ELECTROSTATIC WAVES IN PLASMAS

....with reference to Landau Damping

A thesis submitted by

Kushal Kumar Shah

EE01B040

in partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY

under the esteemed guidance of

Dr. Hari Ramachandran

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS
Chennai - 600036

May 2005

CERTIFICATE

This is to certify that the work contained in the thesis entitled "**Electrostatic Waves in Plasmas**" submitted by Kushal Kumar Shah (EE01B040) in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Electrical Engineering is a bona fide work carried out by him under my guidance and supervision.

Project Guide

Dr. Hari Ramachandran
Associate Professor
Department of Electrical Engineering
Indian Institute of Technology Madras

Place: Chennai

Date:

Acknowledgments

At times our own light goes out and is rekindled by a spark from another person. Each of us has cause to think with deep gratitude of those who have lighted the flame within us.

— Albert Schweitzer

My one year long association with my B.Tech project guide, Dr. Hari Ramachandran, has been a wonderful learning experience. I would like to extend my heartfelt gratitude to Dr. Ramachandran for his help, guidance and the patience with which he taught me the intricate but interesting subject of Plasma Physics. This project has been the most enjoyable part of my academic life at IIT Madras.

I will be ever obliged to IIT Madras in general and the Electrical Engineering Department in particular for having provided me with such an intellectually stimulating environment that is conducive to high academic growth. It was only due to the flexible system of IIT Madras that I was able to do my final year project in an unconventional area like Plasma Physics.

I am thankful to all my professors at IIT Madras who taught me the basic principles of topics like Laplace transform, Fourier transform, differential equations, electromagnetic fields, to name a few, without which it would have been impossible for me to do research in the area of Plasma Physics.

Finally, I would like to thank my parents, Mr. and Mrs. Kishan Shah, for the motivation that they have provided me throughout my academic career in general and the past one year in particular.

Kushal Kumar Shah

May, 2005

Abstract

This thesis considers the effects of an electrostatic wave on a pure electron one dimensional plasma where the ions are essentially at rest. This is primarily a numerical analysis. We begin by trying to understand what happens to the zero order Maxwellian density distribution function in the presence of a landau damped sinusoidal electrostatic field. This is for the case of a homogeneous plasma. Then we turn our attention to an initially inhomogeneous plasma. The electrostatic field is no longer sinusoidal. We come across what is called the airy function. Now, we add some noise to the plasma in the form of a random phased spectrum of waves. So, instead of a single airy function, our electrostatic field is now described by a sum of shifted versions of multiple airy functions with random phase. In the end, we allow our electrostatic field to be modified due to changes in the density distribution function itself. The field is airy only as long as the density gradient is linear. But as the particles move around, the linearity is no longer maintained. We were able to reach a steady state where the field and the distribution function were self-consistent. One of the graphs obtained from the simulation also bears the fingerprints of landau damping.

Contents

List of Figures

List of Tables

Part I

THEORY

Chapter 1

Introduction

In this thesis we present our study of *electrostatic waves* in homogeneous as well as inhomogeneous plasma with reference to the phenomenon of *landau damping*.

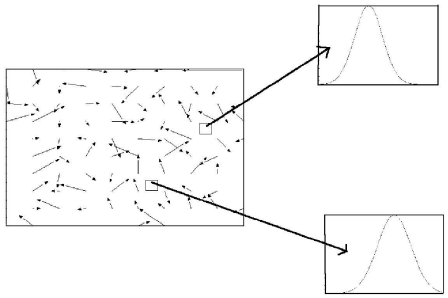

Figure 1.1: Statistical description of a plasma. The random arrows represent the thermal velocity of the particles. The velocity distribution of the particles is fairly Maxwellian. But the characteristics of this Maxwellian are different in different parts of the plasma. The electron and ion temperature can be different in different parts of the plasma.

Figure 1.1 gives a statistical description of a plasma. The random arrows represent the thermal velocity of the particles. The velocity distribution of the particles

is fairly Maxwellian. But the characteristics of this Maxwellian are different in different parts of the plasma. The electron and ion temperature can be different in different parts of the plasma. An electron or ion can even have different temperatures along the three co-ordinate axes. For all our present analysis we have considered a *one-dimensional warm electron plasma* in which the ions are essentially at rest.

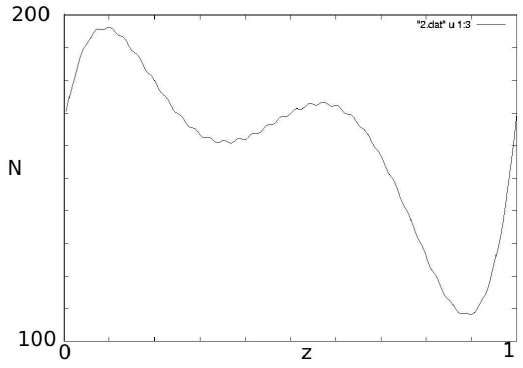

Figure 1.2: Small oscillations in plasma density. This graph shows an arbitrary background density distribution function on which a sinusoidal density perturbation is superimposed.

Figure 1.2 depicts the small oscillations in the plasma density superimposed on the background electron density.

Figure 1.3 depicts the effect of landau damping on the velocity dependent distribution function of a plasma. It can be seen that a small region of the Gaussian curve has been flattened. This happens near the region where the velocity of the electrons is close to the phase velocity of the wave. This is primarily due to particle trapping and phase mixing. Figure 1.4 shows what we mean by particle trapping.

In the field of plasma physics, the term "landau damping" is used to describe two separate damping mechanisms. One is the resonance damping of a wave and

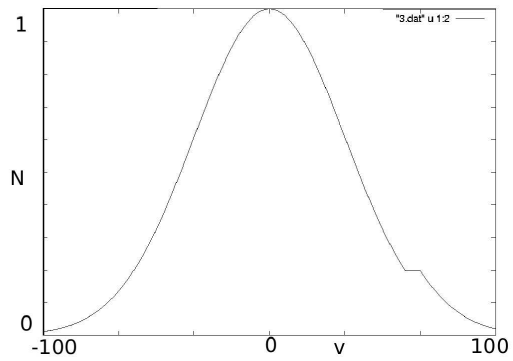

Figure 1.3: Flattening of the Gaussian distribution function due to landau damping.

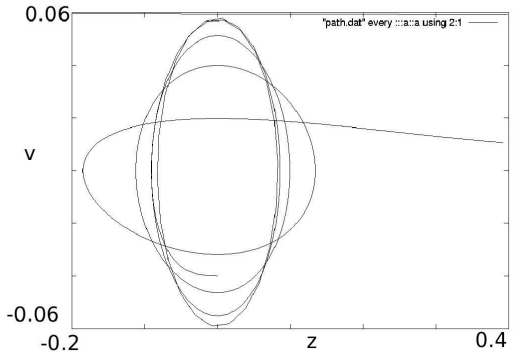

Figure 1.4: Trajectory of a trapped particle in phase space in the reference frame in which the wave is at rest.

the second is the damping of beat waves. We use the term landau damping to describe the *resonance damping* of waves. This kind of damping occurs mainly due to energy transfer from the wave to the particles that are moving at speeds close to the phase velocity of the wave. It must however be noted that this resonance interaction can also cause growth which happens when the number of particles with velocity slightly higher than the phase velocity of the wave is more than the

particles with a slightly lower velocity. The particles traveling with velocity away from the phase velocity of the wave do not get involved in the damping or growth mechanisms.

For an initially homogeneous plasma, landau damping of electrostatic waves[1] is understood to quite a large extent. But for inhomogeneous plasmas, the phenomenon is highly non-trivial. Considerable amount of work has been done by researchers all around the world in this area but still we do not have a very clear picture of what exactly happens when we have non-uniformities in the plasma. Although we were not able to get a perfect understanding of the underlying phenomenon, the insights derived from our work may be useful for further research on the topic.

The homogeneous case can be analyzed by usage of the *Vlasov equation* coupled with the *Poisson's equation*. But for the inhomogeneous case, we have to content ourselves with the *fluid equations*. There are a few publications that have dealt with inhomogeneous plasmas using the Vlasov equation but those are beyond the scope of our present work. But even those few publications are more like works of mathematics with a lot of assumptions. The physics remains obscure.

First order perturbation theory and linearization form the common theme of our mathematical methods. Since we do not have exact analytical expressions for describing the reaction of the plasma to the electrostatic waves in homogeneous as well as inhomogeneous plasmas, we have to fall back on simulations to get a *feel* of the physics involved in the process.

This thesis presents our work in increasing order of complexity. First, we consider the case of a homogeneous plasma. Then, we go on to inhomogeneous plasma where we first consider the case of the electrostatic field modeled as a single

airy function (Figure 1.5).

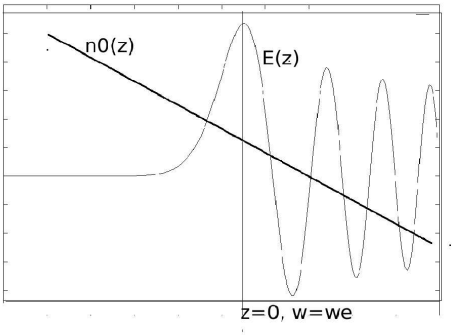

Figure 1.5: The airy function. The frequency of time oscillations, w, of this wave is equal to the plasma frequency, we, at z=0. Hence, the wave has a cut-off at this point.

Then, we discuss the case of a noisy plasma. But for all the above cases we do not consider the nonlinear modification of the electric field itself due to density perturbations. In the last section, we consider this effect and try to understand how the electric field would react to changes in the zero-order plasma density.

All simulations were done in *Fortran 77*. For solving differential equations, doing integrations and most of the purely mathematical things, use was made of the codes given in *Numerical Recipes in Fortran 77*.

Chapter 2

Homogeneous Plasmas

In this chapter we discuss the phenomenon of landau damping for the case of a homogeneous plasma[1]. We have used the *Vlasov* and the *Poisson's* equations for solving the problem. Analytical results pertaining to the problem can be obtained from *Nicholson*[2].

Since the plasma is homogeneous, we can Fourier transform the equations in time as well as space. To get landau damping, we must consider the problem as an initial value problem. Now, when we have damping, the frequency of time oscillations is no longer real. It becomes complex. We use this information for simulating the particle trajectories. The electrostatic field is modeled as an exponentially decaying sinusoidal function. The initial distribution function is assumed to be Maxwellian.

It was found that particles with initial velocity lower than the phase velocity of the wave were speeded up and vice versa. The particles with velocity far away from the phase velocity of the wave do not get involved in the damping mechanisms. We also see a very interesting phenomenon called *particle trapping*. The trajectories of two such trapped particles can be seen in Figures 2.1 and 2.2.

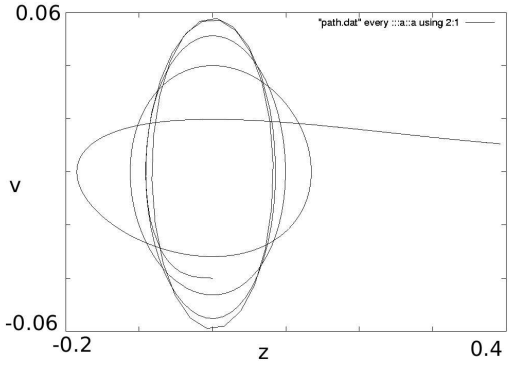

Figure 2.1: This is the plot of the trajectory of a trapped particle for the case of a homogeneous plasma. This is in the wave frame.

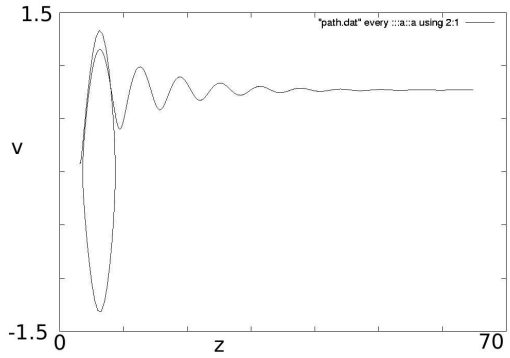

Figure 2.2: This is the plot of the trajectory of a trapped particle for the case of a homogeneous plasma. This is in the wave frame. Unlike the previous figure, in this one, the particle is trapped only for a short time and then it gets free.

Some of the particles get essentially trapped in the wave. If the wave were not damped, these particles would be forever trapped. But since we have damping, the trapped particles happen to escape out at some point of time. It is these trapped particles that are responsible for wave damping. In the initial stages, the damping is linear. But when the trapped particles turn around to complete their first circular trajectory, the damping enters its nonlinear phase. The nonlinear

part of landau damping is due to *phase mixing*.

The initial and the modified distribution functions are plotted in Figures 2.3 and 2.4.

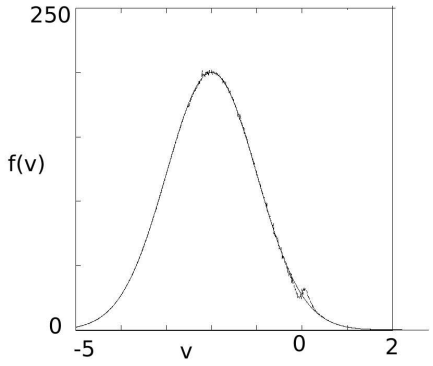

Figure 2.3: This is the plot of the original and the modified distribution function for the case of a homogeneous plasma. The horizontal axis is the velocity and the vertical axis is the number of particles. The velocity v=0 represents the resonance point. This is in the wave frame. It can be clearly seen that the particles that were initially on the left of v=0 have now moved to the right.

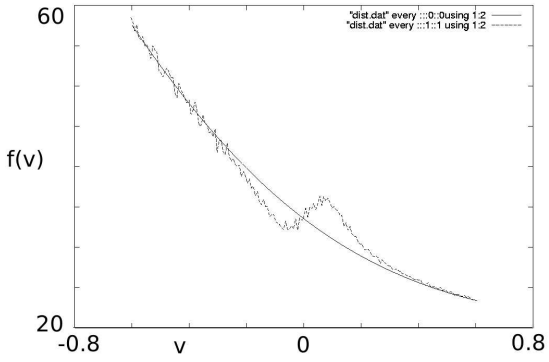

Figure 2.4: This is the zoomed-in version of the previous figure.

Chapter 3

Inhomogeneous Plasmas

To analyze the nature of electrostatic wave propagation in nonuniform plasmas, we have used the fluid equations coupled with the Poisson's equation. Let us forget landau damping for a while.

We have modeled our non uniformity as a linear density profile. The electrostatic wave turns out to be an *airy* function. The derivation of the airy equation is given below.

We have used the 1st order perturbation theory to solve for the electric field. The density is modeled as $n = n_0 + n_1$ and similarly for velocity and electric field. Here n_0 represents the zero order quantity and n_1 is the small perturbation in the zero order value. These quantities are described below.

The zero-oder quantities are,

$$n_0 = n_0(z) \qquad \text{Density} \tag{3.1}$$

$$v_0 = 0 \qquad \text{Velocity} \tag{3.2}$$

$$E_0 = 0 \qquad \text{Electrostatic Field} \tag{3.3}$$

The 1st-oder quantities are,

$$n_1 = n_1(z)\exp(-iwt) \qquad \text{Density} \tag{3.4}$$

$$v_1 = v_1(z)\exp(-iwt) \qquad \text{Velocity} \tag{3.5}$$

$$E_1 = E_1(z)\exp(-iwt) \qquad \text{Electrostatic Field} \tag{3.6}$$

The only equations used to derive the results are the fluid equations and the Poisson's equation.

