

Therefore it is convenient to write the equation in *perturbation form*, where one solves an equation for the perturbation only, and not for the bulk of the plasma. The advantage with this form is that one avoids large cancellation errors and truncation errors when making numerical approximations. For example, calculating the electric field involves calculating differences between ion and electron densities which are almost equal, with the risk of cancellation errors.

With the change of the variable $\widehat{f}(x,\eta,t)$ to a new variable $\widehat{f}^{(1)}(x,\eta,t)$ according to

$$\widehat{f} = \frac{1}{2\pi}\exp\left(-\frac{z^2}{L^2}\right)\exp\left(-\frac{\eta^2}{2}\right) + \widehat{f}^{(1)} \tag{5.30}$$

one obtains the equation system for the new unknown variable as

$$\begin{aligned}&\frac{\partial \widehat{f}^{(1)}}{\partial t} - \mathrm{i}\frac{z\eta}{\pi L^2}\exp\left(-\frac{\eta^2}{2}\right) - \mathrm{i}\frac{\partial^2 \widehat{f}^{(1)}}{\partial x \partial \eta} \\ &\qquad + \mathrm{i}(E + E_0(z,t))\eta\left[\frac{1}{2\pi}\exp\left(-\frac{z^2}{L^2}\right)\exp\left(-\frac{\eta^2}{2}\right) + \widehat{f}^{(1)}\right] = 0\end{aligned} \tag{5.31}$$

$$\frac{\partial E}{\partial z} = -2\pi \widehat{f}^{(1)}(z,0,t) \tag{5.32}$$

The new unknown electron distribution function $\widehat{f}^{(1)}(z,\eta,t)$ describes the *deviation* from a completely neutral and isothermal plasma.

5.4.4 The initial condition

If one would assume the initial condition $\widehat{f}^{(1)} = 0$ everywhere at time $t = 0$, then large transient oscillations would appear as the initially totally neutral plasma would build up a self-consistent electric field, created by a small deviation from neutrality.

As shown in Equation (5.9), there is a small difference between the ion and electron densities, which turns out to be important. A heuristic approach to construct an initial condition, which reduces the transient oscillations, has been to set the initial condition to

$$\widehat{f}^{(1)} = \frac{\partial^2}{\partial z^2} \log \left[\frac{\alpha_1}{L^2} + \exp\left(-\frac{z^2}{L^2} \right) \right] \frac{1}{2\pi} \exp\left(-\frac{\eta^2}{2} \right) \tag{5.33}$$

at time $t = 0$, for some number $\alpha_1 \approx 10$. Far from the profile, this function tends to zero, and it will give the right amount of net charge near the peak of the profile. The total contribution to the net charge will almost add up to zero, since the integral of the function will be close to zero if the limits are far from the origin.

Even if the transient oscillations are strongly reduced with this initial condition, they are not eliminated. A more precise procedure would be to numerically solve the nonlinear boundary value problem for the potential (in dimensionless variables),

$$-\frac{\mathrm{d}^2\phi}{\mathrm{d}z^2} = n_\mathrm{i}(z) - \exp(\phi) \tag{5.34}$$

with the boundary conditions $\mathrm{d}\phi/\mathrm{d}z = 0$ at the boundaries, as as can derived from the original nonlinear equation system (5.1) – (5.4). The second derivative of the potential with respect to z then gives the net charge. This would be an improvement of the existing numerical model.

5.5 Numerical experiments

5.5.1 Parameters used in the numerical experiments

Numerical experiments were carried out for three values of the scale length of the ion profile:

1. The scale length $L = 80\, r_\mathrm{D}$: The simulation domain was $z = -400\, r_\mathrm{D}$ to $z = 400\, r_\mathrm{D}$. The other parameters were chosen $\Delta z = 2.0\, r_\mathrm{D}$, $N_t = 4200$ and $t_\mathrm{end} = 764\, \omega_\mathrm{p}^{-1}$ The Fourier variable was going from $\eta = 0\, r_\mathrm{D}^{-1}$ to $\eta = \eta_\mathrm{max} = 50\, r_\mathrm{D}^{-1}$, and with the grid-size $\Delta\eta = 0.25\, r_\mathrm{D}^{-1}$. For the electric field, the constants were chosen to $\alpha = 10^{-4}\, v_\mathrm{th} \sqrt{n_0 m_\mathrm{e}/\varepsilon_0}$, $\omega = 0.9\, \omega_\mathrm{p}$, $z_0 = -39\, r_\mathrm{D}$ and $L_0 = 10\, r_\mathrm{D}$.

2. The scale length $L = 400\, r_\mathrm{D}$: The simulation domain was $z = -2000\, r_\mathrm{D}$ to $z = 2000\, r_\mathrm{D}$, the spatial grid size $\Delta z = 2.0\, r_\mathrm{D}$, the number of time steps $N_t = 4200$ and the end time $t_\mathrm{end} = 3275\, \omega_\mathrm{p}^{-1}$. The Fourier variable was going from $\eta = 0\, r_\mathrm{D}^{-1}$ to $\eta = \eta_\mathrm{max} = 120\, r_\mathrm{D}^{-1}$, and with the grid size $\Delta\eta = 0.30\, r_\mathrm{D}^{-1}$. For the electric field, the constants were chosen to $\alpha = 10^{-4}\, v_\mathrm{th} \sqrt{n_0 m_\mathrm{e}/\varepsilon_0}$, $\omega = 0.9\, \omega_\mathrm{p}$, $z_0 = -195\, r_\mathrm{D}$ and $L_0 = 20\, r_\mathrm{D}$.

3. The scale length $L = 800\,r_D$: The simulation domain was $z = -4000\,r_D$ to $z = 4000\,r_D$, the spatial grid size $\Delta z = 2.0\,r_D$, the number of time steps $N_t = 15000$ and the end time $t_{end} = 2729\,\omega_p^{-1}$. The Fourier variable was going from $\eta = 0\,r_D^{-1}$ to $\eta = \eta_{max} = 200\,r_D^{-1}$, and with the grid size $\Delta\eta = 0.25\,r_D^{-1}$. For the electric field, the constants were chosen to $\alpha = 10^{-4}\,v_{th}\sqrt{n_0 m_e/\varepsilon_0}$, $\omega = 0.9\,\omega_p$, $z_0 = -390\,r_D$ and $L_0 = 20\,r_D$. The simulation was run on 10 processors on the Sun system (see Section 3.7), at the department of Information Technology, Uppsala.

The numerical solution was Fourier transformed back to velocity space and visualised in Figure 5.2 and 5.3.

As a measure of the strength of tunnelling, the quantity $\max|n_2|/\max|n_1|$ was used, where $\max|n_1|$ is the maximum perturbation of the electron density to the left of the peak (i.e., for $z < 0$), and $\max|n_2|$ is the maximum perturbation to the right of the peak (i.e., for $z > 0$) This measure of the tunnelling strength was plotted against time in Figure 5.4 for different scale lengths L.

5.5.2 The numerical results

One can make a physical interpretation of the numerical results as follows:

1. When the perturbation is initiated at the turning point of the plasma, with the frequency of the perturbation equal to the local plasma frequency, electrostatic Langmuir waves are created, which start to propagate down the profile towards regions with lower ion density. This can be seen in Figure 5.2*a* between $z = -40\,r_D$ and $z = -80\,r_D$ and the velocities between $v = -3.5\,v_{th}$ to $v = +2.5\,v_{th}$, where the particles show a macroscopic wave structure, i.e., the structure is correlated for particles with both negative and positive velocities to form electrostatic waves travelling from the right to the left.

   For the larger profile $L = -400\,r_D$, the same macroscopic wave structure can be seen in Figure 5.3*a* between $z = -200\,r_D$ to $z = -400\,r_D$ and velocities between $v = -4\,v_{th}$ and $v = +3\,v_{th}$.

2. The phase velocities of the left-going waves decrease as they enter plasma with lower density, and the waves start to interact with the left-going particles moving with approximately the same velocity as the phase velocity of the wave. The particles are accelerated, and the energy of the wave is converted into energy of particle motion in the process known as Landau damping (referring to the damping of the wave). The motion of the accelerated particles with negative velocities v is not correlated with the particles having positive velocities, as can be seen in Figure 5.2*a* between $z = -80\,r_D$ to $z = -160\,r_D$

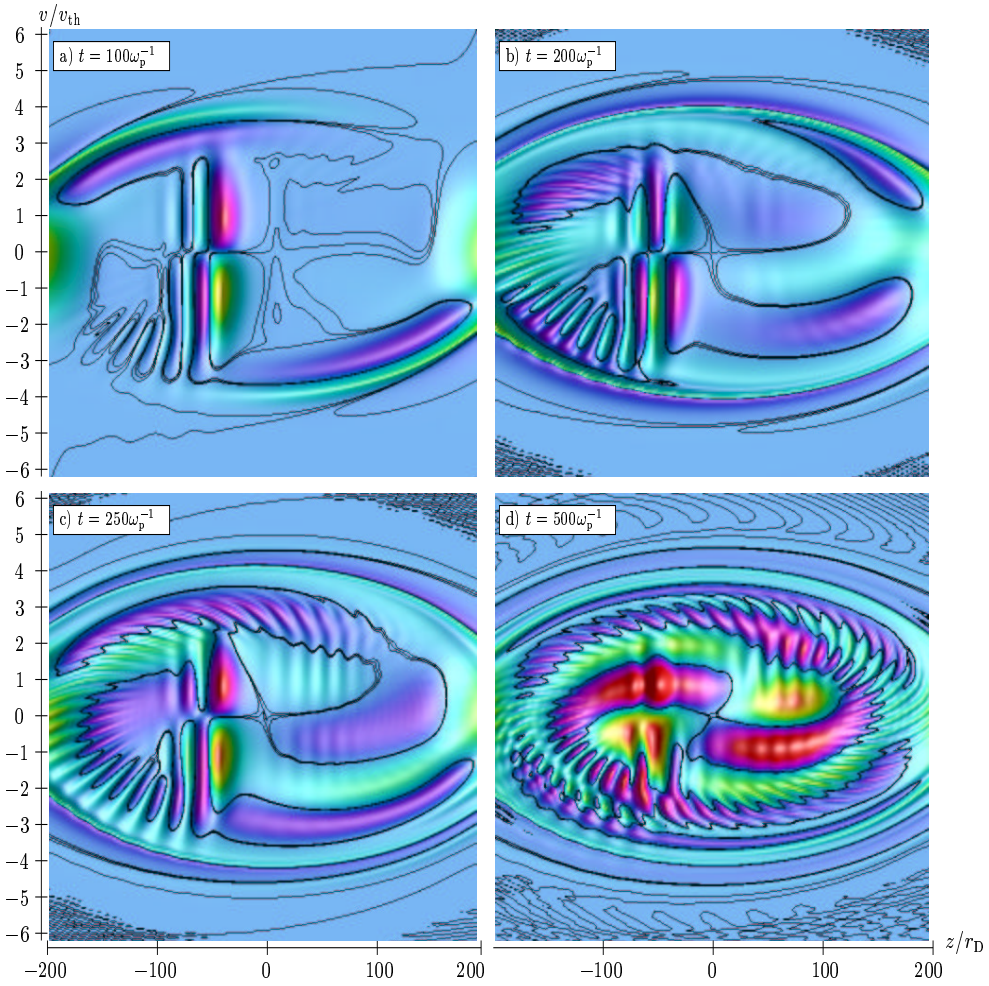


Figure 5.2: Kinetic tunnelling, $L = 80\, r_{\rm D}$.

