UiT
THE ARCTIC
UNIVERSITY
OF NORWAY

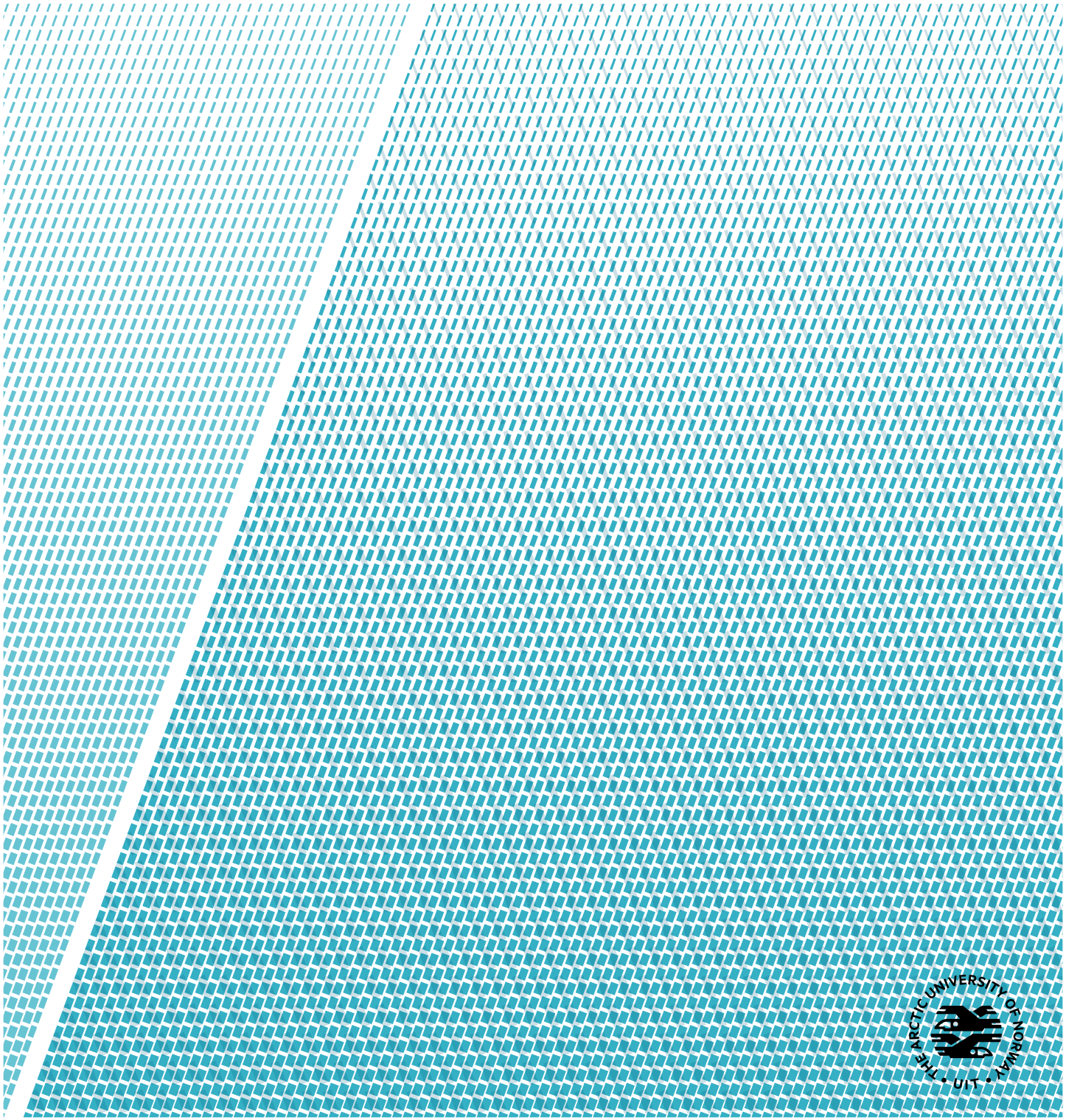
Faculty of Science and Technology
Department of Physics and Technology

A model for IS spectra for magnetized plasma with arbitrary isotropic velocity distributions

—

Eirik Rolland Enger

FYS-3931 Master's thesis in space physics 30 SP — June 2020



“You miss 100% of the shots you don’t take. –Wayne Gretzky”
–Michael Scott

Abstract

The plasma line in the incoherent scatter spectrum is known to provide information about the state of the ionosphere. However, it is weak in signal strength and therefore difficult to measure reliably and consistently. When high-energetic electrons (suprathermal electrons) are present in the ionosphere the plasma line echo power is enhanced and detectable by more radars. Recent measurements made by the Arecibo radar show an altitude and aspect angle (angle between the radar beam and the magnetic field line) dependence on the returned echo power of the plasma line. This was assumed to be due to enhancements in the suprathermal electron velocity distribution but has neither been confirmed through theory nor numerical analysis.