1. Continuity equation:

$$\frac{\partial n}{\partial t} + \vec{\nabla}\cdot(n\vec{v}) = 0 \tag{3.7}$$

2. Equation of motion:

$$m_e n\frac{\partial \vec{v}}{\partial t} + m_e n(\vec{v}\cdot\vec{\nabla})\vec{v} = nq_e\vec{E} + nq_e(\vec{v}\times\vec{B}) - \vec{\nabla}p \tag{3.8}$$

3. Poisson's equation:

$$\vec{\nabla}\cdot\vec{E} = \frac{q_e n}{\epsilon_o} \tag{3.9}$$

The zero-order continuity equation is satisfied easily. To satisfy the zeroth-order momentum equation we introduce a term D that takes care of the $\vec{\nabla}p$ term. This D can be attributed to the volume rate of the recombination, ionization, and loss processes in equilibrium[3].

Now, let us write the 1st order equations. We can use the fact that $pn^{-\gamma} = constant$.

1. Continuity equation:

$$\frac{\partial n_1}{\partial t} + \frac{\partial (n_0 v_1)}{\partial z} = 0 \tag{3.10}$$

2. Equation of motion:

$$m_e n_0 \frac{\partial v_1}{\partial t} = -n_0 e E_1 - \gamma K T \frac{\partial n_1}{\partial z} \tag{3.11}$$

3. Poisson's equation:

$$\frac{\partial E_1}{\partial z} = -\frac{e n_1}{\epsilon_o} \tag{3.12}$$

In writing the above 1st order equations, we have neglected the following terms,

- $n_1 v_1$ in the continuity equation
- $n_1 \partial v_1 \big/ \partial t$, $v_1 \partial v_1 \big/ \partial z$, $n_1 E_1$, in the momentum equation

Now, substituting for n_1, v_1 and E_1, in the momentum equation, we get,

$$-iwv_1(z) = \frac{-e}{m_e} E_1(z) - \frac{\gamma K T}{m_e n_0(z)} \frac{\partial n_1}{\partial z} \tag{3.13}$$

$$v_1(z) = \frac{i}{w}\Big[\frac{-e}{m_e} E_1(z) - \frac{\gamma K T}{m_e n_0(z)} \frac{\partial n_1(z)}{\partial z}\Big] \tag{3.14}$$

Substituting this in the first order continuity equation, we have,

$$w n_1 + \frac{n_0 e}{w m_e} \frac{\partial E_1}{\partial z} + \frac{\gamma K T}{m_e w} \frac{\partial^2 n_1}{\partial z^2} + \frac{e}{m_e w} \frac{\partial n_0}{\partial z} E_1 = 0 \tag{3.15}$$

Now, substituting the Poisson's equation, $\frac{\partial E_1}{\partial z} = -en_1/\epsilon_0$, and $n_0(z) = n_0(1 - z/L)$ into the above equation, we have,

$$\frac{v_e^2}{w^2}\frac{\partial^3 E_1}{\partial z^3} + \Big[1 - \frac{w_e^2}{w^2}(1 - z/L)\Big]\frac{\partial E_1}{\partial z} + \frac{w_e^2}{w^2 L}E_1 = 0 \tag{3.16}$$

In the above equation, $v_e^2 = \gamma KT\Big/m_e$, $w_e^2 = e^2 n_0\Big/m_e\epsilon_0$, and $L(z << L)$ is the scale length of the plasma. If we write $b = w_e^2\Big/v_e^2 L$ and use the fact that $w = w_e$, then, we can write the above equation as,

$$\frac{\partial^3 E_1}{\partial z^3} + bz\frac{\partial E_1}{\partial z} + bE_1 = 0 \tag{3.17}$$

$$\frac{\partial^3 E_1}{\partial z^3} + b(z\frac{\partial E_1}{\partial z} + E_1) = 0 \tag{3.18}$$

$$\frac{\partial^3 E_1}{\partial z^3} + b\frac{\partial z E_1}{\partial z} = 0 \tag{3.19}$$

Integrating the above equation w.r.t z, we get,

$$\frac{\partial^2 E_1}{\partial z^2} + bzE_1 = 0 \tag{3.20}$$

The above is the well known airy equation. This function has been plotted in Figure 3.1.

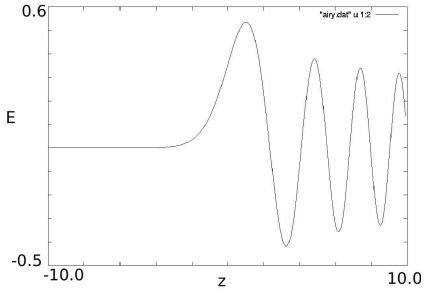

Figure 3.1: This is the electrostatic field when modeled as a single airy function.

3.1 Single Airy

This section considers the case where the electrostatic field is modeled as a single airy function. This is the simplest possible case for an inhomogeneous plasma. So far we have been able to find an equation for the electrostatic wave in a plasma with a linear density profile. But this is not the end of the road. We still do not know the effect of such an electric field on the plasma particles. This is where we leave the world of mathematics and enter the world of simulations. It should be noted that as the particles move about, the density profile does not remain linear. So, in reality, the electrostatic wave no longer remains airy. But, for the time being, we have neglected the modifications of the wave profile due to the motion of particles. We will take up this issue in a later section.

The results of the simulation are as follows:

z-range	v-range	endright (%)
0 to 10	-1.0 to 1.0	55.34
0 to 10	-2.0 to 2.0	57.37
0 to 10	-3.0 to 3.0	56.40
0 to 10	-3.5 to 3.5	55.67
0 to 10	-10.0 to 10.0	74.76
-10 to 10	-3.5 to 3.5	46.75

Table 3.1: Simulation results for the case of an inhomogeneous plasma when the electric field is modeled as a single airy function.

In Table 3.1, the *z-range* and the *v-range* are normalized values. *endright* is the percentage of particles that end up on the right side of the gradient i.e. towards the low density side. This can be understood better from Figures 3.4, 3.5 and 3.6

It can be seen from the above table that the movement of the particles is essentially diffusive in nature and the electric field tends to smoothen the density gradient.

For a better feel of particle trajectories please view Figures 3.2 and 3.3

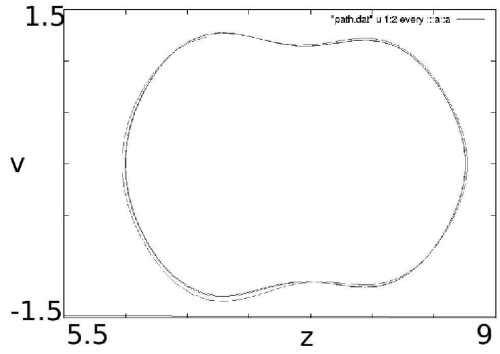

Figure 3.2: Plot of the trajectory of one of the trapped particles when the electrostatic field is a single airy function. We can see that this particle is trapped in the airy wave and will most probably remain so for ever. What is even more interesting is that it keeps on following the same path in phase space in every cycle. This is not so common.

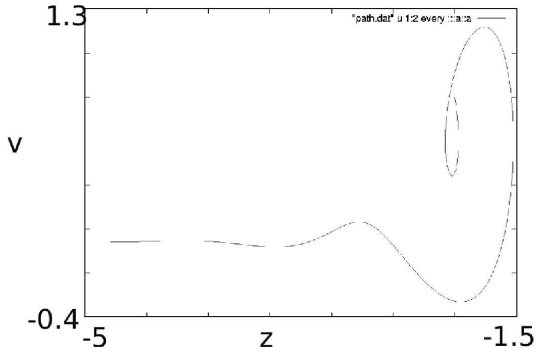

Figure 3.3: Plot of the trajectory of one of the particles when the electrostatic field is a single airy function. This particle gets trapped initially for a while but eventually frees itself.

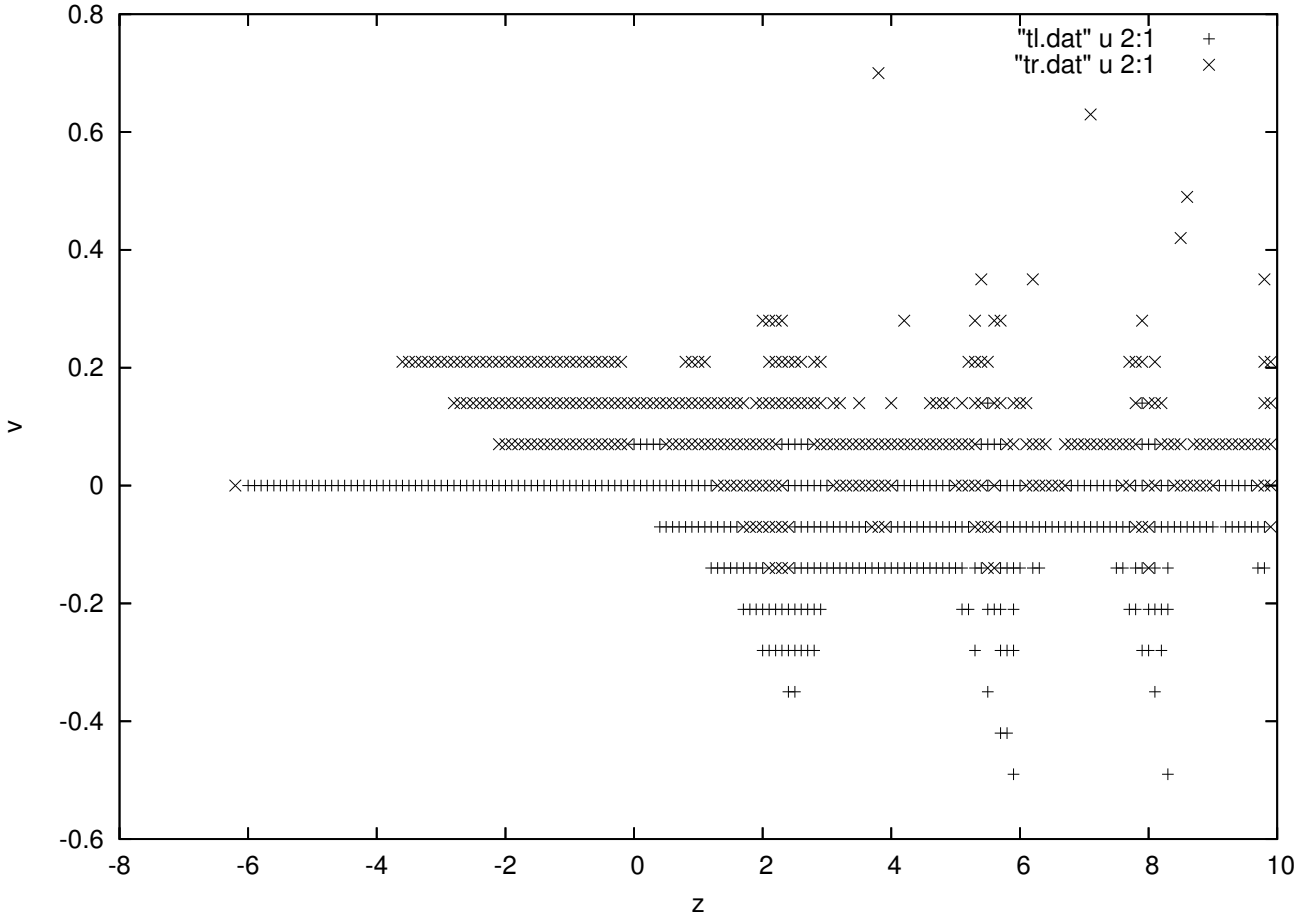

Figure 3.4: Plot of the final relative movements of particles for the case of inhomogeneous plasma where the electrostatic wave is a single airy function. This is for a very weak field. The × dots represent the particles that went to the left but eventually turned around and ended up on the right. The + dots represent the particles that went to the right but eventually turned around and ended up on the left. Here, right represents the low density side. It must also be noted that the dots also include those particles that turned around more than once. We can see that this graph shows much more structure when compared to Figures 3.5 and 3.6. Here the trapping happens only at certain discrete values of velocities, which can be inferred from the horizontal lines in the graph. But we do not clearly see a separate trapping velocity range. This is mainly because the electric field is not low enough.

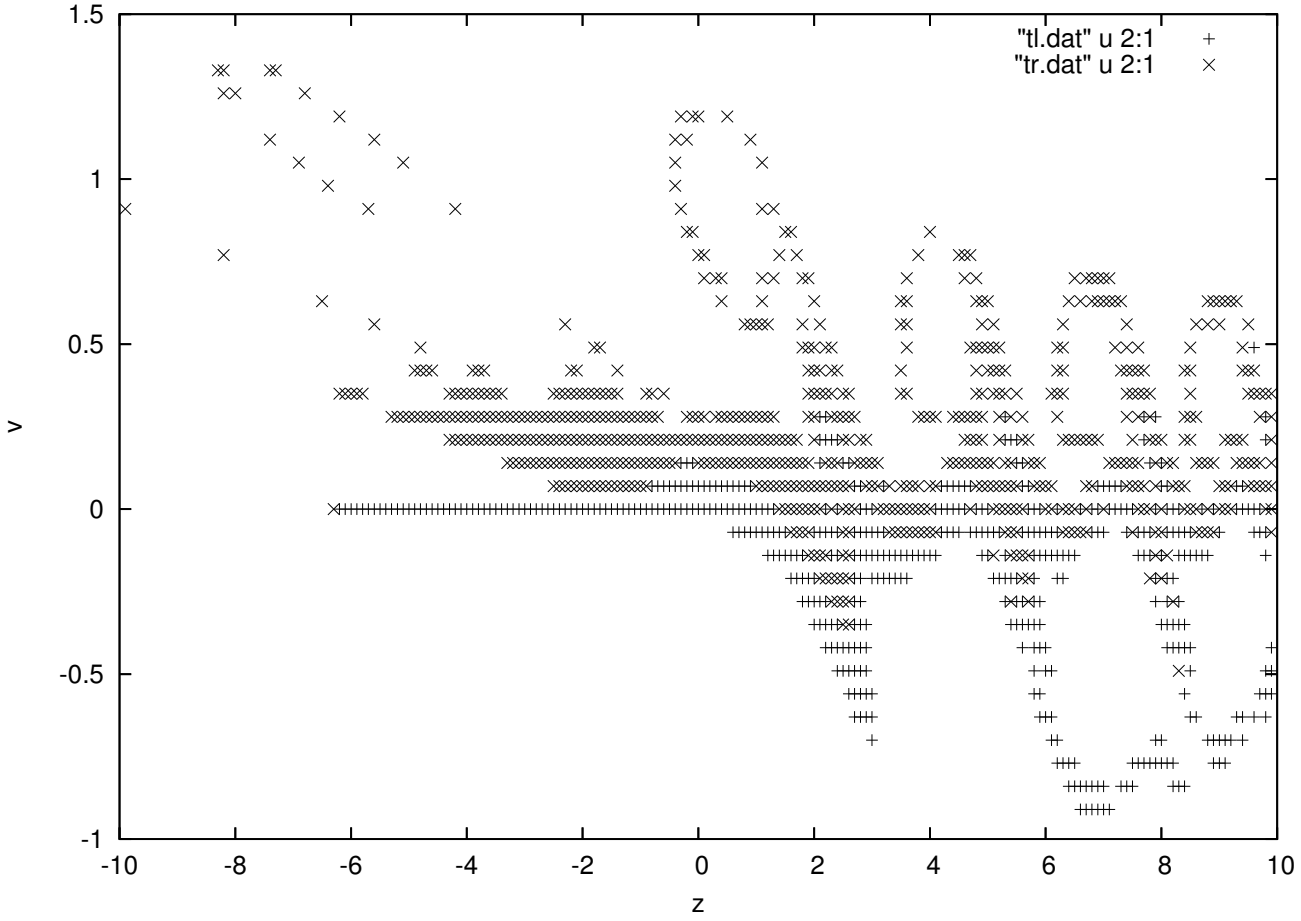

Figure 3.5: Plot of the final relative movements of particles for the case of inhomogeneous plasma where the electrostatic wave is a single airy function. This is for intermediate field strengths. The × dots represent the particles that went to the left but eventually turned around and ended up on the right. The + dots represent the particles that went to the right but eventually turned around and ended up on the left. Here, right represents the low density side. It must also be noted that the dots also include those particles that turned around more than once. This graph has a lot less structure when compared to Figure 3.4. We can see some horizontal lines but there are a lot of loops also formed. It is not very clear what these loops represent. These loops are certainly caused by the high nonlinearity involved in the simulation.