with the accelerated particles having velocities between $v = -1.5\, v_{\rm th}$ and $v = -3.5\, v_{\rm th}$. These accelerated particles are thus free-streaming, ballistic particles, known as *van Kampen modes* [2], which do not contribute to the electric field in a correlated manner.

3. The free-streaming particles start to oscillate in the large-scale potential well created by the ion profile. They first move to the left and then turn and start moving to the right again, approximately following the elliptic trajectories described by Equation 5.17, in the (x, v) space. This can be seen in Figure

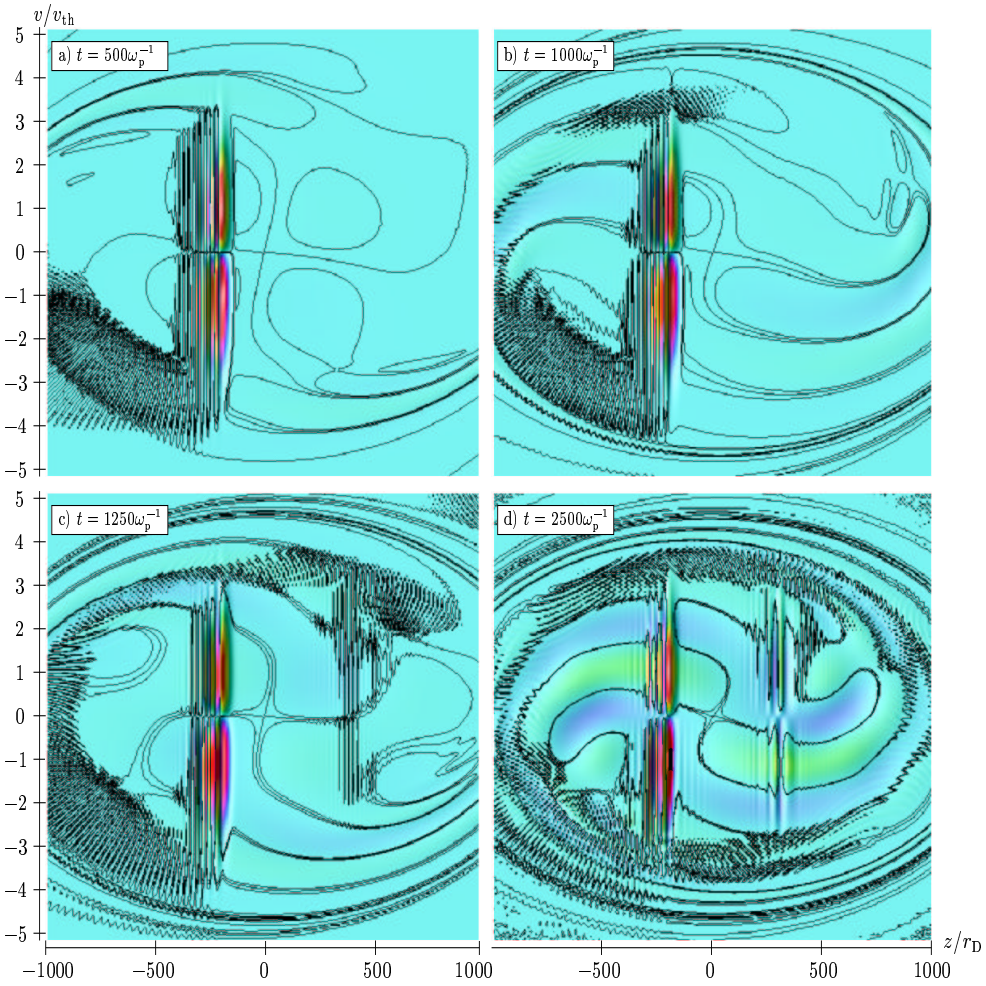
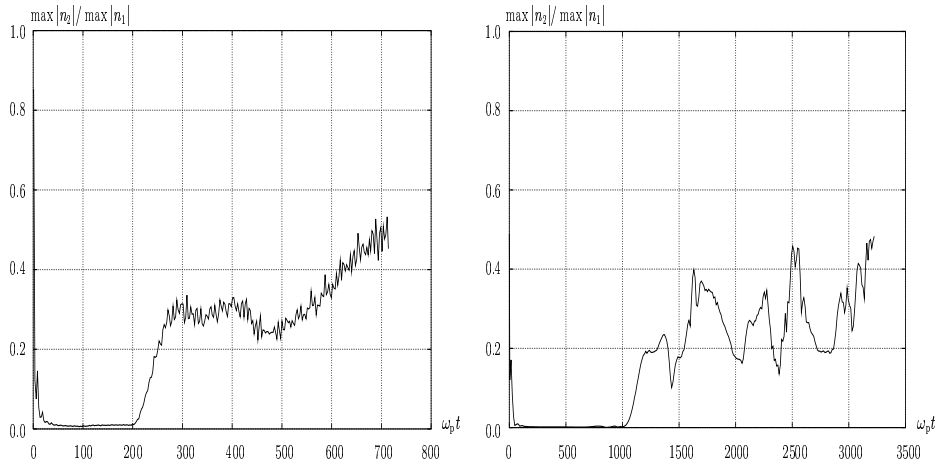


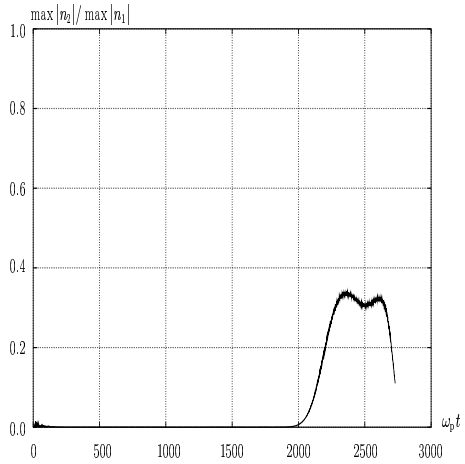
Figure 5.3: Kinetic tunnelling, $L = 400\,r_{\rm D}$.

5.2*b* where the accelerated particles turned and reached $z = -50\,r_{\rm D}$ having positive velocities around $v = +3\,v_{\rm th}$ to $v = +3.5\,v_{\rm th}$.

4. When the ballistic terms reach positive velocities, they gradually begin to line up again. In Figure 5.2*c* one can see the ballistic particles forming almost horizontal structures at $z = -100\,r_{\rm D}$ to $z = -200\,r_{\rm D}$ and with velocities around $v = 1\,v_{\rm th}$. Particles which have reached $z = 50\,r_{\rm D}$, having velocities around $+2\,v_{\rm th}$, form structures which are becoming more and more vertical. The vertical structures give a correlated contribution to the electric field



(a) $L = 80\,r_{\mathrm{D}}$ (b) $L = 400\,r_{\mathrm{D}}$



(c) $L = 800\,r_{\mathrm{D}}$

Figure 5.4: Tunnelling strength for different sizes of ion profiles

again, and one can see a macroscopic motion of particles around $z = 50\,r_{\mathrm{D}}$ to $z = 100\,r_{\mathrm{D}}$ having velocities $v = -2\,v_{\mathrm{th}}$ to $+2\,v_{\mathrm{th}}$. The corresponding collective motion for the larger profile $L = 400\,r_{\mathrm{D}}$ can be seen in Figure 5.3*c* and 5.3*d*, around $z = 250\,r_{\mathrm{D}}$ to $z = 500\,r_{\mathrm{D}}$. In Figure 5.3*d* one can also see the macroscopic, electrostatic waves again creating ballistic particles, this time

with positive velocities $v = 1 - 2\,v_{\mathrm{th}}$ at $x = 400 - 500\,r_{\mathrm{D}}$.

According to the physical interpretation above, the tunnelling effect should appear after approximately one half bouncing time,

$$T_{\mathrm{tunnelling}} = \frac{1}{2} T_{\mathrm{b}} \tag{5.35}$$

since the initial perturbation in the lower left quadrant of phase space, (z, v), will take a half bouncing time to reach the symmetric point in the upper right quadrant of phase space where the tunnelling effect appears, see Figure 5.2 and 5.3. The numerical experiment shows a good agreement between this prediction of the tunnelling time and the time obtained in the experiment. In figure 5.4 the tunnelling strength is measured as the maximum perturbation of the electron density on the right side of the peak at $z = 0$ of the profile, divided by the maximum perturbation of the density to the left of the peak. The tunnelling time is the time it takes for the perturbation, started at time $t = 0$ on the left side of the peak, to create collective perturbations on the right-hand side of the peak. This time can clearly be seen in Figure 5.4 for the three scale lengths $L = 80\,r_{\mathrm{D}}$, $L = 400\,r_{\mathrm{D}}$, and $L = 800\,r_{\mathrm{D}}$, as start of the first increase of the curve. The tunnelling time from the numerical experiment (Numerical) is compared with the theoretical prediction (Theory) in the following table:

L/r_{D}	$\omega_{\mathrm{p}} T_{\mathrm{tunnelling}} = \omega_{\mathrm{p}} \frac{1}{2} T_{\mathrm{b}}$ (Theory)	$\omega_{\mathrm{p}} T_{\mathrm{tunnelling}}$ (Numerical)
80	178	200
400	889	1000
800	1778	2000

As can be seen, the theoretical prediction and numerical result agrees up to about 90 percent. The numerical tunnelling time is slightly longer than the theoretical prediction, which is not surprising, since it takes some time for the electrostatic Langmuir waves to be created and to travel to the point where they are Landau damped, creating the ballistic particles.