The theory describing the plasma line in the incoherent scatter spectrum due to scattering off thermal electrons has been known for a long time. This theory includes radar measurements at large angles to the magnetic field but a similar general derivation has not been formulated where suprathermal electrons are included in the distribution.

In this work a derivation of the dielectric function which is a fundamental part of the derivation of the incoherent scatter spectrum was carried out for an arbitrary isotropic velocity distribution. Further, a program calculating the spectrum using the derived dielectric function was developed. The program was used to model the incoherent scatter spectrum for different electron velocity distributions and the echo power in the plasma line as a function of aspect angle and electron number density. It was shown that the enhancements found in the suprathermal distribution map to the structures found in the plasma line echo power, in line with the proposed explanation based on measurements. These findings support an aspect angle formula relating energy and received plasma resonance frequency based on the assumption that the main contributing factor to the resonance frequency are the electrons with velocity close to parallel to the magnetic field line.

Acknowledgements

I would like to thank my supervisor Björn Gustavsson for introducing me to a very interesting topic that has been both a challenge and good fun, and for always keeping an open door, ready to answer questions.

Also a thank to Juha Vierinen for a lot of helpful discussions, and for giving me access to run my code on a proper computer, rather than my lousy Mac.

And finally the other masters students and people in the Space Physics group for making it easy to keep the spirits up.

Contents

List of Figures

List of Tables

List of Symbols

$\langle \cdot \rangle$ Ensemble average of $\cdot$

$\langle |n_\alpha|^2 \rangle$ Power density of n_α

Υ Auto-correlation function

$\mathfrak{F}\{\cdot\}$ Fourier transform of $\cdot$

$\mathfrak{F}^{-1}\{\cdot\}$ Inverse Fourier transform of $\cdot$

$\mathfrak{R}\{\cdot\}$ **and** $\mathfrak{I}\{\cdot\}$ Real and imaginary part of $\cdot$

ϵ Dielectric function

χ Susceptibility of a dielectric

f Phase-space density distribution function, frequency when specified

f_0 Velocity distribution function

f_r Radar frequency

$f_\mathfrak{R}$ Resonance frequency of an ion or plasma wave

g Gordeyev integral

n_α Number density of particle species α

ρ Charge density

α Electrons or ions ($\alpha = \mathrm{e}, \mathrm{i}$)

q Elementary charge

$\boldsymbol{r}$ Distance vector

$\boldsymbol{k}$ Wave vector

$\boldsymbol{E}$ Electric field vector

$\boldsymbol{B}$ Magnetic field vector

θ Aspect angle, angle between the magnetic field line and the radar wave vector

ω Angular frequency

$\omega_{\mathrm{p}\alpha}$ Angular plasma frequency of particle species α

Ω_α Angular gyro frequency of particle species α

ν_α Collision frequency of particle species α

$v_{\mathrm{th},\alpha}$ Thermal speed of particle species α

C_s Ion sound speed

λ_D Debye length

E Energy (note the difference from the electric field vector, which is in boldface)

m_α Mass of particle species α

T_α Temperature of particle species α

k_B Boltzmann's constant

ε_0 Vacuum permittivity

N_f Number of samples along the frequency axis

N_y Number of samples in the Gordeyev integral variable y (e.g. eq. (2.53))

N_v Number of samples in the velocity integral variable v (eq. (4.4))