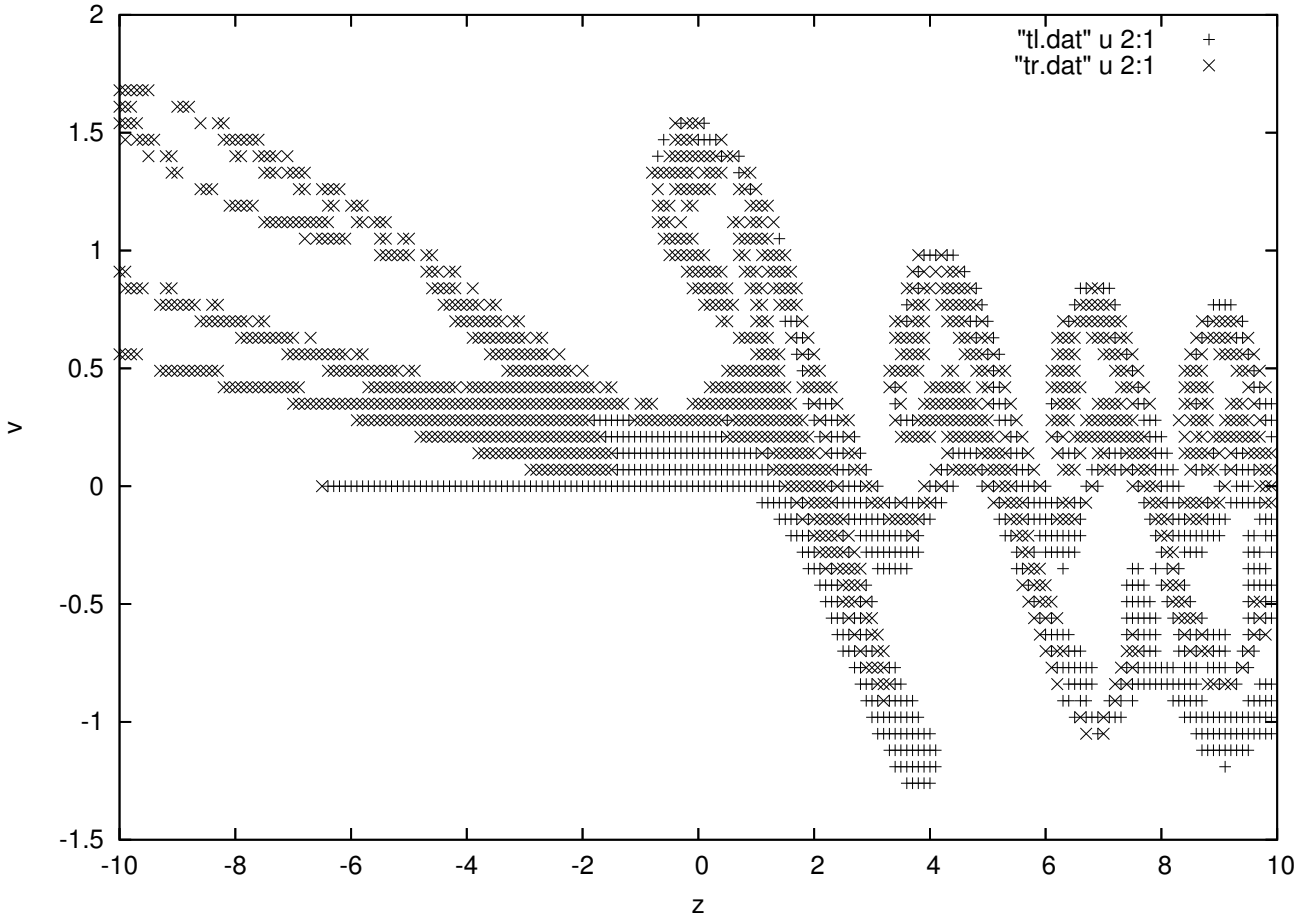

Figure 3.6: Plot of the final relative movements of particles for the case of inhomogeneous plasma where the electrostatic wave is a single airy function. This is for a very strong field. The $\times$ dots represent the particles that went to the left but eventually turned around and ended up on the right. The + dots represent the particles that went to the right but eventually turned around and ended up on the left. Here, right represents the low density side. It must also be noted that the dots also include those particles that turned around more than once. We can clearly see from Figures 3.4, 3.5 and the above that the number of particles that turn around in their trajectories goes up with the field intensity. It is very difficult to find any definite pattern in this graph. It is a highly nonlinear situation.

3.2 Multiple Airy

Now, we add some noise to the plasma in the form of a random phased spectrum of waves. In this case, the electrostatic field is essentially a sum of a large number of *shifted airy functions* with random phase. The electric field at t=0 is plotted as a function of the space in Figure 3.7. It is interesting to note that although the starting phases are random, at some points of time the individual airy functions do get tuned in. The simulations are carried out as in the previous section. The results of the simulation are:

v-range	endright (%)
-1.0 to 1.0	20.41
-2.0 to 2.0	17.86
-3.0 to 3.0	19.52
-3.5 to 3.5	10.40
-10.0 to 10.0	35.89

Table 3.2: Simulation results for the case of an inhomogeneous plasma when the electric field is modeled as a sum of multiple airy functions.

For all the above results, the z-range is -10.0 to 10.0. In Table 3.2, the *z-range* and the *v-range* are normalized values. *endright* is the percentage of particles that end up on the right side of the gradient i.e. towards the low density side. This can be understood better by having a look at Figures 3.15 and 3.16.

It can be seen that this case is very different from the earlier case of a single airy field. Now, majority of the particles are moving up the density gradient. Two of the particle trajectories can be viewed in Figures 3.13 and 3.14

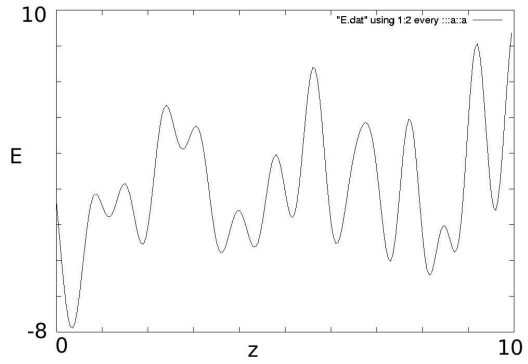

Figure 3.7: This is the electrostatic field when modeled as multiple airy functions at t=0. The horizontal axis is the z-coordinate.

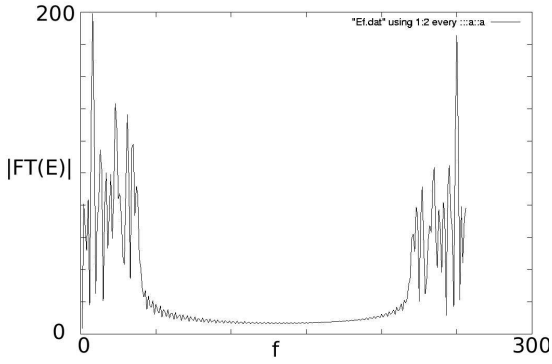

Figure 3.8: This is the magnitude of the space-Fourier transform of the electrostatic field when modeled as multiple airy functions at t=0.

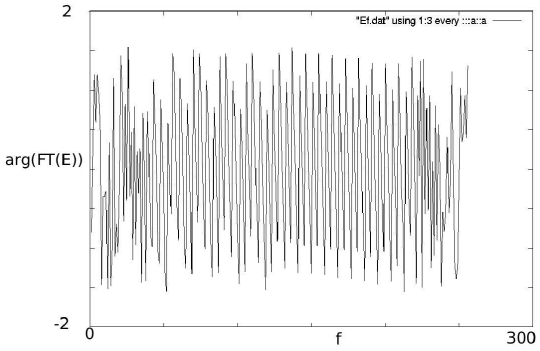

Figure 3.9: This is the phase of the space-Fourier transform of the electrostatic field when modeled as multiple airy functions at t=0. It can be seen that the shifted airy functions are essentially out of phase.

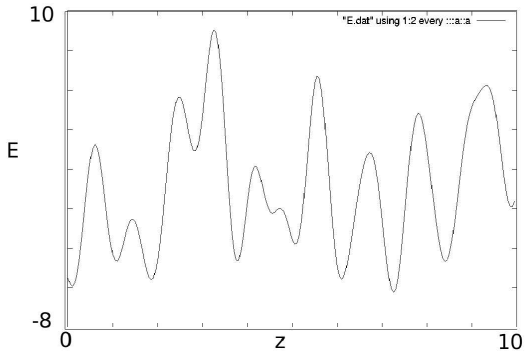

Figure 3.10: This is the electrostatic field when modeled as multiple airy functions at t=1.2. The horizontal axis is the z-coordinate.

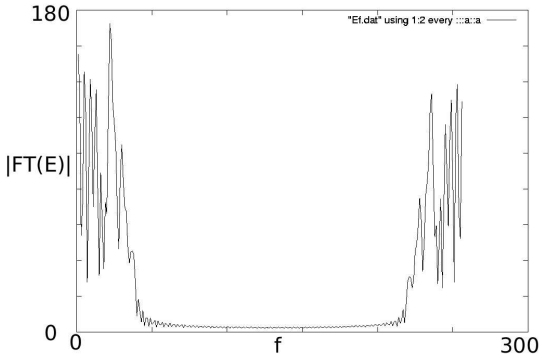

Figure 3.11: This is the magnitude of the space-Fourier transform of the electrostatic field when modeled as multiple airy functions at t=1.2.

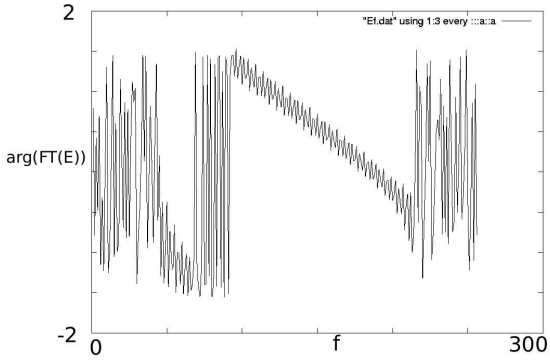

Figure 3.12: This is the phase of the space-Fourier transform of the electrostatic field when modeled as multiple airy functions at t=1.2. This is an interesting graph since it shows that at some instances of time the initially random airy functions do get tuned for a short time.

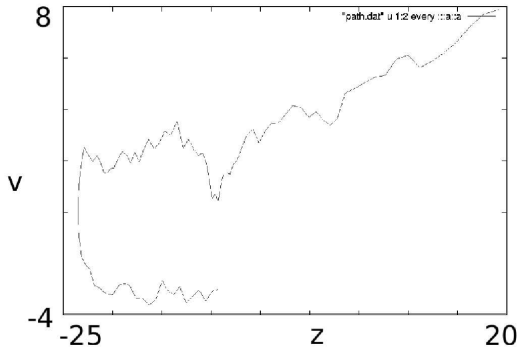

Figure 3.13: Plot of the trajectory of one of the particles when the electrostatic field is a sum of multiple airy functions. This is for the case of a noisy inhomogeneous plasma. This plot shows that the particle was trapped for a short time but before it could complete a cycle in phase space, it got detrapped.

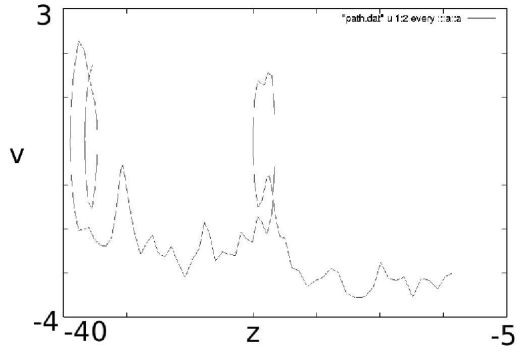

Figure 3.14: Plot of the trajectory of one of the particles when the electrostatic field is a sum of multiple airy functions. This is for the case of a noisy inhomogeneous plasma. This is a very interesting plot as it shows a particle that is undergoing active trapping and detrapping. We can see that the particle was trapped initially and then it got detrapped. But unlike other particles, this one managed to get trapped again at another location in phase space and got detrapped once again.

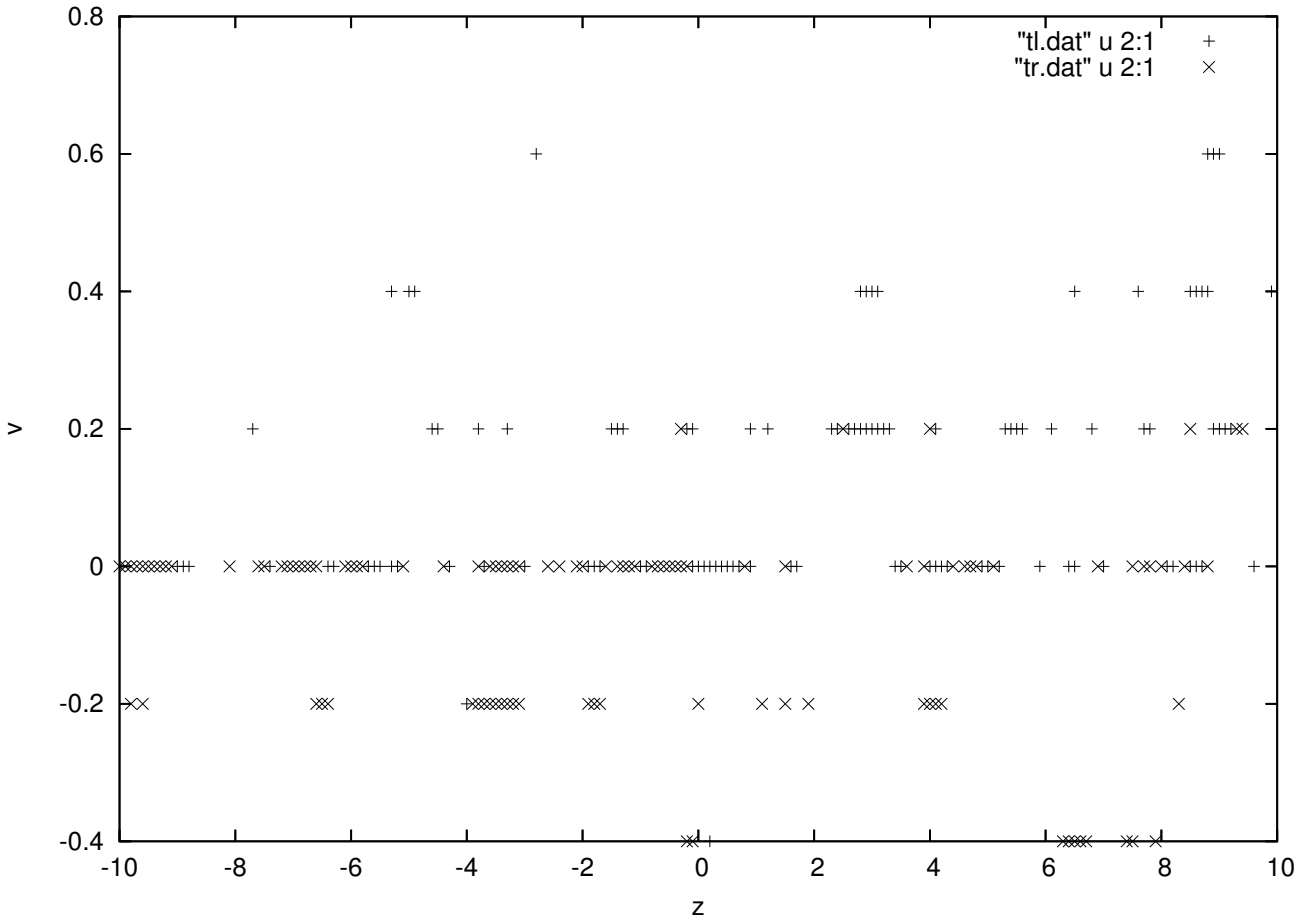

Figure 3.15: Plot of the final distribution of particles for the case of inhomogeneous plasma where the electrostatic wave is modeled as a sum of multiple airy functions. This is for the case of a very weak electric field. The vertical axis represents the x-range. The horizontal axis represents the v-range. The $\times$ dots represent the particles that went to the left but eventually turned around and ended up on the right. The $+$ dots represent the particles that went to the right but eventually turned around and ended up on the left. Here, right represents the low density side. It must also be noted that the dots also include those particles that turned around more than once. This graph shows a structure that is similar to Figure 3.4. We can see that most of the trapping at non zero velocities occurs around $v = \pm 0.2$. There are a lot of particles getting trapped at $v = 0$ but that is not the velocity range we are interested in.