One observation is that the tunnelling strength does not seem to decrease when the scale length is increased, see Figure 5.4; for all cases, the tunnelling strength has peaks around $0.3 - 0.4$.

The regeneration process described here is related to the so-called *plasma echo* effect [9], where an electrostatic wave which is Landau damped may be regenerated if the plasma is perturbed. This nonlinear effect has been observed in experiments.

5.5.3 Suggested experiments

In order to verify whether or not the observed regeneration of waves observed in experiment [7] may be explained by the kinetic tunnelling process investigated in this thesis, the following experiment is suggested: A similar ion modification experiment as described by Isham et al. should be repeated, and the following parameters should be measured:

1. The *time* between the switch on of the transmitter and the regeneration to appear should be measured. It was indicated theoretically and shown numerically that the regeneration of waves should appear after approximately one half bouncing time in the self-consistent potential well of the ionosphere. Electrons at a temperature 1500 K has the thermal velocity $v_{\text{th}} \approx 1.5 \times 10^5$ m/s. The scale length L of the ionosphere is of the order $10 - 100$ km, which gives the bounce time according to Equation (5.16) as $T_{\text{b}} \approx 0.3 - 3$ seconds. If the regeneration appears on this time scale, then the kinetic tunnelling could be an explanation. If the regeneration appears on a *much* faster time scale, then it indicates that some kind of tunnelling of electromagnetic waves must occur, and that it is unlikely that kinetic tunnelling can explain the experimental results. Revenchuk et al. [17, 18] propose an experiment with short pulses in order to measure the time for the regeneration to appear.

2. One should measure the *location* of the regeneration on the top side of the ionosphere. The electrons are likely to follow the magnetic filed lines of the geomagnetic field, since the mobility across the magnetic field lines is very low. If the regeneration is due to kinetic tunnelling of electrons, then it should be possible to predict where the regeneration appears on the top side by following the magnetic field lines from the location of the perturbation on the bottom side, to the top side. If, on the other hand, the regeneration is distributed over a large area of the top side, then it is more likely that some other tunnelling mechanism takes place, for example electromagnetic tunnelling, which is not constrained by the magnetic field lines.

Conclusions

A high-order method for solving numerically the Fourier transformed Vlasov-Poisson system in the velocity space has been developed, with special attention paid to the outflow boundary condition in the Fourier transformed space. It was shown numerically that it is possible to reduce the recurrence phenomenon by this method.

The boundary condition designed for the transformed system has been proved to be well-posed in the continuous case. The numerical scheme did not exhibit any instabilities in the numerical experiments.

As an application of the numerical method, the kinetic tunnelling of an ionospheric layer was studied. The theoretically predicted tunnelling effect appeared in the numerical simulation and did not decrease with larger length scales of the ion profile.

In these simulations, short range collisions between electrons and other particles have been neglected.

Physical experiments have been suggested: By measuring certain parameters, it should be possible to verify if kinetic tunnelling is the cause of the experimentally observed tunnelling of waves.

Acknowledgements

I want to thank Bertil Gustafsson at Department of Scientific Computing, Uppsala University, and Bo Thidé at Swedish Institute of Space Physics, Uppsala Division, for fruitful discussions and their useful advise during my work.

I also want to thank Sergey Revenchuk at Institute of Nuclear Research, National Academy of Sciences of Ukraine, Kiev, Ukraine, and Vladimir Pavlenko at Department of Astronomy and Space Physics, Uppsala, for interesting discussions on the physics of ionospheric layers.

Special thanks to professor Helmut Neunzert at University of Kaiserslauten, Germany, who kindly sent me published and unpublished material on the Boltzmann equation from his research.

This research was financially supported by the Swedish National Graduate School in Scientific Computing (NGSSC) and the Swedish Research Council (NFR).

Bibliography

[1] Thomas P. Armstrong, Rollin C. Harding, Georg Knorr, and David Montgomery. Solution of Vlasov's equation by transform methods. *Methods in Computational Physics (Academic Press)*, 9:29–86, 1970.

[2] Francis F. Chen. *Introduction to plasma physics and controlled fusion, second edition*, pages 261–267. Plenum Press, New York, 1984.

[3] C. Z. Cheng and G. Knorr. The integration of the Vlasov equation in configuration space. *Journal of Computational Physics.*, 22:330–351, 1976.

[4] J. Denavit and W. L. Kruer. Comparison of numerical solutions of the Vlasov equation with particle simulations of collisionless plasmas. *The Physics of Fluids*, pages 1782–1791, 1971.

[5] J. Feng and W. N. G. Hitchon. Self-consistent kinetic simulation of plasmas. *Physical Review E*, 61:3160–3173, 1999.

[6] Bertil Gustafsson and Pelle Olsson. Fourth-order difference methods for hyperbolic ibvps. *Journal of Computational Physics.*, 117:300–317, 1995.

[7] B. Isham, W. Kofman, T. Hagfors, J. Nordling, B. Thidé, C. LaHoz, and P. Stubbe. New phenomena observed by EISCAT during an RF ionospheric modification experiment. *Radio Sci.*, 25(3):251–262, 1990.

[8] Michael C. Kelley. *The Earth's ionosphere, plasma physics and electrodynamics*, pages 4–10. Academic Press, Inc., 1989.

[9] Nicholas A. Krall and Alvin W. Trivelpiece. *Principles of plasma physics*, pages 4–10. McGraw-Hill, Inc, 1973.

[10] Vipin Kumar, Ananth Grama, Anshul Gupta, and George Karypis. *Introduction to parallel computing: design and analysis of algorithms*, chapter 4. The Benjamin/Cummings Publishing Company, Inc, 1994.

[11] Sanjiva K. Lele. Compact finite difference schemes with spectral-like resolution. *Journal of Computational Physics.*, 103:16–42, 1992.

[12] Ching-Huei Lin, J. K. Chao, and C. Z. Cheng. One-dimensional Vlasov simulations of Langmuir solitons. *Phys. Plasmas 2*, pages 4195–4203, 1995.

[13] Giovanni Manfredi. Long-time behaviour of nonlinear Landau damping. *Phys. Rev. Lett.*, 79:2815–2818, 1997.

[14] E. Mishin, T. Hagfors, and B. Isham. A generation mehanism for topside enhanced incoherent backscatter during high frequency modification experiments in Tromsø. *Geophys. Res. Lett.*, pages 479–482, 2001.

[15] Helmut Neunzert. An introduction to the nonlinear Boltzmann-Vlasov equation. preprint no 28. Lectures given at the international summerschool "Kinetic Theories and Boltzmann Equations" of C.I.M.E. in Montecatini (Italy), June 1981.

[16] Helmut Neunzert. Verallgemeinerte Lösungen von Eigenwertproblemen, zugehörige Entwicklungsfragen und die Anwendung auf Gleichungen der Transporttheorie, Jül–816–MA, 1971.

[17] Sergey Revenchuk, Bo Thidé, and Vladimir Pavlenko. Wave tunneling in the ionosphere due to plasma kinetic effects: 1. Collisionless limit, (not published), 2000.

[18] Sergey Revenchuk, Bo Thidé, and Vladimir Pavlenko. Wave tunneling in the ionosphere due to plasma kinetic effects: 2. Influence of collisions, (not published), 2000.

[19] Henry Rishbeth and Owen K. Garriot. *Introduction to Ionospheric physics*, chapter 1. Academic Press, 1969.

[20] George Schmidt. *Physics of high temperature plasmas, second edition*, pages 269–282. Academic Press, Inc., 1979.

[21] John C. Strikwerda. *Finite difference schemes and partial differential equations*, chapter 1.6. Wadsworth, Inc., 1989.

[22] B. Thidé, H. Kopka, and P. Stubbe. *Observations of stimulated scattering of a strong High-Frequency radio wave in the ionosphere*, pages 1561–1564. Phys. Rev. Lett. 49, 1982.

APPENDIX A

Program listings, one-dimensional Vlasov code

A.1 vlasov.f90

```
!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! vlasov.f90    Bengt Eliasson    2000-03-18                    !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! This main program simulates the evolution of                  !
! electrostatic waves in a collisionless plasma,                !
! according to the Vlasov model. The problem is solved          !
! in the Fourier transformed velocity space, whith the          !
! variable 'eta' representing the fourier component of          !
! the velocity.                                                 !
! The program uses a physical object from the class             !
! 'OneDimVlasovPlasma', and some numerical objects              !
! that can operate on objects from this class.                  !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

PROGRAM vlasov

USE vlasov_numeric_mod

IMPLICIT NONE

  INTEGER :: i,j,k
  REAL(8) :: x,eta,omega,E2
  REAL(8) :: Norm,Norm0,NParticles,NParticles0
  REAL(8) :: P,P0,W,W0,Wp

  TYPE (OneDimVlasovPlasma) :: domain1


  ! Allocate memory for the domain
  CALL Constructor(domain1,Nx,Neta)

  ! Initialize the initial conditions and various constants, etc.
  CALL Domain_Initializer(domain1)
```

```
  IF (size.NE.NP) THEN
    WRITE(*,*)'size=',size,' NP=',NP,' are not equal.'
    STOP
  END IF

  ! Calculate various quantities from the initial condition
  CALL Domain_StatisticsCalculator(domain1,NParticles,P,W,Norm,Wp)

  ! Remember some of the initial values for the rest of the run
  NParticles0 = NParticles
  P0 = P
  W0 = W
  Norm0 = Norm

  ! Write results to screen and files
  k=0
  CALL Domain_StatisticsWriter(domain1,k,dt,W0,P0,Norm0,NParticles0,  &
       W,P,Norm,NParticles,Wp)

  DO k = 1,Nt

    ! Runge-Kutta step
    CALL RungeKutta_TimeStepper(domain1,dt,x1,x2,Echoice)

    IF (ABS(MOD(DBLE(k),DBLE(print1))) <= 1.0D-6) THEN
      ! Calculate various quantities from the initial condition
      CALL Domain_StatisticsCalculator(domain1,NParticles,P,W,Norm,Wp)

      ! Write results to screen and files
      CALL Domain_StatisticsWriter(domain1,k,dt,W0,P0,Norm0,NParticles0,  &
       W,P,Norm,NParticles,Wp)
    END IF
  END DO

  ! Clean up
  CALL Domain_Finalizer(domain1)

  ! Deallocate memory
  CALL Destructor(domain1)

  STOP 0

END PROGRAM
!
```