List of Abbreviations

ACF auto-correlation function

FFT fast Fourier transform

IGRF International Geomagnetic Reference Field

IS incoherent scatter

UV ultraviolet

1

Introduction

The term incoherent scatter (IS) describes the process where a radio wave is scattered off numerous rapidly varying structures meeting the Bragg condition in the ionosphere. Using this technique, one can extract information about both the ion and electron composition over a wide range of altitudes in the ionosphere. These structures are typically thermally excited and move as damped waves. Since the propagation depend on the physical properties of the plasma (e.g. density and temperature) the backscattered signal will also contain information about these properties. Ionospheric parameters are obtained through fitting a power density spectrum, a model based on the theory describing IS, to the received signal. The power density spectrum, which model electrons and ions, may be derived from analysing electromagnetic waves scattering off ions and electrons using the Boltzmann equation, as was done by Hagfors (1961). In a plasma, structures move as waves, typically ion acoustic waves and plasma waves for plasmas in near thermodynamic equilibrium. Therefore, the power density spectrum—from here onward referred to as the IS spectrum—can generally be split into two parts, the ion line and the plasma line, depending on the radar wavelength that is used (Yngvesson and Perkins, 1968). A third line known as the gyro line can also be found for scattering at an angle to the magnetic field (Salpeter, 1961; Bjørnå et al., 1990) with intensity that is strongly dependent on the angle between the radar wave vector and the magnetic field (Salpeter, 1961).

The plasma line in the IS spectrum is the result of scattering off high frequency electron waves, and specifically it is the result of the electrons being discrete

particles (Yngvesson and Perkins, 1968). If a plasma is perturbed, say by the introduction of an ion, electric fields are set up so that neutrality can again be restored. It is the light electrons that flow along the electric field lines to restore neutrality, but with the gained momentum they overshoot to set up another electric field, similar to the perturbed state. This motion is recognized as electron plasma oscillations or Langmuir oscillations (Bittencourt, 2004), and the associated frequency is so high that the heavier ions are not able to follow. The angular frequency associated with the oscillation is known as the electron plasma frequency, denoted ω_{pe}. When thermal motion and the pressure gradient are taken into account propagating waves known as electron plasma waves or Langmuir waves arises (Bittencourt, 2004). In plasma oscillations, all electrons move together as a whole, but with thermal motion the phase and group velocities become functions varying in space and depends on both number density and temperature. The two additional effects give a plasma wave frequency of (Perkins and Salpeter, 1965; Showen, 1979; Nicolls et al., 2006)

$$\omega_{\mathfrak{R},\mathrm{e}} := \mathfrak{R}\{\omega_{\mathrm{e}}\} = (\omega_{\mathrm{pe}}^2 + 3k^2 v_{\mathrm{th,e}}^2 + \Omega_{\mathrm{e}}^2 \sin^2\theta)^{1/2}, \tag{1.1}$$

where k is the wave number, $v_{\mathrm{th,e}}$ is the thermal velocity, Ω_{e} is the electron gyro frequency and θ is the angle between the radar wave vector and the magnetic field line known as the aspect angle. ω_{e} is the complex angular plasma wave frequency derived in kinetic theory. In addition, the wave vector is in general not the same for the up- and downshifted plasma lines but given through the mean of the incident and scattered wave frequencies as (Showen, 1979)

$$k_{\pm} = \frac{1}{c}[\omega_{\mathrm{r}} + (\omega_{\mathrm{r}} \pm \omega_{\mathfrak{R},\mathrm{e}})] \tag{1.2}$$

where $\pm$ is for the up- and downshifted waves, c is the speed of light and ω_{r} is the angular radar frequency used to probe the ionosphere. Equation (1.1) states that the wave frequency is higher than the plasma frequency, usually in the MHz range. In the IS spectrum the plasma waves are found at frequencies $\omega_{\mathrm{r}} \pm \omega_{\mathfrak{R},\mathrm{e}}$, where $\omega_{\mathfrak{R},\mathrm{e}} \approx \omega_{\mathrm{pe}} \ll \omega_{\mathrm{r}}$ (Yngvesson and Perkins, 1968).

The ion line in the IS spectrum is the result of ion motion. The heavier ions do not respond to the high frequency of electron plasma waves, but rather in response to waves with frequencies on the order of kHz. Such waves are known as ion acoustic waves and the frequency of these waves can be found through considering longitudinal frequency oscillations. A frequency of

$$\omega_{\mathfrak{R},\mathrm{i}} := \mathfrak{R}\{\omega_{\mathrm{i}}\} = kC_{\mathrm{s}} \tag{1.3}$$

is then obtained, where C_{s} is the ion sound speed (Chen, 1984).