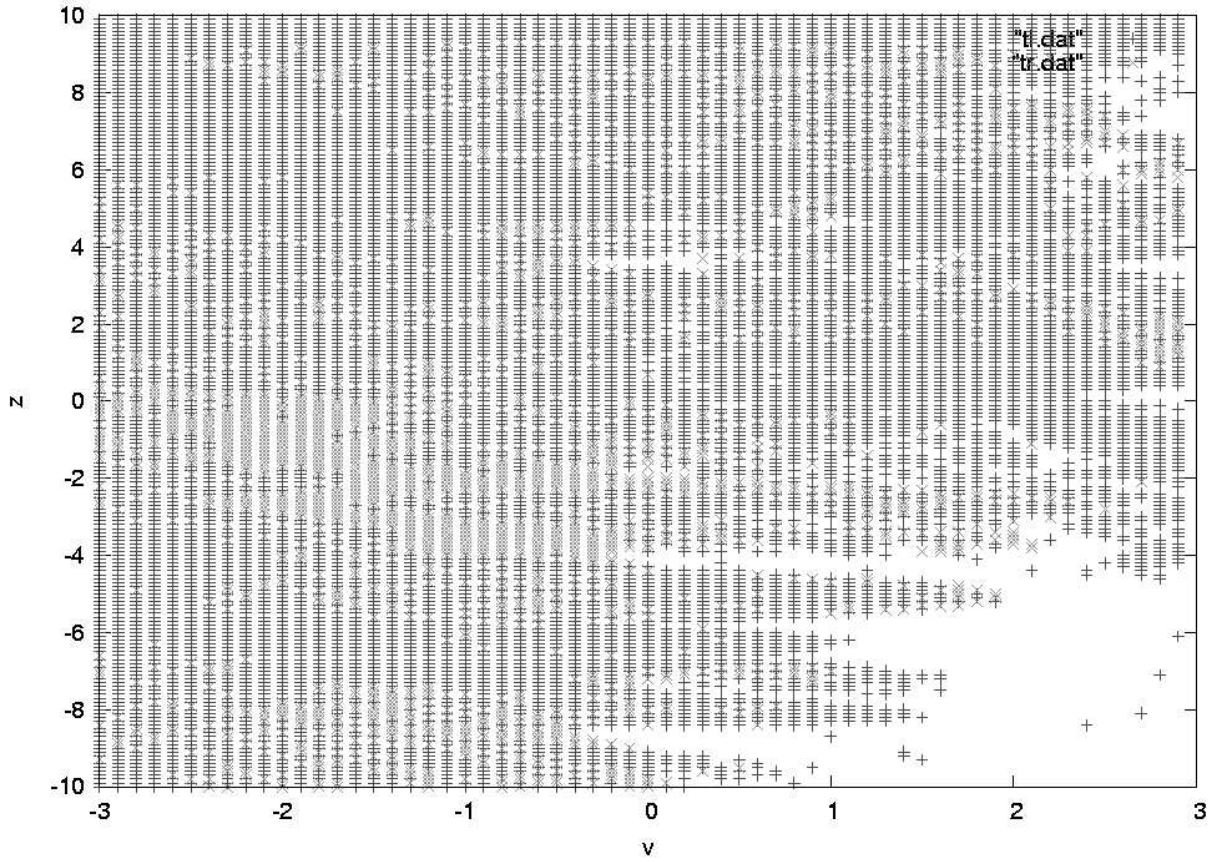

Figure 3.16: Plot of the final distribution of particles for the case of inhomogeneous plasma where the electrostatic wave is modeled as a sum of multiple airy functions. This is for the case of a very strong electric field. The vertical axis represents the x-range. The horizontal axis represents the v-range. The $\times$ dots represent the particles that went to the left but eventually turned around and ended up on the right. The + dots represent the particles that went to the right but eventually turned around and ended up on the left. Here, right represents the low density side. It must also be noted that the dots also include those particles that turned around more than once. This plot is too cluttered to make any sense out of it. This is a highly nonlinear problem and is out of the scope of the present work.

3.3 Wave-particle interaction

So far, we have not considered the modification of the electrostatic field itself due to modification in the density profile. In this section, we consider the linear effects of the perturbation in density on the first order electrostatic field, which is a single airy function to begin with. Also, in the previous two sections, we only considered the paths of the particles. We did not really estimate the modified density profile. But in this case, it is mandatory to calculate the modified density profile because this is the input to calculate the new electric field.

Thus, it is mandatory to find a mechanism that would sustain the gradient in the density profile in the absence of the airy field. We do this by introducing an imbalance in the flux of particles entering the plasma on either extremes, a nonuniform zero order electric field, E0, and collisions.

The imbalance in the flux is created in the following way. As soon as a particle leaves the system at $z1$, which is the leftmost extreme of the plasma, we introduce a new particle at $z2$, the rightmost extreme but with a weight, *Aminus*. And similarly when a particle leaves the system at $z2$, we introduce a new particle at $z1$, with weight, *Aplus*. And, $Aplus > Aminus$. These weights multiply with the Maxwellian to give the actual number of particles that each orbit represents. Thus we have more particles entering $z1$ compared to $z2$.

$E_0(z)$ is derived from the momentum equation. The derivation is as follows.

The zero order quantities are:

$$n_0 = n_0(z) \qquad \text{Density} \tag{3.21}$$

$$v_0 = 0 \qquad \text{Velocity} \tag{3.22}$$

$$E_0 = E_0(z) \qquad \text{Electrostatic Field} \tag{3.23}$$

The momentum equation is,

$$m_e n \frac{\partial \vec{v}}{\partial t} + m_e n(\vec{v} \cdot \vec{\nabla})\vec{v} = nq_e\vec{E} + nq_e(\vec{v} \times \vec{B}) - \vec{\nabla}p \tag{3.24}$$

Retaining only the zero order quantities, we get,

$$-ne\vec{E} - \vec{\nabla}p = 0 \tag{3.25}$$

We know that,

$$\vec{\nabla}p = \gamma KT\frac{\partial n}{\partial z} \tag{3.26}$$

Our density gradient is linear. Thus,

$$n = n_0(1 - \frac{z}{L}) \tag{3.27}$$

where n_0 is the density at $z = 0$ and L is the scale length of our plasma.

Substituting these in the momentum equation, we get,

$$\frac{-eE_0}{m} = \frac{-v_e^2}{L - z} \tag{3.28}$$

where $v_e = \sqrt{\gamma KT/m}$ is the thermal velocity of the particles.

To model collisions we have used probabilistic swapping of the sign of particle velocities. This means that after calculation of the location of each particle in the phase space after each time instant, we generate a random number, pr (say), and if this number is lesser than p times the absolute value of the particle velocity, then we swap the sign of the velocity. The value of p is set by the user and this

controls the amount of collisions. A larger value of p implies more collisions. We have run our simulations for a value of $p = 0.01$.

It was seen that the gradient is maintained to a fairly large extent. There are spikes at points, but that does not cause much trouble.

Now, we add the airy field to our problem. The modified version of the airy equation for this case is,

$$\frac{\partial^2 E_1}{\partial z^2} + \frac{w_e^2 z}{v_e^2 L} E_1 = \frac{w_e^2 n_1}{v_e^2 n_0(0)} E_0 \tag{3.29}$$

To solve for E_1 we use the green's function technique (please refer page 54 for further discussion on this topic). The solutions of the homogeneous equation are the inverted versions of Ai and Bi, which are the standard airy functions. Thus,

$$E_1 = c\text{Ai(-z)}+\text{Bi(-z)} \int_{-\infty}^{z} \text{Ai(-z)} \Big[\frac{w_e^2 n_1}{v_e^2 n_0(0)} E_0(z)\Big] dz + \text{Ai(-z)} \int_{z}^{\infty} \text{Bi(-z)} \Big[\frac{w_e^2 n_1}{v_e^2 n_0(0)} E_0(z)\Big] dz \tag{3.30}$$

where c is an arbitrary constant. The choice of c depends on the amount of nonlinearity one wants to inculcate in the problem. As c goes on increasing the problem becomes more and more nonlinear. We are interested in a weakly nonlinear problem and we want the trapping velocity to be on the tail of the Maxwellian.

The simulation is run for 1000 iterations and the results are shown in the plots that follow. Since we have considered a very weak coupling of the wave and the density perturbation, the airy field does not get modified much except at the extreme left of the plasma as can be seen in Figure 3.17.

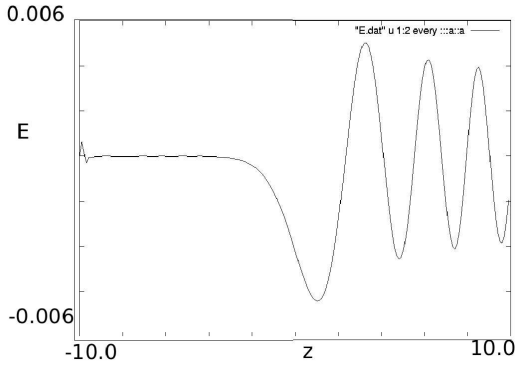

Figure 3.17: This is the 1st order electric field at t=100 for the case of wave-particle interaction. We can see that most of the field is unmodified. Only the first few values of the field are modified due to the density perturbation. This is mainly because the density distribution function itself maintains a gradient on an average even at t=100. If the density function itself shows wide variations from what it was at t=0, then the electric field will also get largely modified.

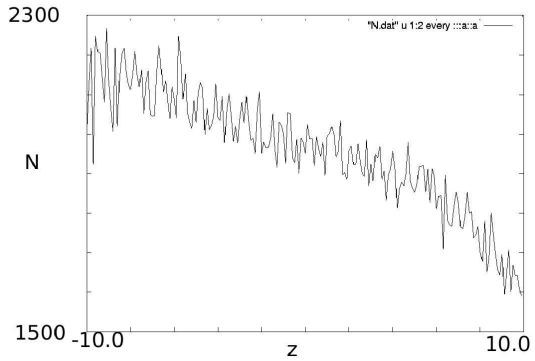

Figure 3.18: This is the density distribution function at t=100 for the case of wave-particle interaction. It can be seen that on an average we still have a linear gradient. Initially the density distribution does show variations from linearity but as time goes on the function settles down to structure shown in this graph.

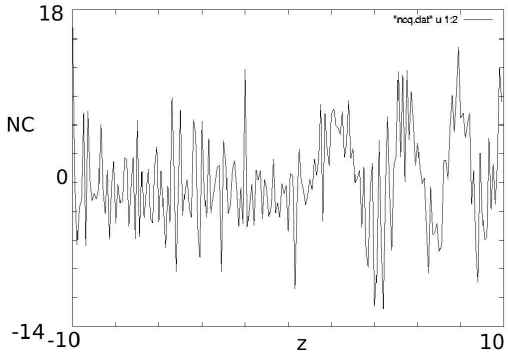

Figure 3.19: This shows the in-phase component of the density perturbation. $NC = \sum_{t=t1}^{t=t2} N(t)\cos(wt)$.

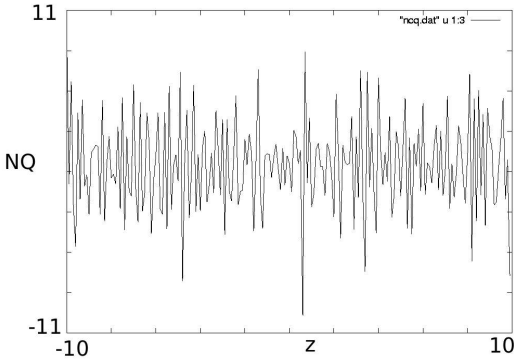

Figure 3.20: This shows the quadrature component of the density perturbation. $NQ = \sum_{t=t1}^{t=t2} N(t)\sin(wt)$.

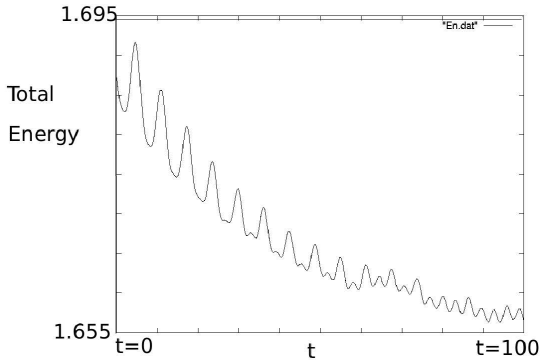

Figure 3.21: This plots the total energy of all the particles in the system as a function of time for the case of wave-particle interaction. We can see that total energy of the system is decreases by around 2.3% from its initial value of 1.695 and then stablizes to a value of around 1.655. The values shown are scaled values and should be taken in the relative sense.

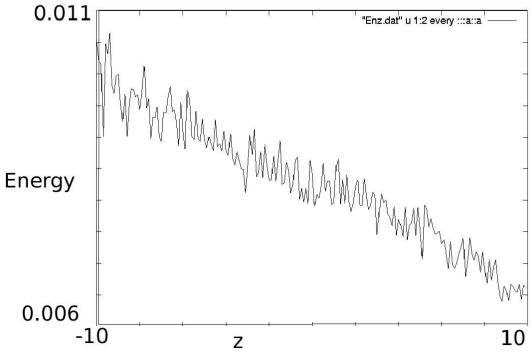

Figure 3.22: This is the plot of the energy of the system as a function of z at t=100 for the case of wave-particle interaction. We can clearly see the gradient in the energy of the system.