A.2 vlasov_numeric_mod.f90

```
!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! vlasov_numeric_mod.f90   Bengt Eliasson    2001-03-18     !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! This module contains numerical objects that can act on    !
```

```
! physical objects from the class OneDimVlasovPlasma, or on !
! parts of the object.                                       !
!                                                           !
! The module contains the following numerical objects:      !
!                                                           !
! Domain_Initializer - Calculates the initial condition for !
!   the problem, calculates various constants, opens data   !
!   files, etc.                                             !
!                                                           !
! Domain_Finalizer - Closes data files, etc.                !
!                                                           !
! Domain_StatisticsWriter - Writes various statistics of    !
!   the solution to file and screen.                        !
!                                                           !
! Domain_StatisticsCalculator - Calculates statistics from  !
!   the solution.                                           !
!                                                           !
! PDE_RHS - Calculates the righthand side of the time-      !
!   dependent partial differential eqation.                 !
!                                                           !
! ElectricField - Calculates the electric field by          !
!   integrating the Poisson equation (Maxwell's equations), !
!   using a spectral method.                                !
!                                                           !
! x_FourierDifferentiator - Calculates d/dx of a            !
!   two-dimensional function f(x,eta), using the a spectral !
!   method.                                                 !
!                                                           !
! TriDiag_Solver - Solves a linear tri-diagonal equation    !
!   system.                                                 !
!                                                           !
! eta_Differetiator - Calculates d/deta of a two-           !
!   dimensional function f(x,eta), using the Pade'          !
!   approximation.                                          !
!                                                           !
! SimpsonIntegrator - integrates a function f(x),           !
!   using the Simpson formula.                              !
!                                                           !
! PeriodicIntegrator - integrates a function f(x),          !
!   where the function is periodic in x.                    !
!                                                           !
! RungeKutta_TimeStepper - Advances the solution one time   !
!   step by using the fouth order Runge-Kutta method.       !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!


module vlasov_numeric_mod
  USE vlasov_domain_mod

  IMPLICIT NONE

contains

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Initialize the initial condition, various constants, etc.
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
subroutine Domain_Initializer(domain)
  IMPLICIT NONE
  TYPE (OneDimVlasovPlasma) :: domain

  INTEGER :: i,j,iostat,info,initial
  REAL(8) :: density(Nx),x,eta,w
  CHARACTER(len=2) :: procname

  ! Initialize MPI
  CALL MPI_INIT(errcode)
  CALL MPI_COMM_SIZE(MPI_COMM_WORLD,size,errcode)
  CALL MPI_COMM_RANK(MPI_COMM_WORLD,rank,errcode)

  ! Assign 'procname' to be used for creating file names
  SELECT CASE(rank)
  CASE(0)
    procname='00'
  CASE(1)
    procname='01'
  CASE(2)
    procname='02'
  CASE(3)
    procname='03'
  CASE(4)
    procname='04'
  CASE(5)
    procname='05'
  CASE(6)
    procname='06'
  CASE(7)
    procname='07'
  CASE(8)
    procname='08'
  CASE(9)
    procname='09'
  CASE(10)
    procname='10'
  CASE(11)
    procname='11'
  CASE(12)
    procname='12'
  CASE(13)
    procname='13'
  CASE(14)
    procname='14'
  CASE(15)
    procname='15'
  CASE(16)
    procname='16'
  CASE(17)
    procname='17'
  CASE(18)
    procname='18'
  CASE(19)
    procname='19'
  CASE(20)
```

```
  progname='20'
CASE(21)
  progname='21'
CASE(22)
  progname='22'
CASE(23)
  progname='23'
CASE(24)
  progname='24'
CASE(25)
  progname='25'
CASE(26)
  progname='26'
CASE(27)
  progname='27'
CASE(28)
  progname='28'
CASE(29)
  progname='29'
CASE(30)
  progname='30'
CASE(31)
  progname='31'
END SELECT


! The grid size in x direction
domain%dx = (x2-x1)/DBLE(TotalNx)

! The grid size in eta direction
domain%deta = eta_max/Neta

! Put arrays to zero
domain%fre(:,:) = zero
domain%fim(:,:) = zero
domain%E(:) = zero


! Initial values
initial=0
IF (initial.EQ.0) THEN
DO i = 1,Nx
  x = DBLE(rank*Nx+i-1)*domain%dx
  DO j = 0,Neta
    eta = DBLE(j)*domain%deta
    domain%fre(j+1,i) = exp(-eta*eta*half)*(one+A*COS(kx*x))/twopi
  END DO
END DO
ELSEIF (initial.EQ.1) THEN
  ! The propagating wave initial condition.
  w=SQRT(one+three*kx*kx)
  DO i = 1,Nx
    x = DBLE(rank*Nx+i-1)*domain%dx
    DO j = 0,Neta
      eta = DBLE(j)*domain%deta
      domain%fre(j+1,i) = cos(w/kx*A*eta*cos(kx*x))*  &
```

```
  exp(-eta*eta*half)*(one+A*COS(kx*x))/twopi
        domain%fim(j+1,i) = sin(w/kx*A*eta*cos(kx*x))*  &
  exp(-eta*eta*half)*(one+A*COS(kx*x))/twopi
      END DO
    END DO
  ENDIF


  ! Calculate E
  CALL ElectricField(domain%fre,domain%E,domain%E123,   &
    domain%dx,Ethoice,x1,x2,Nx,Neta)


  ! Open files
 IF (WriteToFile) THEN
  OPEN (UNIT=outE_1,FILE='E_1.dat',ACCESS='SEQUENTIAL',   &
    FORM='FORMATTED',ACTION='WRITE',IOSTAT=iocheck)

  OPEN (UNIT=outE_2,FILE='E_2.dat',ACCESS='SEQUENTIAL',   &
    FORM='FORMATTED',ACTION='WRITE',IOSTAT=iocheck)

  OPEN (UNIT=outE_3,FILE='E_3.dat',ACCESS='SEQUENTIAL',   &
    FORM='FORMATTED',ACTION='WRITE',IOSTAT=iOcheck)

  OPEN (UNIT=outdP,FILE='P.dat',ACCESS='SEQUENTIAL',   &
    FORM='FORMATTED',ACTION='WRITE',IOSTAT=iocheck)

  OPEN (UNIT=outdW,FILE='dW.dat',ACCESS='SEQUENTIAL',   &
    FORM='FORMATTED',ACTION='WRITE',IOSTAT=iocheck)

  OPEN (UNIT=outCorr,FILE='etaCorr.dat',ACCESS='SEQUENTIAL',   &
    FORM='FORMATTED',ACTION='WRITE',IOSTAT=iocheck)

  OPEN (UNIT=outFourier,FILE='Fourier.dat',ACCESS='SEQUENTIAL',  &
    FORM='FORMATTED',ACTION='WRITE',IOSTAT=iocheck)

  OPEN (UNIT=outE2,FILE='E2.dat',ACCESS='SEQUENTIAL',  &
    FORM='FORMATTED',ACTION='WRITE',IOSTAT=iocheck)

  OPEN (UNIT=outNorm,FILE='Norm.dat',ACCESS='SEQUENTIAL',  &
    FORM='FORMATTED',ACTION='WRITE',IOSTAT=iocheck)

  OPEN (UNIT=outfabs,FILE='fabs.'//proname//'.dat',  &
    ACCESS='SEQUENTIAL',  &
    FORM='FORMATTED',ACTION='WRITE',IOSTAT=iocheck)

  OPEN (UNIT=outF,FILE='F.'//proname//'.dat',   &
    ACCESS='SEQUENTIAL',  &
    FORM='FORMATTED',ACTION='WRITE',IOSTAT=iocheck)
 END IF

end subroutine Domain_Initializer

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!  Close files etc.
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
subroutine Domain_Finalizer(domain)
  TYPE (OneDimVlasovPlasma) :: domain
  INTEGER :: errcode

  ! Close data files
  IF (WriteToFile) THEN
    CLOSE(outE_1)
    CLOSE(outE_2)
    CLOSE(outE_3)
    CLOSE(outdP)
    CLOSE(outdW)
    CLOSE(outCorr)
    CLOSE(outFourier)
    CLOSE(outE2)
    CLOSE(outNorm)
    CLOSE(outfabs)
    CLOSE(outF)
  END IF

  ! Finalize MPI
  CALL MPI_FINALIZE(errcode)
end subroutine Domain_Finalizer

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Write results to screen and to data files
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
subroutine Domain_StatisticsWriter(domain,k,dt,W0,P0,  &
     Norm0,NParticles0,W,P,Norm,NParticles,Wp)
  IMPLICIT NONE
  TYPE (OneDimVlasovPlasma) :: domain
  INTEGER :: k
  REAL(8) :: dt,W0,P0,Norm0,NParticles0,W,P,Norm,NParticles,Wp

  REAL(8) :: Nx_temp(Nx),Neta_temp(Neta+1)

  INTEGER :: i,j
  INTEGER(4) :: SqrTwoPi

  ! FFTPACK
  INTEGER(4) :: L
  REAL(8) :: wsave(8*Neta+15),scaling2
  COMPLEX(8) :: c(2*Neta)

    IF (WriteToFile) THEN

      IF (rank.EQ.0) THEN
        WRITE (outNorm,906) k*dt," ",(Norm-Norm0)/Norm0
        WRITE (outE2,906) k*dt," ",DSQRT(two*Wp)
        WRITE (outdP,906) k*dt," ",P-P0
        WRITE (outdW,906) k*dt," ",(W-W0)/W0
        WRITE (outE_1,906) k*dt," ",domain%E123(1)
        WRITE (outE_2,906) k*dt," ",domain%E123(2)
        WRITE (outE_3,906) k*dt," ",domain%E123(3)
      END IF

!      DO i = 1,Nx
```

```
!        DO j = 0,Neta
!          WRITE (outfabs,907) DSQRT(domain%fre(j+1,i)*domain%fre(j+1,i)+ &
!   domain%fim(j+1,i)*domain%fim(j+1,i))
!        END DO
!      END DO

    IF (0.EQ.1) THEN
      L=2*Neta

      CALL dzffti(L,wsave)

      SqrTwoPi=SQRT(twopi)
      DO i = 1,Nx
        DO j = 1,Neta
          z(j)=DCMPLX(domain%fre(j,i)*  &
            Eta_max/DBLE(Neta)*SqrTwoPi,  &
            domain%fim(j,i)*Eta_max/DBLE(Neta)*SqrTwoPi)

          z(2*Neta+1-j)=DCMPLX(domain%fre(j+1,i)*  &
            Eta_max/DBLE(Neta)*SqrTwoPi,  &
            -domain%fim(j+1,i)*Eta_max/DBLE(Neta)*SqrTwoPi)
        END DO

        CALL dzfftf(L,z,wsave)

        DO j = 1,Neta
          WRITE (outF,*) REAL(z(Neta+j))
        END DO
        DO j = 1,Neta
          WRITE (outF,*) REAL(z(j))
        END DO
      END DO

    END IF


   END IF


   IF (rank.EQ.0) THEN
      WRITE (*,905) k," dNorm: ",(Norm-Norm0)/Norm0,                &
        " dN: ",(NParticles-NParticles0)/NParticles0," dP: ",P-P0,  &
        " dW: ",(W-W0)/W0," Wp: ",Wp/W0
   END IF

901 FORMAT (F15.10,A)
902 FORMAT (99A)
903 FORMAT (F15.10)
904 FORMAT (F15.10,A)
905 FORMAT (I7,A,F15.10,A,F15.10,A,F15.13,A,F15.10,A,F15.10)
906 FORMAT (F15.10,A,F15.10)
907 FORMAT (F15.10)

End subroutine Domain_StatisticsWriter
```

```
subroutine Domain_StatisticsCalculator(domain,NParticles,P,W,Norm,Wp)
  IMPLICIT NONE
  TYPE (OneDimVlasovPlasma) :: domain
  REAL(8) :: NParticles,P,W,Norm,Wp
  REAL(8) :: Nx_temp(Nx),Neta_temp(Neta+1)
  INTEGER :: i,j

  ! MPI
  INTEGER :: root=0

      NParticles = zero
      Norm = zero

      ! The number of particles
      DO i = 1,Nx
        Nx_temp(i)=twopi*domain%fre(1,i)
      END DO
      CALL PeriodicIntegrator(Nx_temp,domain%dx,Nx,NParticles)

      ! Sum all partial sums to proc 0.
      CALL MPI_REDUCE(NParticles,NParticles,1,  &
        MPI_DOUBLE_PRECISION,MPI_SUM,root,MPI_COMM_WORLD,errcode)

      ! The momentum
      DO i = 1,Nx
        Nx_temp(i)=twopi*(45.0d0*domain%fim(2,i)-9.0d0*domain%fim(3,i)  &
          +domain%fim(4,i))/(30.0d0*domain%deta)