A feature of a plasma is Debye shielding due to the electric fields that develop between the charged particles (Bittencourt, 2004). Connected to Debye shielding is the idea of a Debye sphere, referring to the volume of space around a

charged particle where its electric field is greatly influencing other charged particles. Since the electrons are lighter they move faster and are more effective at shielding the potential set up by the ions. This means that when the ions move in an ion acoustic wave, electrons follow and provide Debye shielding. As long as the radar wavelength is much smaller than the Debye length defined by the radius of the Debye sphere, the scattering is off independent, free electrons rather than the group of electrons around ions (Beynon and Williams, 1978). However, when the wavelength is much greater than the Debye length, the scattering is from electron density structures matching the Bragg condition that are controlled by ion acoustic waves and plasma waves (Beynon and Williams, 1978). The ions are ineffective as scatterers due to their large mass (Salpeter, 1960a), but because of the surrounding electrons the backscatter from ion acoustic waves can still be seen as the ion line in the IS spectrum. The ion lines are centred at the radar frequency with a width of $\omega_{\mathfrak{R},\mathrm{i}}$ (Yngvesson and Perkins, 1968).

1.1 Motivation

When the IS technique was developed, the idea was to look at the backscattered signal with a width corresponding to the Doppler shift from thermal motion of independent, free electrons (Gordon, 1958). However, the very first received signal revealed that the backscatter gave rise to in general two lines in the IS spectrum with a much more narrow peak than what was expected for a thermal gas of electrons. The heavier ions largely dictate the low-frequency motion of electrons through the interaction with electric fields, and electrons inside the sphere of influence, the Debye sphere, contribute to the scattering leading to the ion lines in the IS spectrum. Around the peak frequencies of the ion lines and plasma lines more electrons contribute to the ion lines since the plasma lines are the result of scattering off free electrons that are more spread out in frequency due to thermal motion giving a Doppler broadening (Salpeter, 1960b). Therefore, with more scatterers, hence more power in the signal, the ion lines are easier to detect compared to the plasma lines.

Initially, the plasma lines were difficult to observe (Dougherty and Farley, 1960), but observation techniques have improved, and Vierinen et al. (2017) report that it is possible, using the Arecibo radar, to measure the plasma lines from thermal electrons at altitudes as high as 1000 km. These measurements of the full IS spectrum range in frequency from −12.5 MHz to 12.5 MHz with a resolution of about 1.5 kHz, and 1.5 km altitude resolution starting at 200 km. When suprathermal electrons are present, the plasma lines are enhanced and it is possible for less sensitive radars to detect the plasma lines at high altitudes. This is by far the most accurate way of measuring the plasma density from ground

and can also be used to observe electron temperature and ionospheric electron density variations during auroral precipitation (Vierinen et al., 2017).

Djuth et al. (2018) provide observations and measurements of the plasma lines, dependent on "phase energy", meaning energy as a function of the phase velocity of electrons. Their results showed a much larger intensity of photoelectron enhanced plasma waves for high phase velocity than the theory predicted. Djuth et al. (2018) argues that the discrepancy can be traced back to the theory of Perkins and Salpeter (1965), specifically the assumption that the high energy portion of the photoelectron tail was Maxwellian. Djuth et al. (2018) then argue that Guio et al. (1998) did not address this problem in their calculations since "this calculation/formalism is currently only in the $\boldsymbol{B}$ field-aligned direction". It is therefore of interest to improve on this theory to handle backscatter at large angles to the magnetic field.

The work of Djuth et al. (2018) further study the difference in frequency between the up- and downshifted photoelectron enhanced plasma lines, $\Delta f_{\mathfrak{R}} = f_{\mathfrak{R}+} - f_{\mathfrak{R}-}$. This parameter is interesting since it can be used to estimate several ionospheric parameters, for example the electron temperature (Djuth et al., 2018). This was also discussed by Guio et al. (1998), which used a numerical code for the plasma dispersion function that had as its high frequency solutions the up- and downshifted plasma wave frequencies. They then concluded that for low frequency radars the suprathermal electrons are influencing the Doppler frequency of the plasma lines more than the thermal electrons.