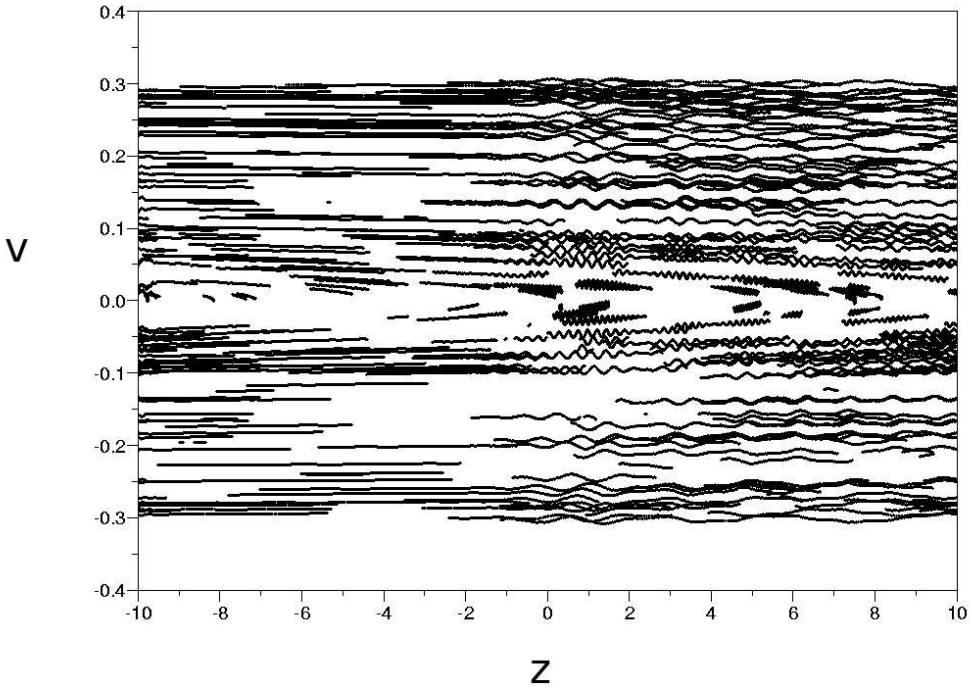

Figure 3.23: Plot of the paths of particles in the lab frame selected from all over the velocity range.

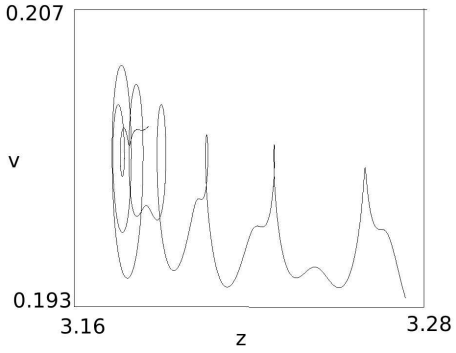

Figure 3.24: Plot of the path of a particle in the wave frame. This particle is initially free, then gets trapped and at the end gets free again.

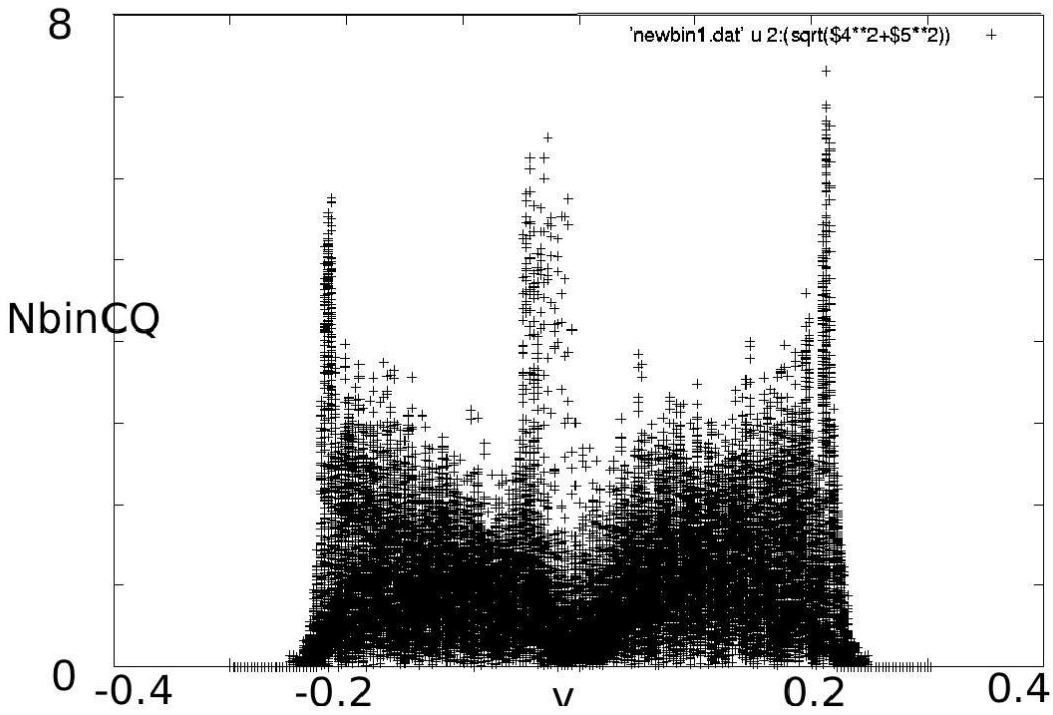

Figure 3.25: This plot shows the relative distribution of particles in various velocity bins. This is the magnitude of the perturbation and the underlying Gaussian has been eliminated.
$NC = \sum_{t=t1}^{t=t2} N(t)\cos(wt)$
$NQ = \sum_{t=t1}^{t=t2} N(t)\sin(wt)$
$NbinCQ = \sqrt{NC^2 + NQ^2}$
We can clearly see peaks near $v = -0.2$ and $v = +0.2$. These velocities are the trapping velocities. We know that due to landau damping the number of particles slightly faster than the phase velocity of the wave increases. It is this phenomenon that has been captured by the above plot. We can also see a peak near $v = 0.05$. This is because there is a lot of wave activity in that region and due to this even the particles with fairly low velocities are getting trapped.

The contour plot of the magnitude of the density perturbations in phase space is shown in figure 3.26 on page 36. The underlying Gaussian distribution function has been eliminated. It is this plot that gives us the maximum amount of information about our simulation. To understand the story that this plot tells, we have marked the interesting regions by labels, namely, 1,2,3,..., 11. Each of these labels corresponds to a region that has a characteristic of its own. Some of the labels can be grouped and seen together to understand yet another phenomenon.

Regions 1, 2, and 3 represent low velocity particles that have been trapped in the airy field. The velocities of these particles is below the thermal velocity, $ve = 0.07$. It is strange that these particles are getting trapped. We can attribute this trapping region to the nonlinear effects of the wave. If we magnify this plot, we will be able to see that these regions when put together closely follows a inverse square root curve in phase space. This is also expected because the resonance velocity is a function of the phase velocity of the electric field wave and that in turn is a function of the square root of the density gradient. Thus the resonance velocity goes like $\sqrt{1 - z/L}$, where L is the scale length of the density gradient of the plasma. And this is what is shown by these regions.

Regions 4 and 5 represent particles that are in the tail of our Gaussian distribution. We had chosen the magnitude of our airy field such as to get trapping in the tail region of the Maxwellian. Ideally this region should have been in between velocity range of $v = -1.9$ and $v = -2.1$. But instead we see this region slightly beyond $v = -2.1$. This, however, is nothing to be concerned about because the estimation of the trapping region involves a lot of approximations and slight deviations are understandable. We see some interesting contours immediately above regions 4 and 5. We shall discuss them later when we discuss regions 9, 10 and 11.

Regions 6, 7 and 8 represent the two most uninteresting areas in our contour plot. We do see some peaks in regions 6 and 8 but these are scattered around and show no structure. Region 7 shows no contours at all. In this region the initial density of particles is very high because this region corresponds to the peak of the Maxwellian. Also, the electric field wave in the region, $z < 0$, is very negligible. So, all that these particles see is the background zero order electric field, E_0. This electric field, $E_0(z)$, is such that it maintains the gradient. So, these particles do not move much because their velocities are also very low. And we have clipped off the very high values of density while generating this plot.

Regions 9, 10 and 11 again show some trapping phenomenon. We can see a certain symmetry between the contours of this region and that of regions 4 and 5. Even here the trapping occurs for velocities slightly higher than $v = 2.1$. Now let come to the contours immediately above regions 4 and 5. We see similar patterns below regions 9, 10 and 11. We also see that the number of these contours decreases with increasing z. These contours represent slightly trapped particles that are diffused over a larger velocity range. The range of velocities for which particles are trapped depends on the magnitude of the electric field wave. And since we have a airy field and not a sinusoidal field, the magnitude of the space dependent field keeps on decreasing with increasing z. Thus the velocity range of trapping is more near $z = 0$ and less at $z = 10$. And that is what we are seeing in our plot.

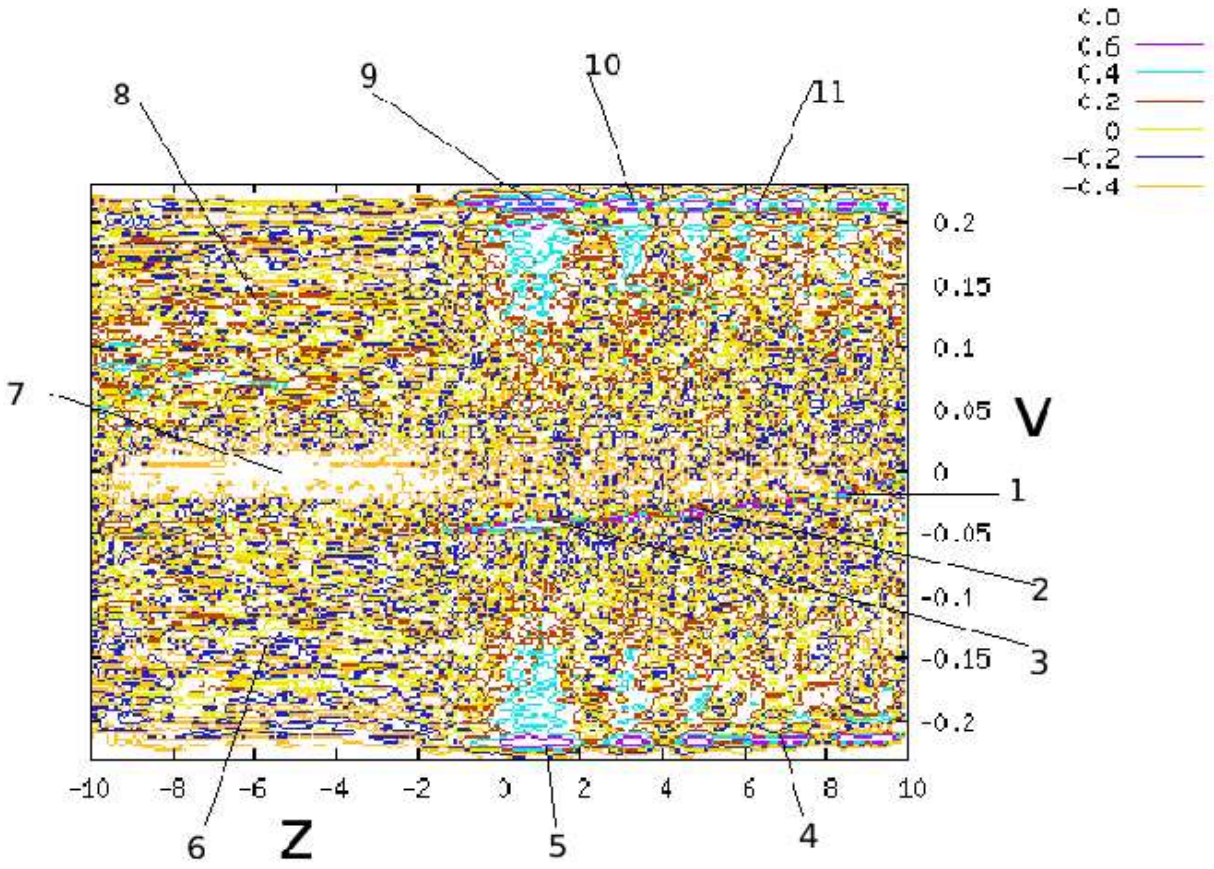

Figure 3.26: This is the contour plot of the magnitude of the density perturbations in phase space. The underlying Gaussian distribution function has been eliminated. It is this plot that gives us the maximum amount of information about our simulation. To understand the story that this plot tells, we have marked the interesting regions by labels, namely, 1,2,3,..., 11. Each of these labels corresponds to a region that has a characteristic of its own. Some of the labels can be grouped and seen together to understand yet another phenomenon.

Chapter 4

Conclusion

In this thesis we have studied the behavior of electrostatic waves in both homogeneous and inhomogeneous plasmas. We have also seen effects like Landau damping for both these cases. For a homogeneous plasma, the phenomenon of Landau damping is well understood. But it was interesting to find similar effects for a weakly inhomogeneous plasma. We could not really get a very clear understanding of the phenomenon but our effort certainly gave us some insights into the complex phenomenon.

The case of a noisy plasma is still obscure and a lot of work needs to be done in that area. We have no doubt explored some avenues but that was just scratching the surface.

The case of the wave particle interaction was the most intriguing of all. We brought in nonuniform zero order fields, collisions and unequal flux of particles on the boundaries of the plasma. We did see fingerprints of Landau damping for this case also.

Bibliography

[1] L.D. Landau, J. Phys. (U.S.S.R.) 10, 25 (1946).

[2] Dwight R. Nicholson, Introduction to Plasma Theory, John Wiley and Sons.

[3] G.J.Morales et al., Phys. Fluids B, Vol. 4, No. 3, March 1992.