      END DO
      CALL PeriodicIntegrator(Nx_temp,domain%dx,Nx,P)

      ! Sum all partial sums to proc 0.
      CALL MPI_REDUCE(P,P,1,MPI_DOUBLE_PRECISION,  &
        MPI_SUM,root,MPI_COMM_WORLD,errcode)


      ! The potential energy
      DO i = 1,Nx
        Nx_temp(i)=domain%E(i)*domain%E(i)*half
      END DO
      CALL PeriodicIntegrator(Nx_temp,domain%dx,Nx,Wp)

      ! Sum all partial sums to proc 0.
      CALL MPI_REDUCE(Wp,Wp,1,MPI_DOUBLE_PRECISION,  &
        MPI_SUM,root,MPI_COMM_WORLD,errcode)

      ! The kinetic energy
      DO i = 1,Nx
        Nx_temp(i)=-pi*(-245.0d0*domain%fre(1,i)+270.0d0*domain%fre(2,i)   &
         -27.0d0*domain%fre(3,i)+2.0d0*domain%fre(4,i))/  &
          (90.0d0*domain%deta*domain%deta)
      END DO

      DO i = 1,Nx
        Nx_temp(i)=Nx_temp(i)+domain%E(i)*domain%E(i)*half
      END DO
```

```
      CALL PeriodicIntegrator(Nx_temp,domain%dx,Nx,W)

      ! Sum all partial sums to proc 0.
      CALL MPI_REDUCE(W,W,1,MPI_DOUBLE_PRECISION,  &
        MPI_SUM,root,MPI_COMM_WORLD,errcode)

      ! The entropy function
      DO j = 1,Neta+1
        DO i = 1,Nx
          Nx_temp(i) = domain%fre(j,i)*domain%fre(j,i)  &
             +domain%fim(j,i)*domain%fim(j,i)
        END DO
        CALL PeriodicIntegrator(Nx_temp,domain%dx,Nx,Norm)
        Neta_temp(j)=Norm
      END DO

      ! Sum all partial sums to proc 0.
      CALL MPI_REDUCE(Neta_temp,Neta_temp,Neta+1,  &
        MPI_DOUBLE_PRECISION,MPI_SUM,root,MPI_COMM_WORLD,errcode)

      CALL SimpsonIntegrator(Neta_temp,domain%deta,Neta,Norm)

end subroutine Domain_StatisticsCalculator

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Calculate the righthand side P of the differential
! equation df/dt=P
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
subroutine PDE_RHS (domain,fre,fim,Pre,Pim,E,dx,deta,x1,x2,Echoice)
  IMPLICIT NONE

  ! Arguments
  TYPE (OneDimVlasovPlasma) :: domain
  REAL(8),DIMENSION(:,:) :: fre,fim,Pre,Pim
  REAL(8) :: E(:),dx,deta,x1,x2
  INTEGER :: Echoice

  REAL(8) :: density(Nx)
  REAL(8) :: d_deta_re(Neta+1,Nx),d_deta_im(Neta+1,Nx)
  REAL(8) :: d_dxdeta_re(Neta+1,Nx),d_dxdeta_im(Neta+1,Nx)
  REAL(8) :: temp(2),eta(1:Neta+1),E123(1:3),frac_1_dx2
  INTEGER :: i,j,N

  ! Fourier transform
  REAL(8) :: FreNx(TotalNx),FimNx(TotalNx)

  ! MPI
  INTEGER :: root=0,status(MPI_STATUS_SIZE),ierror,request
  INTEGER :: right,left,sendtag,recvtag
  REAL(8) :: NxTemp(Nx),sendbuf(1:Neta+1,3)
  REAL(8) :: rcre(1:Neta+1,3),lcre(1:Neta+1,3),rcim(1:Neta+1,3),lcim(1:Neta+1,3)

  ! FFTPACK
  INTEGER :: L
  REAL(8) :: wsave(4*TotalNx+15),scaling
  COMPLEX(8) :: c(TotalNx)
```

```
    ! Calculate E
    CALL ElectricField(fre,E,E123,dx,Ethoice,x1,x2,Nx,Neta)

    ! Calculate d/deta
    CALL eta_Differentiator(domain,fre,d_deta_re,deta,'R',Nx,Neta)
    CALL eta_Differentiator(domain,fim,d_deta_im,deta,'I',Nx,Neta)

    ! Calculate the x derivative
    CALL x_FourierDifferentiator(domain,d_deta_re,d_deta_im,  &
      d_dxdeta_re,d_dxdeta_im,dx,Nx,Neta)

    ! Create eta
    DO j = 1,Neta+1
      eta(j) = DBLE(j-1)*deta
    END DO

    ! Calculate the righthand side
    DO i = 1,Nx
      DO j = 1,Neta+1
        Pre(j,i) = -d_dxdeta_im(j,i)+eta(j)*E(i)*fim(j,i)
      END DO
    END DO

    DO i = 1,Nx
      DO j = 1,Neta+1
        Pim(j,i) =  d_dxdeta_re(j,i)-eta(j)*E(i)*fre(j,i)
      END DO
    END DO

!!!!! Boundary eta=eta_max !!!!!!!!!!!!!!!!!!

    ! Proc 0 calculates the boundary condition. All procs sends data to proc 0.
    DO i=1,Nx
      NxTemp(i)=Pre(Neta+1,i)
    END DO

    CALL MPI_GATHER(NxTemp,Nx,MPI_DOUBLE_PRECISION,  &
      FreNx,Nx,MPI_DOUBLE_PRECISION,  &
      root,MPI_COMM_WORLD,errcode)

    DO i=1,Nx
      NxTemp(i)=Pim(Neta+1,i)
    END DO

    CALL MPI_GATHER(NxTemp,Nx,MPI_DOUBLE_PRECISION,  &
      FimNx,Nx,MPI_DOUBLE_PRECISION,  &
      root,MPI_COMM_WORLD,errcode)

    IF (rank.EQ.root) THEN
      L=TotalNx

      DO i=1,TotalNx
        c(i)=DCMPLX(FreNx(i),FimNx(i))
      END DO
```

```
      ! Initialize the FFT routine
      CALL dzffti(L,wsave)

      ! Perform FFTon the boundary
      CALL dzfftf(L,z,wsave)

      ! Remove negative k_x.
      DO i=1+TotalNx/2,TotalNx
        z(i)=DCMPLX(zero,zero)
      END DO

      CALL dzfftb(L,z,wsave)

      scaling=one/DBLE(TotalNx)

      DO i=1,TotalNx
        FreNx(i)=DREAL(z(i))*scaling
      END DO

      DO i=1,TotalNx
        FimNx(i)=DIMAG(z(i))*scaling
      END DO
    END IF


    CALL MPI_SCATTER(FreNx,Nx,MPI_DOUBLE_PRECISION,  &
      NxTemp,Nx,MPI_DOUBLE_PRECISION,  &
      root,MPI_COMM_WORLD,errcode)

    DO i=1,Nx
      Pre(Neta+1,i)=NxTemp(i)
    END DO

    CALL MPI_SCATTER(FimNx,Nx,MPI_DOUBLE_PRECISION,  &
      NxTemp,Nx,MPI_DOUBLE_PRECISION,  &
      root,MPI_COMM_WORLD,errcode)

    DO i=1,Nx
      Pim(Neta+1,i)=NxTemp(i)
    END DO
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

    frac_1_dx2=one/(dx*dx)

    ! Add a 6:th order dissipative term in x direction
    DO i=4,Nx-3
      DO j=1, Neta+1
        Pre(j,i)=Pre(j,i)+diss*(fre(j,i+3)-six*fre(j,i+2)    &
          +15.0d0*fre(j,i+1)-20.0d0*fre(j,i)+15.0d0*fre(j,i-1)  &
         -six*fre(j,i-2)+fre(j,i-3))*frac_1_dx2
      END DO
    END DO

    DO i=4,Nx-3
      DO j=1, Neta+1
```

```
    Pim(j,i)=Pim(j,i)+diss*(fim(j,i+3)-six*fim(j,i+2)    &
      +15.0d0*fim(j,i+1)-20.0d0*fim(j,i)+15.0d0*fim(j,i-1)  &
      -six*fim(j,i-2)+fim(j,i-3))*frac_1_dx2
  END DO
END DO

! Exchange the boundary cells with neighbouring processors
IF (rank.EQ.0) THEN
  left=size-1
ELSE
  left=rank-1
END IF

IF (rank.EQ.size-1) THEN
  right=0
ELSE
  right=rank+1
END IF

sendbuf(:,1:3)=fre(:,Nx-2:Nx)
sendtag=100+rank
recvtag=100+left
CALL MPI_ISEND(sendbuf,3*(Neta+1),MPI_DOUBLE_PRECISION,  &
  right,sendtag,MPI_COMM_WORLD,request,ierror)
CALL MPI_WAIT(request,status,ierror)
CALL MPI_IRECV(lore,3*(Neta+1),MPI_DOUBLE_PRECISION,  &
  left,recvtag,MPI_COMM_WORLD,request,ierror)
CALL MPI_WAIT(request,status,ierror)

sendbuf(:,1:3)=fim(:,Nx-2:Nx)
sendtag=100+rank
recvtag=100+left
CALL MPI_ISEND(sendbuf,3*(Neta+1),MPI_DOUBLE_PRECISION,  &
  right,sendtag,MPI_COMM_WORLD,request,ierror)
CALL MPI_WAIT(request,status,ierror)
CALL MPI_IRECV(loim,3*(Neta+1),MPI_DOUBLE_PRECISION,  &
  left,recvtag,MPI_COMM_WORLD,request,ierror)
CALL MPI_WAIT(request,status,ierror)

sendbuf(:,1:3)=fre(:,1:3)
sendtag=100+rank
recvtag=100+right
CALL MPI_ISEND(sendbuf,3*(Neta+1),MPI_DOUBLE_PRECISION,  &
  left,sendtag,MPI_COMM_WORLD,request,ierror)
CALL MPI_WAIT(request,status,ierror)
CALL MPI_IRECV(rore,3*(Neta+1),MPI_DOUBLE_PRECISION,  &
  right,recvtag,MPI_COMM_WORLD,request,ierror)
CALL MPI_WAIT(request,status,ierror)

sendbuf(:,1:3)=fim(:,1:3)
sendtag=100+rank
recvtag=100+right
CALL MPI_ISEND(sendbuf,3*(Neta+1),MPI_DOUBLE_PRECISION,  &
  left,sendtag,MPI_COMM_WORLD,request,ierror)
CALL MPI_WAIT(request,status,ierror)
CALL MPI_IRECV(roim,3*(Neta+1),MPI_DOUBLE_PRECISION,  &
```

```
      right,recvtag,MPI_COMM_WORLD,request,ierror)
    CALL MPI_WAIT(request,status,ierror)


    DO j=1,Neta+1
      Pre(j,1)=Pre(j,1)+diss*(fre(j,4)-six*fre(j,3)    &
        +15.0d0*fre(j,2)-20.0d0*fre(j,1)+15.0d0*lore(j,3) &
        -six*lore(j,2)+lore(j,1))*frac_1_dx2
      Pim(j,1)=Pim(j,1)+diss*(fim(j,4)-six*fim(j,3)    &
        +15.0d0*fim(j,2)-20.0d0*fim(j,1)+15.0d0*loim(j,3) &
        -six*loim(j,2)+loim(j,1))*frac_1_dx2
      Pre(j,2)=Pre(j,2)+diss*(fre(j,5)-six*fre(j,4)    &
        +15.0d0*fre(j,3)-20.0d0*fre(j,2)+15.0d0*fre(j,1)  &
        -six*lore(j,3)+lore(j,2))*frac_1_dx2
      Pim(j,2)=Pim(j,2)+diss*(fim(j,5)-six*fim(j,4)    &
        +15.0d0*fim(j,3)-20.0d0*fim(j,2)+15.0d0*fim(j,1)  &
        -six*loim(j,3)+loim(j,2))*frac_1_dx2
      Pre(j,3)=Pre(j,3)+diss*(fre(j,6)-six*fre(j,5)    &
        +15.0d0*fre(j,4)-20.0d0*fre(j,3)+15.0d0*fre(j,2)  &
        -six*fre(j,1)+lore(j,3))*frac_1_dx2
      Pim(j,3)=Pim(j,3)+diss*(fim(j,6)-six*fim(j,5)    &
        +15.0d0*fim(j,4)-20.0d0*fim(j,3)+15.0d0*fim(j,2)  &
        -six*fim(j,1)+loim(j,3))*frac_1_dx2

      Pre(j,Nx-2)=Pre(j,Nx-2)+diss*(rore(j,1)-six*fre(j,Nx)  &
        +15.0d0*fre(j,Nx-1)-20.0d0*fre(j,Nx-2)+15.0d0*fre(j,Nx-3)  &
        -six*fre(j,Nx-4)+fre(j,Nx-5))*frac_1_dx2
      Pim(j,Nx-2)=Pim(j,Nx-2)+diss*(roim(j,1)-six*fim(j,Nx)  &
        +15.0d0*fim(j,Nx-1)-20.0d0*fim(j,Nx-2)+15.0d0*fim(j,Nx-3)  &
        -six*fim(j,Nx-4)+fim(j,Nx-5))*frac_1_dx2
      Pre(j,Nx-1)=Pre(j,Nx-1)+diss*(rore(j,2)-six*rore(j,1)  &
        +15.0d0*fre(j,Nx)-20.0d0*fre(j,Nx-1)+15.0d0*fre(j,Nx-2)  &
        -six*fre(j,Nx-3)+fre(j,Nx-4))*frac_1_dx2
      Pim(j,Nx-1)=Pim(j,Nx-1)+diss*(roim(j,2)-six*roim(j,1)  &
        +15.0d0*fim(j,Nx)-20.0d0*fim(j,Nx-1)+15.0d0*fim(j,Nx-2)  &
        -six*fim(j,Nx-3)+fim(j,Nx-4))*frac_1_dx2
      Pre(j,Nx)=Pre(j,Nx)+diss*(rore(j,3)-six*rore(j,2)         &
        +15.0d0*rore(j,1)-20.0d0*fre(j,Nx)+15.0d0*fre(j,Nx-1)  &
        -six*fre(j,Nx-2)+fre(j,Nx-3))*frac_1_dx2
      Pim(j,Nx)=Pim(j,Nx)+diss*(roim(j,3)-six*roim(j,2)         &
        +15.0d0*roim(j,1)-20.0d0*fim(j,Nx)+15.0d0*fim(j,Nx-1)  &
        -six*fim(j,Nx-2)+fim(j,Nx-3))*frac_1_dx2
    END DO

end subroutine PDE_RHS

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! The ElectricField object calculates the electric field
! from the electron particle density.
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
subroutine ElectricField(fre,E,E123,dx,Echoice,x1,x2,Nx,Neta)
  IMPLICIT NONE

  ! Arguments
  REAL(8) :: fre(:,:),E(:),E123(1:3)
  REAL(8) :: dx,x1,x2
```

```
INTEGER :: Ethoice,Nx,Neta

REAL(8) :: Edelta,U,frac_dx_24
INTEGER :: i

! Fourier transform
REAL(8) :: FreNx(TotalNx),FimNx(TotalNx)

! FFTPACK
REAL(8) :: wsave(4*TotalNx+15),scaling2
COMPLEX(8) :: c(TotalNx)
INTEGER :: L

REAL(8) :: NxTemp(Nx)

! MPI
INTEGER :: root=0


  IF (Ethoice.EQ.0) THEN

    frac_dx_24=dx*0.041666666666666666666d0


    ! Use trigonometric interpolation

    ! Proc 0 calculates the electric field. All procs sends data to
    ! proc 0 for calculating the electric field.
    DO i=1,Nx
      NxTemp(i)=one-twopi*fre(1,i)
    END DO


    CALL MPI_GATHER(NxTemp,Nx,MPI_DOUBLE_PRECISION,  &
      FreNx,Nx,MPI_DOUBLE_PRECISION,  &
      root,MPI_COMM_WORLD,errcode)

    ! Calculate E-field if proc 0.
    IF (rank.EQ.0) THEN

      L=TotalNx

      ! Initialize the FFT routine
      CALL dffti(L,wsave)

      scaling2=twopi/(dx)


      DO i=1,TotalNx
        FimNx(i)=zero
        c(i)=DCMPLX(FreNx(i),FimNx(i))
      END DO

      CALL dfftf(L,c,wsave)

      ! Positive k_x.
```

```
        ! No zero frequency component.