A major result from Djuth et al. (2018) was an aspect angle function that the measured plasma line frequency followed, $f_{\mathfrak{R}}(\theta) = A(\cos\theta)^{0.97}$, where A is a normalization constant. This was an empirically derived formula using a value of $\boldsymbol{B}$ from the International Geomagnetic Reference Field (IGRF) model, and it was discussed whether the power should in theory have been 1.0. The authors argued that the error could not be associated with the IGRF model since this would yield an unrealistically high error in the model, eventually leading the authors to the conclusion that an improved theory which includes the magnetic field is needed. Guio et al. (1998) developed a code that could calculate the plasma dispersion function parallel to the magnetic field for arbitrary distribution functions dependent on velocity and pitch angle, where pitch angle refer to the angle between the particle velocity vector and the magnetic field line. A possible solution proposed by Djuth et al. (2018) was to extend the formalism of Guio et al. (1998) to include directions at large angles to the magnetic field, and that "Simulations/theoretical efforts aimed at determining how a bump-on-tail instability develops in the ionosphere in the presence of the multi-peaked PE [photoelectron] distribution function are highly desirable".

1.2 Thesis structure

In chapter 2 the theoretical background is laid out. Section 2.1 gives a derivation of the IS spectrum as presented in Hagfors (1961). The IS spectrum can be derived through different approaches. Here, a perturbed Vlasov equation and density fluctuations is used. Section 2.2 gives an overview of what is meant by "suprathermal electrons" and section 2.3 take a look at the work done by Guio (1998) about incorporating the suprathermal electrons into the derivation of the plasma line in the IS spectrum.

Chapter 3 presents derivations of dielectric functions. The kind of functions that have historically been used to represent the distribution of particles in the ionosphere are described. In addition, further analysis is done of the equations for the calculation of the IS spectrum, and a solution to numerically solve the IS spectrum for arbitrary isotropic velocity distribution functions is presented.

Chapter 4 explains how the computer code was implemented and some issues that arose, leading to the calculation of the IS spectrum using two different methods, a Simpson's rule algorithm and a chirp z-transform. Further, an explanation of how the arbitrary isotropic distribution was included to the derivation of the IS spectrum is given, and tests for the numerical precision obtained by the program are described.

In chapter 5 the results obtained from the program are presented and discussed in line with the ideas presented in the preceding chapters. The IS spectrum is calculated using the different dielectric functions discussed and presented in chapter 3. The power in the plasma line and how it changes with electron number density and aspect angle is investigated, in reference to an observation made by the Arecibo radar.

Finally, chapter 6 presents a conclusion of the work done in the thesis. This also includes summarizing the shortcomings of the program developed here and a discussion of some suggested future work relevant to this work that are possible further uses of the program.

2

Background

The IS spectrum is derived in this chapter following the work by Hagfors (1961). This describes the theory behind measurements of the plasma lines at large angles to the magnetic field which was done to later be able to extend the Hagfors-theory by including suprathermal electrons. A presentation of what is meant by the term suprathermal electrons is given, in addition to some background on the work that has been done to derive the velocity distribution function for electrons at ionospheric heights.

2.1 Derivation of the incoherent scatter spectrum

Before going into the derivation of the equation for the IS spectra, or its dual representation the auto-correlation function (ACF), some mathematical notation is presented. This cover formulas that are used extensively in the derivation of the IS spectra and that make the notation and the derivation more compact and readable.

2.1.1 Fourier transforms

When dealing with waves, it is useful to move from space and time coordinates to their respective frequency representations. In time, this means frequency, f, or angular frequency, $\omega = 2\pi f$; while in space the wave vector, $\boldsymbol{k}$, is used, which represents the direction of propagation of harmonic plane waves. Moving from time and space to the frequency representations are done through Fourier transformations, which, for an arbitrary function Ψ of space and time, may be defined as

$$\mathfrak{F}_T\left\{\Psi(\boldsymbol{r},t)\right\} = \Psi(\boldsymbol{r},\omega) = \int_T \Psi(\boldsymbol{r},t)\exp[-i\omega t]\mathrm{d}t \tag{2.1a}$$

$$\mathfrak{F}_V\left\{\Psi(\boldsymbol{r},t)\right\} = \Psi(\boldsymbol{k},t) = \int_V \Psi(\boldsymbol{r},t)\exp[i\boldsymbol{k}\cdot\boldsymbol{r}]\mathrm{d}^3\boldsymbol{r} \tag{2.1b}$$