[4] E. Kreyszig, Advanced Engineering Mathematics, 8th edition, page 108

Appendix A

dist.f

This Fortran 77 program is used to estimate the modified distribution function in the presence of a landau damped electrostatic wave in a homogeneous plasma.

```
      program dist
              integer i,j,N,Nx,Nv
```

N is the maximum value of Nx and Nv. Nx is the number of divisions of the co-orinate axis and Nv represents the number of divisions of velocity space.

```
              parameter(N=1000)
              real fold(N),v(N),vnew(N,N)
```

fold(N) represents the velocity dependent distribution function before the wave has set in. This fold(N) is taken to be Maxwellian in our simulation. v(N) represents the sampled points in velocity space at t=0. vnew(i,j) represents the final velocity of the particle in the ith space bin and jth velocity bin.

```
              real fnewplot(N),vnewplot(N)
```

fnewplot(N) represents the velocity dependent distribution function at t=tmax. vnewplot(N) represents the sampled points in velocity space at t=tmax.

```
              real tmax,E1,wi,k,vmin,vmax,vwave
```

tmax is the time till which we run the simulation. E1 is the magnitude of the first order sinusoidal electric field. wi is the imaginary part of the frequency of oscillations of the wave. The rate of damping is determined by wi. This is actually dependant on the slope of the velocity distribution function at the point where the particle velocity is equal to the phase velocity of the wave. But in this simulation we have allowed the user to choose the value of wi. k is the wave number of the wave. vmin and vmax are the minimum and maximum values of the intitial velocities of the particles. vwave is the phase velocity of the wave.

```
real vstep,vnewmin,vnewmax,vnewstep
```

vstep is the step-size in the intial velocity space. vnewmin and vnewmax are the minimum and maximum velocities at t=tmax. vnewstep is the step-size in the final velocity space.

```
external pathfinder
```

pathfinder is a subroutine used to find the final velocity of a charged particle after having travelled in a user defined electric field. This in turn uses standard routines odeint, rkck and rkqs from "Numerical Recipes in Fortran 77". These three routines have not been included in this code.

```
real xmin,xman,xstep,x(N)
```

xmin and xmax are the minimum and maximum values of the intitial positions of the particles. xstep is the step-size in the intial co-ordinate space. x(N) represents the sampled points in the co-ordinate space.

```
common /pathparameters/ E1,wi,k
```

These common parameters are used by the subroutine derivs

```
integer status,system

open(19,FILE="distparameters.dat",STATUS="OLD")
read(19,*) tmax,E1,wi,k,vmin,vmax,vwave,Nv,xmin,xmax,Nx
```

distparameters.dat is a text file containing the above mentioned values. These values can be written to this file by using a scilab code.

```
do i=1,Nv
        read(19,*) v(i)
        enddo
do i=1,Nx
        read(19,*) x(i)
        enddo
close(19)
```

The above two loops are for reading the sample points in velocity and co-ordinate space from distparameters.dat.

```
do i=1,Nv
        fold(i)=exp(-0.5*((v(i)+vwave)**2))
```

The above intitializes our velocity dependent distribution function to a gaussian curve.

```
do j=1,Nx
        call pathfinder(tmax,v(i),x(j),vnew(i,j))
        enddo
enddo
```

pathfinder takes as input values tmax, v(i) and x(j) and returns vnew(i,j), which is the final velocity of a particle at time tmax that started its trajectory at the point x(j) with an initial velocity of v(i).

```
vnewmin=vnew(1,1)
vnewmax=vnewmin
do i=1,Nv
        do j=1,Nx
                if(vnew(i,j).gt.vnewmax) then
                        vnewmax=vnew(i,j)
                elseif(vnew(i,j).lt.vnewmin) then
                        vnewmin=vnew(i,j)
                        endif
                enddo
        enddo
```

The above two loops are used to find vnewmin and vnewmax. This step is not really useful when the velocity range is large. But when we are considering a small region near the phase velocity of the wave, this can improve accuracy.

```
vnewstep=(vnewmax-vnewmin)/Nv

do i=1,Nv
        vnewplot(i)=vnewmin+i*vnewstep
        fnewplot(i)=0
        enddo
```

vnewplot(*) stores the sample points in the new velocity space, i.e. at t=tmax. fnewplot(*) is the modified distribution function. Here it is initialized to zero.

```
do i=1,Nv
        do j=1,Nx
                k=(vnew(i,j)-vnewmin)/vnewstep+1
```

k represents the velocity bin to which the current particle belongs.

```
                                fnewplot(k)=fnewplot(k)+fold(i)
```

After having found 'k', the number of particles in the kth bin are incremented by the number of particles that were in the ith bin at t=0.

```
                                enddo
                        enddo
```

We have come to the end of our main program. Now, we just write out the evaluated values to dist.dat which in turn is used by dist.gnu to plot the results.

```
                open(19,FILE="dist.dat",STATUS="UNKNOWN")
                do i=1,Nv
                        write(19,*) v(i),Nx*fold(i)
                        enddo
                write(19,*) ''
                do i=1,Nv
                        write(19,*) vnewplot(i),fnewplot(i)
                        enddo
                write(19,*) ''
                do i=2,Nv
                        write(19,*) v(i),vnew(i,1)
                        enddo
                close(19)

                status=system("gnuplot<dist.gnu")
                status=system("gv dist.ps")

                end

      subroutine pathfinder(tmax,vo,xo,vfinal)
C     driver for routine odeint
              INTEGER KMAXX,NMAX,NVAR
              PARAMETER (KMAXX=500,NMAX=50,NVAR=2)
              INTEGER i,kmax,kount,nbad,nok
              REAL dxsav,eps,h1,hmin,x1,x2,x,y,ystart(NVAR)
              COMMON /path/ kmax,kount,dxsav,x(KMAXX),y(NMAX,KMAXX)
              EXTERNAL derivs,rkqs
```

As said earlier, pathfinder is a routine that takes in values tmax, vo and xo and returns vfinal. It uses adaptive stepsize Runge-Kutta method of integration of ordinary differential equations. For storage of intermediate results we have

declared the common block path. KMAXX is the size of the array that is used to store intermediate results. NMAX is the maximum number of variables in the system of ordinary differential equations. NVAR is the actual number of variables. In our case these two variables are co-ordinate space and velocity. kmax is the maximum number of intermediate results stored. This can be set to KMAXX. kount is the actual number of stored results. nok and nbad is the number of good and bad steps taken respectively. dxsav is minimum interval above which the values are stored. eps is the desired accuracy. h1 is the guessed first step size. hmin is the minimum allowed stepsize. x1 and x2 are the starting and ending points in time. x stores the intermediate values of time. y(1,) and y(2,*) store the intermediate values of velocity and co-ordiante space respectively. ystart(1) and ystart(2) are the initial values of velocity and position of a particle respectively. derivs is the routine that defines the system of differential equations. rkqs is a standard routine used by odeint and can be obtained from "Numerical Recipes in Fortran 77".*

```
            integer status,system
            real tmax,vo,xo,vfinal

            x1=0
            x2=tmax
            ystart(1)=vo
            ystart(2)=xo
            eps=1.0e-4
            h1=.1
            hmin=0.0
            kmax=KMAXX
            dxsav=(x2-x1)/KMAXX
            call odeint(ystart,NVAR,x1,x2,eps,h1,hmin,
     *                  nok,nbad,derivs,rkqs)
            vfinal=y(1,kount)
      end

      subroutine derivs(x,y,dydx)
            real x,y(*),dydx(*)
            real e,E1,me,wi,k
            parameter(e=1.6e-19,me=9.1e-31)
            common /pathparameters/ E1,wi,k
            dydx(1)=-(e*E1*exp(-wi*x)/me)*sin(k*y(2))
            dydx(2)=y(1)
```

y(1) is velocity and y(2) is position. dydx(1) represents acceleration of the electron due to the electric field. dydx(2) represents y(1), i.e. velocity. In this simulation, the electric field is a sinusoidal wave with an exponentially decaying amplitude. The values of the E1, wave number, k, and rate of damping, wi, are read from a file, distparameters.dat in the main program and are made available to this subroutine through a common block statement. Since the simulation is done in the frame in which the wave is at rest, we do not have to specfy the frequency of oscillation of the wave.

```
            return
      end
```

Appendix B

path.ai.f

This Fortran 77 program is used to simulate the path of particles for the case of an inhomogeneous plasma where the electric field is modeled as a single airy function.

```
      PROGRAM main
              integer TSTEP
              parameter(TSTEP=200)
```

TSTEP is the number of time steps from t1 to t2.

```
              integer KMAXX,NMAX,NVAR,Nx,Nv
              parameter (KMAXX=5000,NMAX=50,NVAR=2,Nx=100,Nv=100)
```

KMAXX is maximum value of TSTEP. NMAX is the maximum value of NVAR. NVAR is number of variables in our system of ordinary differential equations. Nx is the number of sample points in the co-ordinate space. Nv is the number of sample points in the velocity space.

```
              integer i,j,k
              real v1,v2,vstep
```

v1 is the minimum velocity of a particle at t=0. v2 is the maximum velocity of a particle at t=0. vstep is the step-size in the velocity space.

```
              real x1,x2,xstep,t,y,vstart(NVAR)
```

x1 is the minimum value of position of a particle at t=0. x2 is the maximum value of position of a particle at t=0. xstep is the step-size in the co-ordinate space. t stores the sample points on the time axis. y stores the position and velocity of a particle at all time instants. vstart stores the initial values of position and velocity of a particle.

```
real t1,t2
```

t1 represents the starting time of simulations. t2 represents the time till which we run our simulations.

```
common /path/ t(KMAXX),y(NMAX,KMAXX)
external derivs,rkdumb
```

derivs is the subroutine where we define our system of differential equations. rkdumb is a standard subroutines found in "Numerical Recipes in Fortran 77".

```
integer turnleft,turnright
```

turnleft and turnright are used to find out which particles are turning around in their paths.

```
integer status,system

open(19,FILE="path.dat",STATUS="UNKNOWN")
open(11,file="tl.dat",status="unknown")
open(12,file="tr.dat",status="unknown")
```

path.dat is used to store the position and velocity of all particles at all time instants. tl.dat and tr.dat contain the intial position and velocity of particles that turned left and right on their paths respectively. The particles that were found to the right of their intial positions at some point of time but ended up on the left side of the initial position are termed to have turned left and vice verca.

```
x1=0.0
x2=10.0
xstep=(x2-x1)/Nx

v1=-3.5
v2=3.5
vstep=(v2-v1)/Nv

t1=0.
t2=20.

do j=1,Nv
write(*,*) j
   do i=1,Nx
        turnleft=0
        turnright=0
```

```
                        vstart(1)=x1+(i-1)*xstep
                        vstart(2)=v1+(j-1)*vstep

                        call rkdumb(vstart,NVAR,t1,t2,TSTEP,derivs)
```

vstart(1) and vstart(2) are the initial position and velocity of the current particle. A call to rkdumb updates the value of y(,*) to the positions of velocities of the current particle for all time instants. This is then written to a file "path.dat" and is also used to find which particles turnaround.*

```
                        do k=1,TSTEP
                                write(19,*) y(1,k),y(2,k),t(k)
                                if(y(1,k).gt.y(1,k-1)) turnleft=1
                                if(y(1,k).lt.y(1,k-1)) turnright=1
                        enddo
```

The particle can be said to turnleft iff at some point of time it moved to the right but eventually ended up on the left of its starting point. Similarly, for turnright. By setting turnleft or turnright to 1, we are marking those particles that qualify to be categorized as turnleft or turnright respectively. If for a particle, turnleft is equal to 1, means that this particle was found on the right of its intial position at some point of time. Below, we check if at the end of our simulation time, i.e. after TSTEP iterations, if the particle was found to the left of its initial position, and if turnleft is equal to 1, then we write the initial position and velocity of this particle to tl.dat. Similarly, for particles that are termed to have turned right.

```
                        if(y(1,TSTEP).lt.y(1,1).and.
     *                                  (turnleft.eq.1)) then
                                        write(11,*) vstart(2),vstart(1)
                        elseif(y(1,TSTEP).gt.y(1,1).and.
     *                                  (turnright.eq.1)) then
                                        write(12,*) vstart(2),vstart(1)
                        endif
                        write(19,*) ''
                   enddo
                enddo
                close(19)
        end

       subroutine derivs(t,y,dydt)
               real t,y(*),dydt(*)
```

t represents time. y(1) represents position. y(2) represents velocity.

```
      real ai,bi,aip,bip
```

ai is one of the values returned by the airy function. This is the value that is used further. bi, aip and bip are also the values returned by the airy function but these values are never used any further.

```
      external airy
```

airy(z,ai,bi,aip,bip) is a standard subroutine used to calculate the airy function. This can be found in "Numerical Recipes in Fortran 77".

```
      call airy(-y(1),ai,bi,aip,bip)
      dydt(1)=y(2)
      dydt(2)=-0.4*ai*cos(t)
      return
end
```

Appendix C

path.va.f

This Fortran 77 program is used to simulate the path of particles for the case of an inhomogeneous noisy plasma where the electric field is modeled as the sum of multiple shifted and random phased airy functions.

```
      program main
        integer KMAXX,NMAX,NVAR,Nx,Nv,Nai
        parameter (KMAXX=200,NMAX=50,NVAR=2,Nx=200,Nv=100,Nai=1000)
```

Nai is the number of sample points of the electric field. In this case, the electric field is modelled as the sum of shifted and random phased multiple airy functions.

```
        integer NSTEP
        parameter(NSTEP=100)
        real vstart(NVAR)
        integer i,j,k
        real x1,x2,xstep,v1,v2,vstep
        real t1,t2,t,y
        common /path/ t(KMAXX),y(NMAX,KMAXX)
        external rkdumb,rk4,derivs,ran0,Efield
        real ran0,Efield
```

ran0 is a standard routine used to generate random numbers. Efield is a function that returns the electric field at a given point.

```
        integer status,system
        real PI
        parameter(PI=3.1415927)
        real aiphase(Nai)
```

aiphase(*) stores the phase information of the various random phased airy functions.

```
      integer idum
      real ai(Nai,Nai),xaimin,xaimax,xaistep,xai(Nai)
```

ai(,*) stores the values of the various shifted airy functions. xaimin and xaimax represent the endpoints of our co-ordinate space.*

```
      real waimin,waimax,waistep,wai(Nai)
```

wai(*) stores the frequency of oscillations of the various airy functions. The point of reflection of an airy wave depends on its frequency.

```
      common /aiparameters/ aiphase,ai,xaimin,
     *                  xaimax,xaistep,xai,wai
```

The above common parameters are used by the function Efield to calcute the electric field at any given point.

```
      integer turnleft,turnright

      idum=-1

      t1=0.
      t2=20.

      x1=-10.0
      x2=10.0
      xstep=(x2-x1)/Nx

      v1=-10.0
      v2=10.0
      vstep=(v2-v1)/Nv

      xaimin=-50.
      xaimax=50.
      xaistep=(xaimax-xaimin)/Nai

      waimin=0.5
      waimax=1.5
      waistep=(waimax-waimin)/Nai

      do i=1,Nai
            aiphase(i)=2.0*PI*ran0(idum)
      enddo
```

```
        open(19,file="sairy.dat",status="old")
```

sairy.dat is a text file containing information about the values of the various shifted airy functions.

```
        do i=1,Nai
                do j=1,Nai
                        read(19,*) xai(i),ai(i,j)
                        enddo
                enddo

        do i=1,Nai
                xai(i)=xaimin+(i-1)*xaistep
                wai(i)=sqrt(waimin+(i-1)*waistep)
        enddo

        open(19,file="path.dat",status="unknown")
        open(11,file="tl.dat",status="unknown")
        open(12,file="tr.dat",status="unknown")

        do j=1,Nv
                do i=1,Nx
                        turnleft=0
                        turnright=0
                        vstart(1)=x1+(i-1)*xstep
                        vstart(2)=v1+(j-1)*vstep

                        call rkdumb(vstart,NVAR,t1,t2,NSTEP,derivs)

                        do k=1,NSTEP
                                write(19,*) y(1,k),y(2,k),t(k)
                                if(y(1,k).gt.y(1,1)) turnleft=1
                                if(y(1,k).lt.y(1,1)) turnright=1
                        enddo
                        write(19,*) ''

                        if((y(1,NSTEP).lt.y(1,1)).and.
     *                          (turnleft.eq.1)) then
                                write(11,*) vstart(2),vstart(1)
                        elseif((y(1,NSTEP).gt.y(1,1)).and.
```

```
*                           (turnright.eq.1)) then
                            write(12,*) vstart(2),vstart(1)
                    endif
            enddo
   enddo
   close(19)
   end

   subroutine derivs(t,y,dydt)
         real t,y(*),dydt(*)
         real Efield
         external Efield
         dydt(1)=y(2)
         dydt(2)=-Efield(t,y(1))
         return
   end

   function Efield(t,x)
           integer Nai,Npol
           parameter(Nai=1000,Npol=4)
```

Npol is the number of points used for doing the polynomial interpolation. We have the electric field tabulated at known points in space. To calculate the electric field at intermediate points, we do polynomial interpolation.

```
           real t,x,Efield
           real aiphase(Nai),wai(Nai)
           integer i,j
           real xaipos,xaipol(Npol)
           real ai(Nai,Nai),aipol(Npol)
```

xaipol(*) and aipol(*) store the values of the co-ordinate space and the field that are used for getting the interpolated value of the field at the required point.

```
           real xaimin,xaimax,xaistep,xai(Nai)
           external polint
           real err
```

polint is a standard routine for doing polynomial interpolation. err is the possible error in the interpolated value.

```
           common /aiparameters/ aiphase,ai,xaimin,
*                   xaimax,xaistep,xai,wai
```

```
                xaipos=int((x-xaimin)/xaistep)+1
```

xaipos represents the point at which we know the electric field and that is closest to the given point.

```
                if(xaipos.lt.Npol/2) then
                        Efield=0
                        return
                endif
                if(xaipos.gt.(Nai-Npol/2)) then
                        Efield=0
                        return
                endif
```

We do not need to bother about points that are on the extreme end of our co-ordinate space. So, we assign them an electric field of 0.

```
                do i=1,Npol
                        xaipol(i)=xai(xaipos+i-Npol/2)
                        enddo
                do i=1,Npol
                        aipol(i)=0
                        do j=1,Nai
                                aipol(i)=aipol(i)+ai(j,xaipos+i-Npol/2)*
    *                                   cos(wai(j)*t+aiphase(j))
                        enddo
                enddo
                call polint(xaipol,aipol,Npol,x,Efield,err)
                return
           end
```

Appendix D

Green's function technique

Here we discuss the green's function technique used in getting at Equation 3.30 on page 28. This technique applies to differential equations that have the general form,

$$y'' + p(z)y' + q(z)y = r(z) \tag{D.1}$$

with arbitrary variable functions p, q and r that are continuous on some interval I. A prime denotes derivate w.r.t. z. The method gives a particular solution, y_p of equation D.1 on I in the form,

$$y_p(z) = -y_1 \int_{c1}^{z} \frac{y_2 r}{W} dz + y_2 \int_{c2}^{z} \frac{y_1 r}{W} dz \tag{D.2}$$

where y_1, y_2 form a basis of solutions of the homogeneous equation,

$$y'' + p(z)y' + q(z)y = 0 \tag{D.3}$$

c1 and c2 are arbitrary constants and W is the Wronskian, given by,

$$W = y_1 y_2' - y_2 y_1' \tag{D.4}$$

It must however be noted that before applying this technique one should make sure that his/her equation is written in the standard format of equation D.1, with y'' as the first term; divide by $f(z)$ if it starts with $f(z)y''$.