        z(1)=DCMPLX(zero,zero)

        ! The non-zero frequency components.
        DO i=2,TotalNx/2
          z(i)=z(i)*dcmplx(zero,-one)/DBLE(i-1)
        END DO

        ! Negative k_x.
        DO i=1,TotalNx/2
          z(TotalNx+1-i)=z(TotalNx+1-i)*dcmplx(zero,one)/DBLE(i)
        END DO
        z(TotalNx/2+1)=zero

        ! Return some components, for statistics ...
        E123(1)=ABS(z(2))
        E123(2)=ABS(z(3))
        E123(3)=ABS(z(4))

        CALL dfftb(L,z,wsave)

        DO i=1,TotalNx
          FreNx(i)=DREAL(z(i))/scaling2
        END DO

      END IF

      CALL MPI_SCATTER(FreNx,Nx,MPI_DOUBLE_PRECISION,  &
        E,Nx,MPI_DOUBLE_PRECISION,  &
        root,MPI_COMM_WORLD,errcode)

    ELSE

      DO i = 1,Nx
        E(i) = zero*COS(pi*DBLE(rank*Nx+i-1)/DBLE(TotalNx))
      END DO

  END IF
end subroutine ElectricField


!


!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! The x_FourierDifferentiator calculates d/dx of a two-
! dimensional function f(x,eta), using the fourth
! order difference approximation.
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
subroutine x_FourierDifferentiator(domain,  &
    vinRe,vinIm,voutRe,voutIm,dx,Nx,Neta)
  ! Calculate d/dx of 'vin', the result returned in 'vout'
  IMPLICIT NONE

  ! Arguments
  TYPE (OneDimVlasovPlasma) :: domain
  REAL(8),DIMENSION(:,:) :: vinRe,vinIm,voutRe,voutIm
```

```
REAL(8) :: dx
INTEGER :: Nx,Neta

INTEGER :: i,j

!MPI
INTEGER :: root

! FFTPACK
REAL(8) :: wsave(4*TotalNx+15),scaling2
COMPLEX(8) :: z(TotalNx),NxTempZ(Nx)
INTEGER :: L

L=TotalNx

! Initialize the FFT routine
CALL dzffti(L,wsave)

scaling2=twopi/(dx*DBLE(TotalNx*TotalNx))

DO j=1,Neta+1,size
    DO root=0,size-1
      IF (j+root.LE.Neta+1) THEN

        ! Proc 'root' calculates the x derivative. All procs sends data to
        ! proc 0.
        DO i=1,Nx
          NxTempZ(i)=DCMPLX(vinRe(j+root,i),vinIm(j+root,i))
        END DO

        CALL MPI_GATHER(NxTempZ,Nx,MPI_DOUBLE_COMPLEX,  &
          z,Nx,MPI_DOUBLE_COMPLEX,  &
          root,MPI_COMM_WORLD,errcode)

      END IF
  END DO

  ! All procs calculate the x derivative in parallel.
  IF (j+rank.LE.Neta+1) THEN

    ! Perform forward Fourier transform.
    CALL dzfftf(L,z,wsave)

    ! Multiply by ik_x, positive k_x.
    DO i=1,TotalNx/2
      z(i)=z(i)*DBLE(i-1)*DCMPLX(zero,one)
    END DO

    ! Multiply by ik_x, negative k_x.
    DO i=1,TotalNx/2
      z(TotalNx+1-i)=z(TotalNx+1-i)*DBLE(i)*DCMPLX(zero,-one)
    END DO

    ! The Fourier component corresponding to the highest frequency is
    ! put to zero - it belongs both to positive and negative k_x.
    z(TotalNx/2+1)=DCMPLX(zero,zero)
```

```
      ! Perform backward Fourier transform.
      CALL dfftb(L,c,wsave)
    END IF

    ! Re-distribute the result of the calculation.
    DO root=0,size-1
      IF (j+root.LE.Neta+1) THEN
        CALL MPI_SCATTER(c,Nx,MPI_DOUBLE_COMPLEX,  &
          NxTempZ,Nx,MPI_DOUBLE_COMPLEX,  &
          root,MPI_COMM_WORLD,errcode)

        DO i=1,Nx
          voutRe(j+root,i)=DREAL(NxTempZ(i))*scaling2
          voutIm(j+root,i)=DIMAG(NxTempZ(i))*scaling2
        END DO

      END IF
    END DO

  END DO

end subroutine x_FourierDifferentiator

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! The TriDiag_Solver solves a linear tri-diagonal
! equation system, using the Thomas algorithm.
! Reference: John C. Strikwerda, Finite Difference
!            Schemes and Partial Differential Equations,
!            Chapman & Hall, 1989
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
subroutine TriDiag_Solver(a,b,c,d,r,p,w,m,iopt)
  ! Arguments:
  ! a - vector containing the subdiagonal of matrix
  ! b - vector containing the diagonal of matrix
  ! c - vector containing the upper diagonal of matrix
  ! d - vector containing the righthand side
  ! p - vector containing a working area
  ! r - vector containing a working area
  ! w - the resulting vector - output
  ! m - the number of unknowns in eqation system
  ! iopt - option: iopt=1: Create help vector p
  !                iopt=2: Solve the system
  !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
  IMPLICIT NONE

  ! Arguments
  REAL(8),DIMENSION(:) :: a,b,c,d,r,p,w
  INTEGER :: m,iopt

  REAL(8) :: q(1:m)

  INTEGER :: i

  IF (iopt.EQ.2) THEN
    q(1)=d(1)
```

```
    DO i=1,m-1
      q(i+1)=r(i)*(d(i+1)-a(i+1)*q(i))
    END DO
    w(m+1)=(-a(m+1)*q(m)+d(m+1))/(1+a(m+1)*p(m))
    DO i=m,1,-1
      w(i)=p(i)*w(i+1)+q(i)
    END DO
  ELSE IF(iopt.EQ.1) THEN
    p(1) = -c(1)
    DO i=1, m-1
      r(i)=one/(a(i+1)*p(i)+b(i+1))
      p(i+1)=-c(i+1)*r(i)
    END DO
  END IF
end subroutine TriDiag_Solver

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! The eta_Differetiator calculates d/deta of a two-
! dimensional function f(x,eta), using the Pade'
! approximation.
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
subroutine eta_Differentiator(domain,vin,vout,deta,ReIm,Nx,Neta)
  ! Calculate d/deta of 'vin', the result returned in 'vout'
  IMPLICIT NONE

  ! Declarations of arguments
  TYPE (OneDimVlasovPlasma) :: domain
  REAL(8),DIMENSION(:,:) :: vin,vout
  REAL(8) :: deta
  CHARACTER :: ReIm
  INTEGER :: Nx,Neta

  ! Declarations of local variables
  INTEGER :: i,j,iopt
  REAL(8) :: rhs(Neta+1),frac_1_deta, frac_3_deta

  ! To be used by TriDiag_Solver
  REAL(8) :: a(Neta+1),b(Neta+1),c(Neta+1)
  REAL(8) :: r(Neta+1),p(Neta+1),w(Neta+1)

  frac_1_deta=one/deta
  frac_3_deta=three/deta

  IF (ReIm.EQ.'R') THEN
    a(1)=zero
    b(1)=one
    c(1)=zero
  ELSE
    a(1)=zero
    b(1)=one
    c(1)=half
  END IF

  DO i=2,Neta
    a(i)=one
    b(i)=four
```

```
    c(i)=one
  END DO

  a(Neta+1)=two
  b(Neta+1)=one
  c(Neta+1)=zero

  iopt=1
  CALL TriDiag_Solver(a,b,c,rhs,r,p,w,Neta,iopt)

  DO i=1,Nx
    ! The righthand side of the equation
    ! The boundary eta=0
    IF (ReIm.EQ.'R') THEN
      ! The Real part is an even function
      rhs(1)=zero
    ELSE
      ! The Imaginary part is an odd function
      rhs(1)=vin(2,i)*frac_3_deta*half
    END IF
    ! The inner points
    DO j=2,Neta
      rhs(j)=(vin(j+1,i)-vin(j-1,i))*frac_3_deta
    END DO
    ! The boundary eta=eta_max
    rhs(Neta+1)=-(-five*vin(Neta+1,i)+  &
      four*vin(Neta,i)   &
      +vin(Neta-1,i))*half*frac_1_deta

    iopt=2
    CALL TriDiag_Solver(a,b,c,rhs,r,p,w,Neta,iopt)

    ! Store the solution of the equation
    DO j=1,Neta+1
      vout(j,i)=w(j)
    END DO
  END DO

end subroutine eta_Differentiator


!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! The SimpsonIntegrator integrates a function f(x),
! using the Simpson formula.
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
subroutine SimpsonIntegrator(vin,d,N,result)
  ! Perform the integral of a function 'vin'
  IMPLICIT NONE

  ! Arguments
  REAL(8),DIMENSION(:) :: vin
  REAL(8) :: d,result
  INTEGER :: N

  REAL(8) :: frac_d_3,frac_2d_3,frac_4d_3
  INTEGER :: i
```

```
  ! Use Simpson's formula for integration
  frac_d_3=d*frac_1_3
  frac_2d_3=d*frac_2_3
  frac_4d_3=d*frac_4_3

  result = frac_d_3*(vin(1)+vin(N+1))+frac_4d_3*vin(N)
  DO i = 1,N/2-1
    result = result +frac_4d_3*vin(i+i)+frac_2d_3*vin(i+i+1)
  END DO
end subroutine SimpsonIntegrator

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! The PeriodicIntegrator integrates a periodic function f(x),
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
subroutine PeriodicIntegrator(vin,d,N,result)
  ! Perform the integral of a function 'vin'
  IMPLICIT NONE

  ! Arguments
  REAL(8),DIMENSION(:) :: vin
  REAL(8) :: d,result
  INTEGER :: N

  INTEGER :: i

  result = zero
  DO i = 1,N
    result = result+vin(i)*d
  END DO
end subroutine PeriodicIntegrator

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Runge-Kutta time stepper
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
subroutine RungeKutta_TimeStepper(domain,dt,x1,x2,Echoice)
  IMPLICIT NONE

  ! Arguments
  TYPE(OneDimVlasovPlasma) :: domain
  REAL(8) :: density(Nx),dt,x1,x2
  INTEGER :: Echoice

  REAL(8) :: fre_1(Neta+1,Nx),fim_1(Neta+1,Nx)
  REAL(8) :: fre_2(Neta+1,Nx),fim_2(Neta+1,Nx)
  REAL(8) :: Pre1(Neta+1,Nx),Pim1(Neta+1,Nx)
  REAL(8) :: frac_dt_2, frac_dt_6,frac_2dt_6,E123(1:3)
  INTEGER :: i,j



  frac_dt_2=dt/two
  frac_dt_6=dt/six
  frac_2dt_6=two*dt/six

    DO i=1,Nx
```

```
  DO j=1,Neta+1
    fre_1(j,i)=domain%fre(j,i)
  END DO
END DO

DO i=1,Nx
  DO j=1,Neta+1
    fim_1(j,i)=domain%fim(j,i)
  END DO
END DO

CALL PDE_RHS (domain,fre_1,fim_1,Pre1,Pim1,  &
  domain%E,domain%dx,domain%deta,x1,x2,Ezhoize)

DO i=1,Nx
  DO j=1,Neta+1
    domain%fre(j,i)=fre_1(j,i)+Pre1(j,i)*frac_dt_6
  END DO
END DO

DO i=1,Nx
  DO j=1,Neta+1
    domain%fim(j,i)=fim_1(j,i)+Pim1(j,i)*frac_dt_6
  END DO
END DO

DO i=1,Nx
  DO j=1,Neta+1
    fre_2(j,i)=fre_1(j,i)+Pre1(j,i)*frac_dt_2
  END DO
END DO

DO i=1,Nx
  DO j=1,Neta+1
    fim_2(j,i)=fim_1(j,i)+Pim1(j,i)*frac_dt_2
  END DO
END DO

CALL PDE_RHS (domain,fre_2,fim_2,Pre1,Pim1,  &
  domain%E,domain%dx,domain%deta,x1,x2,Ezhoize)

DO i=1,Nx
  DO j=1,Neta+1
    domain%fre(j,i)=domain%fre(j,i)+Pre1(j,i)*frac_2dt_6
  END DO
END DO

DO i=1,Nx
  DO j=1,Neta+1
    domain%fim(j,i)=domain%fim(j,i)+Pim1(j,i)*frac_2dt_6
  END DO
END DO

DO i=1,Nx
  DO j=1,Neta+1
    fre_2(j,i)=fre_1(j,i)+Pre1(j,i)*frac_dt_2
```

```
      END DO
    END DO

    DO i=1,Nx
      DO j=1,Neta+1
        fim_2(j,i)=fim_1(j,i)+Pim1(j,i)*frac_dt_2
      END DO
    END DO

    CALL PDE_RHS (domain,fre_2,fim_2,Pre1,Pim1,  &
      domain%E,domain%dx,domain%deta,x1,x2,Echoice)

    DO i=1,Nx
      DO j=1,Neta+1
        domain%fre(j,i)=domain%fre(j,i)+Pre1(j,i)*frac_2dt_6
      END DO
    END DO

    DO i=1,Nx
      DO j=1,Neta+1
        domain%fim(j,i)=domain%fim(j,i)+Pim1(j,i)*frac_2dt_6
      END DO
    END DO

    DO i=1,Nx
      DO j=1,Neta+1
        fre_2(j,i)=fre_1(j,i)+Pre1(j,i)*dt
      END DO
    END DO

    DO i=1,Nx
      DO j=1,Neta+1
        fim_2(j,i)=fim_1(j,i)+Pim1(j,i)*dt
      END DO
    END DO

    CALL PDE_RHS (domain,fre_2,fim_2,Pre1,Pim1,  &
      domain%E,domain%dx,domain%deta,x1,x2,Echoice)

    DO i=1,Nx
      DO j=1,Neta+1
        domain%fre(j,i)=domain%fre(j,i)+Pre1(j,i)*frac_dt_6
      END DO
    END DO

    DO i=1,Nx
      DO j=1,Neta+1
        domain%fim(j,i)=domain%fim(j,i)+Pim1(j,i)*frac_dt_6
      END DO
    END DO

    CALL ElectricField(domain%fre,domain%E,domain%E123,  &
      domain%dx,Echoice,x1,x2,Nx,Neta)
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
  end subroutine RungeKutta_TimeStepper
```

```
end module vlasov_numeric_mod
!
```