where $\boldsymbol{r}$ is the position vector, t is the time, V is the volume of space that is integrated over and T is the total time that is integrated over. The inverse transformations are defined as

$$\mathfrak{F}_T^{-1}\{\Psi(\boldsymbol{r},\omega)\} = \Psi(\boldsymbol{r},t) = \frac{1}{2\pi}\int_\Omega \Psi(\boldsymbol{r},\omega)\exp[i\omega t]\mathrm{d}\omega \tag{2.2a}$$

$$\mathfrak{F}_V^{-1}\{\Psi(\boldsymbol{k},t)\} = \Psi(\boldsymbol{r},t) = \frac{1}{(2\pi)^3}\int_{\boldsymbol{K}} \Psi(\boldsymbol{k},t)\exp[-i\boldsymbol{k}\cdot\boldsymbol{r}]\mathrm{d}^3\boldsymbol{k} \tag{2.2b}$$

where Ω is the span of frequencies, ω, that is integrated over and $\boldsymbol{K}$ is the span of wave vectors, $\boldsymbol{k}$, that is integrated over. This give a transformation for time and space as

$$\mathfrak{F}_{V,T}\{\Psi(\boldsymbol{r},t)\} = \Psi(\boldsymbol{k},\omega) = \int_V\int_T \Psi(\boldsymbol{r},t)\exp[\boldsymbol{k}\cdot\boldsymbol{r}-\omega t]\mathrm{d}t\mathrm{d}^3\boldsymbol{r}. \tag{2.3}$$

The subscripts on the Fourier transform symbol, $\mathfrak{F}$, denote a transformation to or from space (V) or time (T).

2.1.2 Ensemble average

Functions of parameters that are of stochastic nature, with statistical properties at least approximately independent of space and time, so-called statistically homogenous and stationary random processes, can be represented as a power spectrum or an ACF. The ensemble average is defined to get information about the power spectrum, more specifically the expression $\langle|\Psi(\boldsymbol{k},t+\tau)|^2\rangle$, i.e. the

notation $\langle\cdot\rangle$ define an ensemble average. Further, we write

$$\begin{aligned}\langle\Psi(\boldsymbol{k}, t+\tau)\Psi^*(\boldsymbol{k}, t)\rangle &= \int_V\int_V \langle\Psi(\boldsymbol{r}_1, t+\tau)\Psi^*(\boldsymbol{r}_2, t)\rangle \\ &\qquad \times \exp[i\boldsymbol{k}\cdot\boldsymbol{r}_1]\exp[-i\boldsymbol{k}\cdot\boldsymbol{r}_2]\mathrm{d}^3\boldsymbol{r}_1\mathrm{d}^3\boldsymbol{r}_2 \\ &= \int_V\int_V \langle\Psi(\boldsymbol{r}+\boldsymbol{r}', t+\tau)\Psi^*(\boldsymbol{r}, t)\rangle \\ &\qquad \times \exp[i\boldsymbol{k}\cdot\boldsymbol{r}']\mathrm{d}^3\boldsymbol{r}\mathrm{d}^3\boldsymbol{r}'\end{aligned} \tag{2.4}$$

and let $\boldsymbol{r}_1 \to \boldsymbol{r}+\boldsymbol{r}'$ and $\boldsymbol{r}_2 \to \boldsymbol{r}$. The expected value is assumed to be independent of $\boldsymbol{r}$ and t, i.e. the assumptions of homogeneity and stationarity are applied. This makes the first integral over $\boldsymbol{r}$ trivial, yielding

$$\langle\Psi(\boldsymbol{k}, t+\tau)\Psi^*(\boldsymbol{k}, t)\rangle = V\left\langle|\Psi(\boldsymbol{r}, t)|^2\right\rangle\int_V \Upsilon(\boldsymbol{r}', \tau)\exp[\boldsymbol{k}\cdot\boldsymbol{r}']\mathrm{d}\boldsymbol{r}' \tag{2.5}$$

where $\Upsilon(\boldsymbol{r}', \tau)$ is the ACF of Ψ in space and time normalized so that $\Upsilon(0,0) \equiv 1$. Now the Fourier transforms in time are included and the same manipulation is carried out:

$$\begin{aligned}\langle|\Psi(\boldsymbol{k}, \omega)|^2\rangle &= \int_V\int_V\int_T\int_T \langle\Psi(\boldsymbol{r}_1, t_1)\Psi^*(\boldsymbol{r}_2, t_2)\rangle\exp[i(\boldsymbol{k}\cdot\boldsymbol{r}_1 - \omega t_1)] \\ &\qquad \times \exp[-i(\boldsymbol{k}\cdot\boldsymbol{r}_2 - \omega t_2)]\mathrm{d}t_2\mathrm{d}t_1\mathrm{d}\boldsymbol{r}_2\mathrm{d}\boldsymbol{r}_1 \\ &= VT\langle|\Psi(\boldsymbol{r}, t)|^2\rangle\int_V\int_T \Upsilon(\boldsymbol{r}', \tau)\exp[i(\boldsymbol{k}\cdot\boldsymbol{r}' - \omega\tau)]\mathrm{d}\tau\mathrm{d}\boldsymbol{r}'.\end{aligned}$$

The result in eq. (2.5) can be used to simplify this as

$$\begin{aligned}\langle|\Psi(\boldsymbol{k}, \omega)|^2\rangle &= T\int_T \langle\Psi(\boldsymbol{k}, t+\tau)\Psi^*(\boldsymbol{k}, t)\rangle\exp[-i\omega\tau]\mathrm{d}\tau \\ &= VT\langle|\Psi(\boldsymbol{k}, t)|^2\rangle\int \Upsilon(\boldsymbol{k}, \tau)\exp[-i\omega\tau]\mathrm{d}\tau\end{aligned} \tag{2.6}$$

which is defined as the power density spectrum of the function Ψ, and where we have the normalization such that $\Upsilon(\boldsymbol{k}, 0) \equiv 1$.

2.1.3 Scattering cross section

For weak scattering (Born approximation) the scattering cross section per unit solid angle, per unit incident power density, and per unit scattering volume is obtained (Hagfors, 1961)

$$\sigma = \sigma_\mathrm{e} V\langle|n_\mathrm{e}(\boldsymbol{k})|^2\rangle \tag{2.7}$$

where σ_e is the single electron scattering cross section per unit solid angle and per unit incident power density, and where $\boldsymbol{k}$ is the difference between the

wave vectors of the incident radar wave ($\boldsymbol{k}_\mathrm{r}$) and the scattered wave, i.e.

$$\boldsymbol{k} = \left(-\boldsymbol{k}_\mathrm{r} + \frac{\pm\boldsymbol{\omega}_\Re}{c}\right) - \boldsymbol{k}_\mathrm{r}, \tag{2.8}$$

where ± is for up- and downshifted waves, $\boldsymbol{\omega}_\Re$ is the angular resonance frequency and direction of the ionospheric wave and c is the speed of light. Due to the ± on the resonance frequency, the wave vector will in general have the subscript ± for up- and downshifted waves, but this is omitted. $n_\mathrm{e}(\boldsymbol{k})$ is the number density of electrons as a function of wave vector, defined as the Fourier transform of $n_\mathrm{e}(\boldsymbol{r})$ through eq. (2.1b) as

$$n_\mathrm{e}(\boldsymbol{k}) = \frac{1}{V}\int_V n_\mathrm{e}(\boldsymbol{r})\exp[i\boldsymbol{k}\cdot\boldsymbol{r}]\mathrm{d}\boldsymbol{r} \tag{2.9}$$

where $n_\mathrm{e}(\boldsymbol{r})$ is the number density of electrons in space. The scattering cross section is needed for the power density spectrum of the scattered energy and given as

$$\sigma(\omega) = \sigma_\mathrm{e} V\langle|n_\mathrm{e}(\boldsymbol{k},\omega)|^2\rangle. \tag{2.10}$$