For a proof of this technique please refer to the source [4] from which we have taken this method .

The equation that we are considering is equation 3.29 on page 28,

$$\frac{\partial^2 E_1}{\partial z^2} + \frac{w_e^2 z}{v_e^2 L} E_1 = \frac{w_e^2 n_1}{v_e^2 n_0(0)} E_0(z) \tag{D.5}$$

We can write this in the standard form of equation D.1 as,

$$y'' + zy' = r(z) \tag{D.6}$$

where the units have been normalized such that $w_e^2 \big/ v_e^2 L = 1$ and

$$r(z) = \frac{w_e^2 n_1}{v_e^2 n_0(0)} E_0(z) \tag{D.7}$$

We know that the homogeneous equation,

$$y'' + zy' = 0 \tag{D.8}$$

has two solutions, Ai(-z) and Bi(-z) namely, where Ai(z) and Bi(z) are solutions of the equation,

$$y'' - zy' = 0 \tag{D.9}$$

and are plotted in Figures D.1 and D.2 respectively.
The Wronskian of Ai and Bi is,

$$W = \frac{1}{\pi} \tag{D.10}$$

Now, we have two make a choice of the limits of the integration and that is the most important part of this problem.

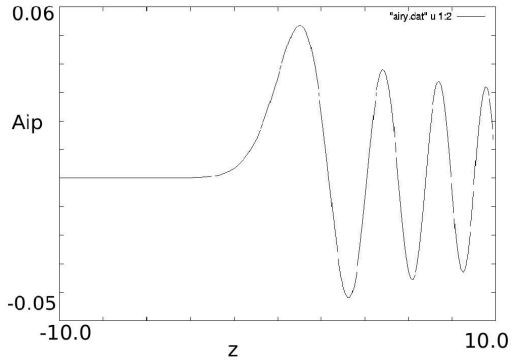

Figure D.1: The airy function, Ai(-z). The frequency of time oscillations, w, of this wave is equal to the plasma frequency, we, at z=0. Hence, the wave has a cut-off at this point.

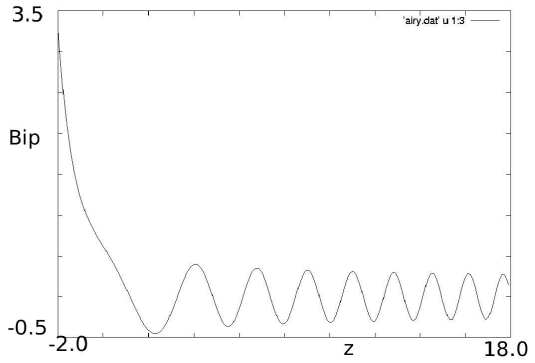

Figure D.2: The airy function, Bi(-z). The frequency of time oscillations, w, of this wave is equal to the plasma frequency, we, at z=0. Hence, the wave has a cut-off at this point.

We can see that Bi(-z) is monotonically increasing towards the left. We also need to keep in mind that our modified electric field should have contributions from the entire density function. Thus, we need to choose one among c1 and c2 as $-\infty$ and the other as $+\infty$. The constant associated with Bi(-z) has to be chosen to be $+\infty$ because otherwise, our electric field will blow up. Thus, the solution to equation D.6 is,

$$y_p = \pi B(z) \int_{-\infty}^{z} A(u)r(u)du + \pi A(z) \int_{z}^{\infty} B(u)r(u)du \tag{D.11}$$

where $A(z)$ and $B(z)$ are Ai(-z) and Bi(-z) respectively and $r(u)$ is given by equation D.7. But it must be noted that when $E_0 = 0$, $r(z) = 0$, and in that case our airy field will also go to zero. But we do not want that to happen. Our airy field is present even when there is no feedback from the density fluctuations. Thus we add the homogeneous solution, Ai(-z), to the above solution. Thus our required electric field is the one as given by equation 3.30.

$$E_1 = c\text{Ai(-z)}+\text{Bi(-z)} \int_{-\infty}^{z} \text{Ai(-z)}\Big[\frac{w_e^2 n_1}{v_e^2 n_0(0)} E_0(z)\Big]dz+\text{Ai(-z)} \int_{z}^{\infty} \text{Bi(-z)}\Big[\frac{w_e^2 n_1}{v_e^2 n_0(0)} E_0(z)\Big]dz \tag{D.12}$$

Appendix E

feedback.f

This Fortran 77 program is used to simulate the problem of wave particle interaction.

```
      program main
      integer Nz,Nv,Nzbin,Nvbin,MNp,Np,Nt,SNt,tn
```

Nz is the number of divisions in the co-ordinate space. Nv is the number of divisions in velocity space. Nzbin × Nvbin is the number of baskets in the phase space used for statistical purposes. MNp is the maximum number of particles for which we calculate the orbits. Np is the actual number of particles. Nt is the number of divisions on the time scale SNt determines the relative scale length of the calculations of modification in electric field. We might be interested in situations when the electric field varies at a much slower time scale than the density perturbation. SNt=1 implies that the electic field varies as quickly as the density perturbation. tn is the number of steps taken for the calculation of the orbits in each time step.

```
      integer KMAXX,NMAX,NVAR,N
```

KMAXX is the maximum number of intermediate results of integration that are stored. NMAX is the maximum number of variables in our system of ordinary differential equations. NVAR is the actual number of variables in our system, namely, v and z. N is the maximum possible values of Nv, Nz and Nt.

```
      parameter (KMAXX=500,NMAX=20,NVAR=2,N=2000)
      parameter (MNp=1000000)
      real V0(MNp),Z0(MNp),Zp(MNp),V(MNp)
```

V0(*) stores the initial velocity of all the particles. V(*) stores the updated velocity of all the particles at each time instant. This value is then used in the next iteration. Z0(*) stores the initial position of all the particles. Zp(*) stores the updated position of all the particles at each time instant. This value is then used in the next iteration.

```
integer i,j,k,bz,bv,j0
```

These are just indices used in our simulation.

```
real z1,z2,zstep,z(N),v1,v2,vstep,ve,vstart(NVAR),L
```

z1 and z2 define the boundaries of space. zstep is the length of the smallest division in space. v1 and v2 are the minimum and maximum values of the initial velocities of the particles. vstep is the length of the smallest division in velocity space. z(*) stores the values of the sampled points in co-ordinate space. ve is the thermal velocity of the particles. vstart(*) stores the values of the position and velocity of the particles for which the new position and velocity at the next time instant is to be calculated. L is the scale length of the plasma.

```
real t1,t2,tstep,t(N),tnow,tnext
```

t1 and t2 are the starting and ending times of our simulation. tstep is the incremental step in time. t(*) stores the values of the time instants from t1 to t2. tnow represents the time for which we have finished calculating the particle orbits. tnext represents the next time instant.

```
integer N0(N),N1(N),totalgauss(N)
real N1C(N),N1Q(N)
```

NO(*) is the initial density gradient. This is linear for the current problem. N1(N) is the modified density function. This is the sum of the background gradient and the perturbation due to the electric field. totalgauss(N) is the actual number of particles corresponding to a particular position. The value stored in N0(i) for some i just represents the actual number of particles, i.e. totalgauss(i). N1C(*) and N1Q(*) are the inphase and quadrature components of the density perturbations. $N1C(i) = \sum_{t=t1}^{t=t2}(N1(i) - N0(i))cos(wt)$*, for all i.* $N1Q(i) = \sum_{t=t1}^{t=t2}(N1(i) - N0(i))sin(wt)$*, for all i. For our simulations we have chosen w to be 1.*

```
real energy(N),En(N),E(N)
```

energy(*) stores the total kinetic energy of all the particles at all time instants. En(i) stores the total kinetic energy of the particles present in between positions i and i+1 at a particular time instant. E(*) stores the z-dependant electric field for all points in space at a particular time instant.

```
real zbinstep,vbinstep
integer N1bin(N,N)
real N1binC(N,N),N1binQ(N,N),Vbin(N,N),Enbin(N,N)
```

zbinstep is the step size of one bin in space. vbinstep is the step size of one bin in velocity space. N1bin(,*) is the number of particles in a particular bin at a particular time. N1binC(*,*) and N1binQ(*,*) are the inphase and quadrature components of N1bin(*,*). These are defined in a manner similar to N1C(*) and N1Q(*). Vbin(*,*) is the drift velosity of a bin at a particular time. En(*,*) is the total kinetic energy of all particles in a bin at a particular time.*

```
external ran0,derivs,rkdumb,distf
real ran0
integer distf
```

ran0 is a random number generator. derivs is a routine that is used to define our system of ordinary differential equations, i.e., $\dot{(z)} = v$ *and* $\dot{v} = a$*. It is this a that is the force due to the electric field. rkdumb is a standard routine used to solve a system of ordinary differential equations. We could have better codes but that would make the simulation extremely slow. distf is a function that is used to actual number of particles corresponding to a particular valus of initial velocity and position. ran0 and rkdumb are standard routines and can be obtained from "Numerical Recipes in Fortran 77".*

```
real ttemp(KMAXX),ytemp(NMAX,KMAXX)
```

ttemp(*) stores the intermediate values of time instants used by rkdumb to perform the integration. ytemp(1,) and ytemp(2,*) store the intermediate values of position and velocity as calculated by rkdumb.ytemp(1,tn) and ytemp(2,tn) are the final values.*

```
real fzv
real p,pr
integer idum
```

fzv is just a temporary variable and is used to reduce the number of function calls to the function distf. p and pr are used to model the collisions. p determines the rate of collisions. A higher p implies more collisions. pr is defined at the place where it is used in the code. idum is the seed value for the random number generator.

```
real ai(N),bi(N),aip(N),bip(N)
```

ai(*) and bi(*) are the inverted versions of the two airy functions Ai and Bi. aip(*) and bip(*) are their derivatives. In our simulation we have used only ai(*).

```
integer Npos
integer pathpos(MNp)
```

Npos is the number of points for which we want to keep track of their entire orbits in phase space. pathpos(*) stores the indicies of the orbits that we want to keep track of.

```
integer Afac(MNp),Aplus,Aminus
```

When a particle reaches one of the boundaries of our plasma, z1 or z2, we move it to the other extreme and make its velocity equal to its initial velocity. But due to the gradient in the plasma, for each particle that leaves the boundary, the number of particles entering at z1 will be more than the number of particles entering at z2. To model this we define an array Afac(*). Afac(i) stores the relative value of the number of actual particles corresponding to the particle with index i. Aplus and Aminus define the relative number of particles entering at the two end points in space.

```
real phi,E0field
external phi,E0field,Efield
```

phi and E0field are functions that return the zero order potential and electric field respectively.

$$\frac{-eE_0}{m} = \frac{-v_e^2}{L-z} \tag{E.1}$$

Efield is a routine that calculates the modified electric field on each iteration. This includes only the airy function and the green's function and not the zero order field.

$$\frac{\partial^2 E_1}{\partial z^2} + \frac{w_e^2 z}{v_e^2 L} E_1 = \frac{w_e^2 n_1}{v_e^2 n_0(0)} E_0 \tag{E.2}$$

```
common /path/ ttemp,ytemp
common /density/ N1,N0
common /calce/ E
common /airydat/ z,ai,bi,aip,bip
common /feedparameters/ z1,z2,zstep,ve,L,Nz,Nv

open(9,file="feedparameters.dat",status="old")
open(10,file="airy.dat",status="old")
open(11,file="zv.dat",status="unknown")
open(12,file="ncq.dat",status="unknown")
open(14,file="path.dat",status="unknown")
open(15,file="bin.dat",status="unknown")
open(17,file="Enz.dat",status="unknown")
open(18,file="En.dat",status="unknown")
open(19,file="N.dat",status="unknown")
open(20,file="E.dat",status="unknown")
```

feedparameters.dat contains the initial conditions and and various parameters for the simulation to run. airy.dat contains the values of the inverted airy function. zv.dat is the file to which can be used as a feedparameters.dat if we want to run the simulation again from the time at which we finished last time. ncq.dat stores the inphase and quadrature components of the density perturbations. path.dat stores the phase space orbits of Npos number of particles. bin.dat storess the values of N1bin(,*),N1binC(*,*),N1binQ(*,*),Vbin(*,*) and Enbin(*,*) at time t=t2 Enz.dat stores the values of En(*) at all time instants. En.dat stores the values of energy(*) at all time instants. N.dat stores the density profile at all time instants. E.dat contains the z-dependant modified airy electric field at all time instants.*

```
      read(9,*) p,Aplus,Aminus,z1,z2,Nz,Nzbin,v1,v2,Nv,Nvbin,ve,
     *          t1,t2,Nt,SNt,tn,L,Np,Npos
      do i=1,Np
               read(9,*) Z0(i),V0(i)
      enddo
      do i=1,Nz
              read(9,*) N0(i),totalgauss(i)
      enddo
      close(9)

      do i=1,Nz
              read(10,*) z(i),ai(i),bi(i),aip(i),bip(i)
              call Efield()
              write(20,*) z(i),E(i)
              write(19,*) z(i),N0(i)
      enddo
      close(10)
      write(20,*) ''
      write(19,*) ''
```