A.3 vlasov_domain_mod.f90

```
!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! vlasov_domain_mod.f90   Bengt Eliasson   2000-03-18       !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Conatains the class OneDimVlasovPlasma, and constructors/ !
! destructors for this class.                               !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

module vlasov_domain_mod
  USE vlasov_param_mod

  IMPLICIT NONE

  type OneDimVlasovPlasma
    ! The distribution function
    REAL(8), DIMENSION(1:Neta+1,1:Nx) :: fre, fim
    ! The electric field
    REAL(8), DIMENSION(1:Nx) :: E
    ! The step sizes in x and eta direction
    REAL(8) :: dx,deta,E123(1:3)

  end type

  CONTAINS

  ! Constructor and Destructor, used if dynamic memory is used ...
  SUBROUTINE Constructor(domain,Nx,Neta)
    IMPLICIT NONE
    TYPE (OneDimVlasovPlasma) :: domain
    INTEGER :: Nx, Neta
  END SUBROUTINE Constructor

  SUBROUTINE Destructor(domain)
    IMPLICIT NONE
    TYPE (OneDimVlasovPlasma) :: domain
  END SUBROUTINE Destructor
end module vlasov_domain_mod
!
```

A.4 vlasov_param_mod.f90

```
!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! vlasov_param_mod.f90    Bengt Eliasson   2000-03-18  !
```

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! This module contains mathematical constants and      !
! parameters used to set up a run.                     !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

MODULE vlasov_param_mod
  IMPLICIT NONE

  !---- Include file for MPI and some various
  !---- parameters and variables
  include 'mpif.h'
  INTEGER :: errcode,size,rank

  ! The number of processors
  INTEGER,PARAMETER :: NP=10

  !---- Static parameters
  REAL(8),PARAMETER :: pi=3.14159265358979323846d0
  REAL(8),PARAMETER :: zero=0.0d0
  REAL(8),PARAMETER :: one=1.0d0
  REAL(8),PARAMETER :: two=2.0d0
  REAL(8),PARAMETER :: three=3.0d0
  REAL(8),PARAMETER :: four=4.0d0
  REAL(8),PARAMETER :: five=5.0d0
  REAL(8),PARAMETER :: six=6.0d0
  REAL(8),PARAMETER :: seven=7.0d0
  REAL(8),PARAMETER :: ten=10.0d0
  REAL(8),PARAMETER :: thirteen=13.0d0
  REAL(8),PARAMETER :: fourteen=14.0d0
  REAL(8),PARAMETER :: twentytwo=22.0d0
  REAL(8),PARAMETER :: twopi=6.28318530717958647692d0
  REAL(8),PARAMETER :: halfpi=1.57079632679489661923d0
  REAL(8),PARAMETER :: half=0.5d0
  REAL(8),PARAMETER :: frac_1_6=0.166666666666666666666666d0
  REAL(8),PARAMETER :: frac_5_2=2.5d0
  REAL(8),PARAMETER :: frac_1_3=0.333333333333333333333333d0
  REAL(8),PARAMETER :: frac_2_3=0.666666666666666666666667d0
  REAL(8),PARAMETER :: frac_4_3=1.333333333333333333333333d0
  REAL(8),PARAMETER :: SqrtThree=1.73205080756887729352744d0
  REAL(8),PARAMETER :: SqrtEight=2.82842712474619009760404d0

  ! File units
  INTEGER,PARAMETER :: outE_1=21
  INTEGER,PARAMETER :: outE_2=22
  INTEGER,PARAMETER :: outE_3=23
  INTEGER,PARAMETER :: outdP=18
  INTEGER,PARAMETER :: outdW=19
  INTEGER,PARAMETER :: outCorr = 12
  INTEGER,PARAMETER :: outFourier = 13
  INTEGER,PARAMETER :: outE2 = 14
  INTEGER,PARAMETER :: outNorm = 15
  INTEGER,PARAMETER :: outfabs = 16
  INTEGER,PARAMETER :: outF = 17

  !---- Settings for a run ----!
  !----- Parameters for the domain ----!
```

```
  ! The total number of gridpoints in x-direction.
  INTEGER,PARAMETER :: TotalNx=100

  ! The number of gridpoints on this processor.
  INTEGER,PARAMETER :: Nx=TotalNx/NP

  ! The number of grid points in eta-direction.
  INTEGER,PARAMETER :: Neta=100

  ! The wave number used in the initial condition
  REAL(8),PARAMETER :: kx = 0.5d0

  ! The amplitude used in the initial condition
  REAL(8),PARAMETER :: A = 0.5D0

  ! The domain in x direction
  REAL(8),PARAMETER :: x1=zero
  REAL(8),PARAMETER :: x2=2.0d0*pi/kx

  ! The domain in eta direction
  REAL(8),PARAMETER :: eta_max=30.0d0

  ! Numerical dissipation
  REAL(8),PARAMETER :: diss=0.001d0

  ! The choice of model for the electric field
  INTEGER,PARAMETER :: EChoice=0

  !---- Other parameters for the run ----!
  ! The number of steps in time.
  INTEGER,PARAMETER :: Nt=5000

  ! Stability parameters
  ! The parameter combining the stability condition for the
  ! Runge-Kutta, Spectral scheme and Pade' scheme.
  REAL(8),PARAMETER :: S_limit=SqrtEight/(SqrtThree*pi)
  ! The CFL number, CFL<1 for stability
  REAL(8),PARAMETER :: CFL=0.71D0

  ! The time step size
  REAL(8),PARAMETER :: dt=CFL*S_limit*((x2-x1)/TotalNx)*(eta_max/Neta)

  ! The stop time
  REAL(8),PARAMETER :: t_end=Nt*dt

  ! If data files is to be be created
  ! Yes: .TRUE.
  ! No: .FALSE.
  LOGICAL,PARAMETER :: WriteToFile=.TRUE.

  ! The number of times to write result
  INTEGER,PARAMETER :: NrPrints=1000
  INTEGER,PARAMETER :: print1 = Nt/NrPrints

END MODULE vlasov_param_mod
!
```