The above is used to read the values of the various parameters from the files feedparameters.dat and airy.dat.

```
      idum=1

      zstep=(z2-z1)/Nz
      zbinstep=(z2-z1)/Nzbin
      vbinstep=(v2-v1)/Nvbin
      tstep=(t2-t1)/Nt
```

```
do i=1,Np
        Zp(i)=Z0(i)
        V(i)=V0(i)
        if((V0(i)**2).lt.(2*phi(Z0(i),z2))) then
                Afac(i)=Aplus
        elseif(V0(i).lt.0) then
                Afac(i)=Aminus
        else
                Afac(i)=Aplus
        endif
enddo

do i=1,Nz
        N1(i)=0
        N1C(i)=0
        N1Q(i)=0
enddo

do i=1,Nzbin
        do j=1,Nvbin
                N1binC(i,j)=0
                N1binQ(i,j)=0
        enddo
enddo
```

The above loops initialize the values of the various variables.

```
do i=1,Npos
        pathpos(i)=int(Np*ran0(idum))+1
enddo
```

The above loop randomly selects Npos orbits that will be written to the file path.dat.

```
do k=1,Nt/SNt
        do i=1,SNt
                t((k-1)*SNt+i)=t1+((k-1)*SNt+i-1)*tstep
                do j=1,Np
                        vstart(1)=Zp(j)
```

```
                                vstart(2)=V(j)
                                tnow=t((k-1)*SNt+i)
                                tnext=tnow+tstep
                                call rkdumb(vstart,NVAR,tnow,
     *                          tnext,tn,derivs)
                                if(ytemp(1,tn).gt.z2) then
                                        Zp(j)=z1
                                        V(j)=abs(V0(j))
                                        Afac(j)=Aplus
                                elseif(ytemp(1,tn).lt.z1) then
                                        Zp(j)=z2
                                        V(j)=-abs(V0(j))
                                        Afac(j)=Aminus
                                else
                                        Zp(j)=ytemp(1,tn)
                                        V(j)=ytemp(2,tn)
                                endif
```

This loop calculates the orbits of all the particles at each time instant. The if statement above is used to handle those particles that move out of the boundaries of the plasma, z1 and z2.

```
                                pr=ran0(idum)
                                if(pr.lt.(p*abs(V(j)))) V(j)=-V(j)
```

The above two lines are used to model the collisions. The velocity of the jth particle is swapped with a probability of $p|V(j)|$.

```
                        enddo
                enddo

                do i=1,Npos
                        write(14,*) Zp(pathpos(i)),V(pathpos(i))
                enddo
                write(14,*) ''

                do i=1,Nz
                        N1(i)=0
                        En(i)=0
                enddo
                do i=1,Nzbin
```

```
                do j=1,Nvbin+2
                        N1bin(i,j)=0
                enddo
        enddo

        energy(k)=0

        do i=1,Np
                j=int((Zp(i)-z1)/zstep)+1
                j0=int((Z0(i)-z1)/zstep)+1
```

j maps the current location of a particle to the nearest sampled point in position. j0 maps the initial location of a particle to the nearest sampled point in position.

```
                fzv=distf(V0(i),Afac(i),N0(j0),totalgauss(j0))
```

fzv stores the value returned by the function distf. This is the actual number of particles represented by the orbit with index i. This number is then added to the jth bin to get the values of density and energy as a function of z.

```
                N1(j)=N1(j)+fzv
                En(j)=En(j)+(V(i)**2)*fzv
                energy(k)=energy(k)+(V(i)**2)*fzv
                bz=(Zp(i)-z1)/zbinstep+1
                bv=(V(i)-v1)/vbinstep+1
                if(bv.lt.1) then
                        bv=1
                elseif(bv.gt.(Nvbin)) then
                        bv=Nvbin
                endif
                N1bin(bz,bv)=N1bin(bz,bv)+fzv
```

bz and bv are the indices of a particular bin in phase space. N1bin(bz,bv) is then increased by the value fzv. Calculating N1bin gives us good insight into the relative distribution of particle density in phase space. We can then calculate the inphase and quadrature components of this quantity to see the wave features in density.

```
        enddo

        if((k.gt.300).and.(k.le.991)) then
                do i=1,Nzbin
                        do j=1,Nvbin+2
                                N1binC(i,j)=N1binC(i,j)+
```

```
*                                        float(N1bin(i,j))*cos(tnext)
                                        N1binQ(i,j)=N1binQ(i,j)+
*                                        float(N1bin(i,j))*sin(tnext)
                                enddo
                        enddo
                endif
```

N1binC and N1binQ are the inphase and quadrature components of N1bin, which in turn stores the density of particles in different bins in phase space.

```
                write(18,*) k,energy(k)
                write(*,*) 'energy=',k,energy(k)
                do i=1,Nz
                        N1(i)=int(float(N1(i))/1000)
                        if((k.gt.300).and.(k.le.991)) then
                                N1C(i)=N1C(i)+float(N1(i))*cos(tnext)
                                N1Q(i)=N1Q(i)+float(N1(i))*sin(tnext)
                        endif
                        write(19,*) z(i),N1(i)
                        write(17,*) z(i),En(i)
                enddo
                write(19,*) ''
                write(17,*) ''
```

N1C and N1Q are the inphase and quadrature components of the z-dependent density profile. These quantities also have wave like features. We have chosen 300 and 991 for two reasons. One is to remove the effects of early transients and the other is to remove truncation effects. We should average the quantities over an integer number of wavelengths.

```
                call Efield()
                do i=1,Nz
                        write(20,*) z1+(i-1)*zstep,E(i)
                enddo
                write(20,*) ''
        enddo

        vstep=(v2-v1)/Nv
```

```
      do i=1,Np
                        j0=int((Z0(i)-z1)/zstep)+1
                        bz=(Zp(i)-z1)/zbinstep+1
                        bv=(V(i)-v1)/vbinstep+2
                        if(bv.lt.1) then
                                bv=1
                        elseif(bv.gt.(Nvbin+2)) then
                                bv=Nvbin+2
                        endif
                        fzv=distf(V0(i),Afac(i),N0(j0),totalgauss(j0))
                        Vbin(bz,bv)=Vbin(bz,bv)+V(i)*fzv
                        Enbin(bz,bv)=Enbin(bz,bv)+(V(i)**2)*fzv
      enddo
```

Vbin(,*) stores the velocity of each bin in phase space. Similarly, Enbin(*,*) stores the total energy of the particles in each bin in phase space.*

```
      do i=1,Nzbin
              do j=1,Nvbin+2
                      write(15,*) z1+(i-1)*zbinstep,v1+(j-1)*vbinstep,
     *                        N1bin(i,j),N1binC(i,j),N1binQ(i,j),
     *                        Vbin(i,j),Enbin(i,j)
              enddo
              write(15,*) ''
      enddo

      do i=1,Nz
              write(12,*) z(i),N1C(i),N1Q(i)
      enddo
      write(11,*) p,Aplus,Aminus,z1,z2,Nz,Nzbin,v1,v2,Nv,Nvbin,ve,
     *         tnext,tnext+100.0,Nt,SNt,tn,L,Np,Npos
      do i=1,Np
              write(11,*) Zp(i),V(i)
      enddo
      do i=1,Nz
              write(11,*) N0(i),totalgauss(i)
      enddo
      end


      subroutine derivs(t,y,dydt)
```

derivs is the routine that defines our system of differential equations, namely, $dz/dt = v$ *and* $dv/dt = a$*, where a is the acceleration of electrons due the electric field. So, it basically defines a.*

```
integer N,Nz,pol
parameter (N=2000,pol=6)
```

N is the maximum allowed value of Nz and Nz is the number of sample points in the co-ordinate space. In our code, we recalculate the electric field at Nz points in each iteration in the main program. To get the field strength at intermediate points, we use polyomial interpolation. pol is the number of points we use to get the interpolated value of field strength at a particular point. This must be an even value for this code to give correct results.

```
real Epol(pol),Zpol(pol)
```

Epol(*) and Zpol(*) store the values of the field strength and z-coordinates of the points used for interpolation.

```
integer i,j
real t,y(*),dydt(*)
```

t represents time. y(1) represents z. dydt(1) and y(2) represent v. dydt(2) represents acceleration due to the electric field force.

```
real E(N)
```

E(*) stores the values of the electric field strength at Nz sample points in space.

```
real z1,z2,zstep,z
external polint,E0field
real E0field
```

polint is a standard routine used for doing polynomial interpolation. E0field is a routine that returns the value of the zero order electric field at a given point in space.

```
real Efz,err
```

Efz is the interpolated value of the first order field strength at a particular point in space. err is the error in the interpolation estimate.

```
common /feedparameters/ z1,z2,zstep,ve,L,Nz,Nv
common /calce/ E
z=y(1)
```

```
i=int((z-z1)/zstep)+1

if(i.lt.pol) then
        do j=1,pol
                Epol(j)=E(i+j)
                Zpol(j)=z1+(i+j)*zstep
        enddo
elseif(i.gt.(Nz-pol)) then
        do j=1,pol
                Epol(j)=E(i-j)
                Zpol(j)=z1+(i-j)*zstep
        enddo
else
        do j=1,pol
                Epol(j)=E(i-j+pol/2)
                Zpol(j)=z1+(i-j+pol/2-1)*zstep
        enddo
endif
call polint(Zpol,Epol,pol,z,Efz,err)
dydt(1)=y(2)
dydt(2)=E0field(y(1))+Efz*cos(t)
return
end

subroutine Efield()
integer N,Nz
parameter (N=2000)
real E(N)
integer i,j
external qromb,ain1,bin1
real ain1,bin1
real term1(N),term2(N),term3(N),term4(N),term5(N)
real term6(N),term7(N)
```

The first order electric field is given by the equation,

$$E_1 = c(iAi) + iBi \int_{-\infty}^{z} (iAi)\Big[\frac{w_e^2 n_1}{v_e^2 n_0(0)} E_0(z)\Big]dz + iAi \int_{z}^{\infty} (iBi)\Big[\frac{w_e^2 n_1}{v_e^2 n_0(0)} E_0(z)\Big]dz \tag{E.3}$$

qromb is used to do the integration. ain1 is a routine that returns the integrand of the 2nd term. bin1 is a routine that returns the integrand of the 3rd term. term1,

term2, till term7 are temporary variables used for storing intermediate values of various terms in the above equation. term3 is the value of the integration term in the 2nd term. term5 is term3 times the value of iBi. term4 is the value of the integration term in the 3rd term. term6 is term4 times the value of iAi. term7 is the sum of term4 and term6.

```
real ai(N),bi(N),aip(N),bip(N)
real z1,z2,zstep,z(N)
common /feedparameters/ z1,z2,zstep,ve,L,Nz,Nv
common /airydat/ z,ai,bi,aip,bip
common /calce/ E
zstep=(z2-z1)/Nz
z(1)=z1
term1(1)=0
do i=2,Nz
        call qromb(ain1,z(i-1),z(i),term1(i))
        term3(i)=term1(i-1)+term1(i)
enddo

term2(Nz)=0
do i=Nz-1,1,-1
        call qromb(bin1,z(i),z(i+1),term2(i))
        term4(i)=term2(i)+term2(i+1)
enddo
term3(1)=term1(1)
term4(Nz)=term2(Nz)
do i=1,Nz
        term5(i)=term3(i)*bi(i)
        term6(i)=term4(i)*ai(i)
        term7(i)=term5(i)+term6(i)
        E(i)=-0.012*ai(i)+term7(i)/(2000.0*(ve**2))
enddo
return
end

function ain1(z)
integer N,Nz
parameter (N=2000)
real z
real ain1,E0field
external airy,E0field
```

```
integer N1(N),N0(N)
integer pol
parameter (pol=6)
real E0
real Npol(pol)
real Nfin
real Zpol(pol)
common /density/ N1,N0
integer i,j
real temp
real z1,z2,zstep
real ai(N),bi(N),aip(N),bip(N),aipol(pol)
common /airydat/ zt(N),ai,bi,aip,bip
common /feedparameters/ z1,z2,zstep,ve,L,Nz,Nv

i=int((z-z1)/zstep)+1
if(i.lt.pol) then
        do j=1,pol
                aipol(j)=ai(i+j)
                Npol(j)=N1(i+j)-N0(i+j)
                Zpol(j)=z1+(i+j)*zstep
        enddo
elseif(i.gt.(200-pol)) then
        do j=1,pol
                aipol(j)=ai(i-j)
                Npol(j)=N1(i-j)-N0(i-j)
                Zpol(j)=z1+(i-j)*zstep
        enddo
else
        do j=1,pol
                aipol(j)=ai(i-j+pol/2)
                Npol(j)=N1(i-j+pol/2)-N0(i-j+pol/2)
                Zpol(j)=z1+(i-j+pol/2-1)*zstep
        enddo
endif

call polint(Zpol,Npol,pol,z,Nfin,temp)
call polint(Zpol,aipol,pol,z,ain1,temp)
ain1=ain1*Nfin*E0field(z)
return
end
```

```
function bin1(z)
integer N,Nz
parameter (N=2000)
real bin1
real z
real E0field
external airy,spline,splint,E0field
real sp1(2),sp2(2),sp3(2)
integer N1(N),N0(N)
integer pol
parameter (pol=6)
real E0
real Npol(pol)
real Nfin
real Zpol(pol)
common /density/ N1,N0
integer i,j
real temp
real z1,z2,zstep
real ai(N),bi(N),aip(N),bip(N),bipol(pol)
common /airydat/ zt(N),ai,bi,aip,bip
common /feedparameters/ z1,z2,zstep,ve,L,Nz,Nv

i=int((z-z1)/zstep)+1
if(i.lt.pol) then
        do j=1,pol
                bipol(j)=ai(i+j)
                Npol(j)=N1(i+j)-N0(i+j)
                Zpol(j)=z1+(i+j)*zstep
        enddo
elseif(i.gt.(Nz-pol)) then
        do j=1,pol
                bipol(j)=bi(i-j)
                Npol(j)=N1(i-j)-N0(i-j)
                Zpol(j)=z1+(i-j)*zstep
        enddo
else
        do j=1,pol
                bipol(j)=bi(i-j+pol/2)
                Npol(j)=N1(i-j+pol/2)-N0(i-j+pol/2)
```

```
                Zpol(j)=z1+(i-j+pol/2-1)*zstep
        enddo
endif
call polint(Zpol,Npol,pol,z,Nfin,temp)
call polint(Zpol,bipol,pol,z,bin1,temp)
bin1=bin1*Nfin*E0field(z)
return
end

function distf(v,c,n,tg)
real v
integer distf,dist
external dist
integer n,tg
integer c
real ve
common /feedparameters/ z1,z2,zstep,ve,L,Nz,Nv
distf=c*int(1e5*exp(-0.5*(v/ve)**2)*float(n)/float(tg))
return
end

function phi(zstart,zend)
real zstart,zend
real z1,z2,ve,L
real phi
common /feedparameters/ z1,z2,zstep,ve,L,Nz,Nv
phi=(ve**2)*log((L-zstart)/(L-zend))
return
end

function E0field(z)
real z,E0field
real z1,z2,ve,L
common /feedparameters/ z1,z2,zstep,ve,L,Nz,Nv
E0field=-(ve**2)/(L-z)
return
end
```

VITA

Name	:	Kushal Kumar Shah
Address	:	c/o- Maliram Baijnath Motiganj Bazar Balasore - 756003 Orissa Ph: (06782) 262601(R), 262137(O) Email: atmabodha@gmail.com
D.O.B	:	3rd November 1981
Father	:	Sri Kishan Shah
Mother	:	Srimati Asha Shah
1998	:	10th from St. Vincent's Convent School, Balasore, Orissa
2000	:	12th from St. James' School, Kolkata
2005	:	Bachelor of Technology in Electrical Engineering from IIT Madras