Here, $\langle|n_\mathrm{e}(\boldsymbol{k},\omega)|^2\rangle$ is the power density spectrum for electron number density of the spatial Fourier component of wave vector $\boldsymbol{k}$. Equations (2.7) and (2.10) are related through

$$\sigma = \int_{-\infty}^{\infty}\sigma(\omega)\mathrm{d}\omega. \tag{2.11}$$

2.1.4 Fluctuations

We assume fluctuations in a neutral plasma and that the average number density of ions and electrons are $n_{\mathrm{i},0}$ and $n_{\mathrm{e},0}$. The number of charges on the ions (to make things neutral) is then $Z := n_{\mathrm{e},0}/n_{\mathrm{i},0}$. The number density of electrons and ions are given as a sum over the given species inside a large periodicity cube $V = L^3$, as

$$n_\mathrm{e}(\boldsymbol{r}) = \sum_{j=1}^{n_{\mathrm{e},0}V}\delta(\boldsymbol{r}-\boldsymbol{r}_{\mathrm{e},j}) \tag{2.12a}$$

$$n_\mathrm{i}(\boldsymbol{r}) = \sum_{j=1}^{n_{\mathrm{i},0}V}\delta(\boldsymbol{r}-\boldsymbol{r}_{\mathrm{i},j}). \tag{2.12b}$$

$\boldsymbol{r}_{\mathrm{e},j}$ and $\boldsymbol{r}_{\mathrm{i},j}$ are the positions of all the electrons and ions. Charge density becomes

$$\rho(\boldsymbol{r}) = q\left[Zn_\mathrm{i}(\boldsymbol{r}) - n_\mathrm{e}(\boldsymbol{r})\right] \tag{2.13}$$

where q is the elementary charge, and the corresponding spatial Fourier component is

$$\rho(\boldsymbol{k}) = q\left[Zn_\mathrm{i}(\boldsymbol{k}) - n_\mathrm{e}(\boldsymbol{k})\right] \tag{2.14}$$

where we let $\boldsymbol{k} = 2\pi(\ell_1, \ell_2, \ell_3)/L,\ \ell_j \in \mathbb{Z}$ and have used the Fourier transform for the k^{th} coefficient as given in eq. (2.9).

The interactions between particles of different charges is through the electrical field, $\boldsymbol{E}$. $\boldsymbol{E}$ is a function of $\boldsymbol{r}$, and can therefore be expanded within a periodicity cube using Fourier series. By far the most dominant interactions in a non-relativistic plasma are through Coulomb forces. By neglecting other forces one implicitly assume that the velocity of interaction is infinite, hence $\boldsymbol{E}$ can be derived from a scalar electrical potential (Hagfors, 1961). From Poisson's equation:

$$\boldsymbol{E}(\boldsymbol{k}) = \frac{i\boldsymbol{k}}{\varepsilon_0 k^2}\rho(\boldsymbol{k}) \tag{2.15}$$

where ε_0 is the permittivity in a vacuum. This is a good approximation if the thermal energy of the electrons is considerably smaller than the relativistic rest energy of the electrons, meaning $k_{\text{B}}T_{\text{e}}/m_{\text{e}}c^2 \ll 1$ (Hagfors, 1961), where k_{B} is the Boltzmann constant, T_{e} is the electron temperature, m_{e} is the electron mass and c is the speed of light. The total energy of the plasma may be written as a sum of the contributions from the kinetic energy of the ions and electrons and the potential energy of the electric field, as

$$E = \frac{1}{2}\left[\sum_{j=1}^{n_{\text{i},0}V} m_{\text{i}} v_{\text{i},j}^2 + \sum_{j=1}^{n_{\text{e},0}V} m_{\text{e}} v_{\text{e},j}^2 + \varepsilon_0 \int_V \|\boldsymbol{E}(\boldsymbol{r})\|^2 \mathrm{d}^3\boldsymbol{r}\right]. \tag{2.16}$$

Parceval's theorem in combination with eqs. (2.14) and (2.15) can be used to rewrite the last term:

$$\begin{aligned} E_{\text{el}} &= \frac{1}{2}\varepsilon_0 \int_V \|\boldsymbol{E}(\boldsymbol{r})\|^2 \mathrm{d}^3\boldsymbol{r} = \frac{V\varepsilon_0}{2}\sum_{\boldsymbol{k}} \|\boldsymbol{E}(\boldsymbol{k})\|^2 \\ &= \frac{Vq^2}{2\varepsilon_0}\sum_{\boldsymbol{k}} -\frac{1}{k^2}|[Zn_{\text{i}}(\boldsymbol{k}) - n_{\text{e}}(\boldsymbol{k})]\,|^2 \end{aligned} \tag{2.17}$$

which is the same with or without an external magnetic field and is not altered by the presence of neutral particles colliding with ions and electrons (Hagfors, 1961). This leaves us with a total energy of

$$E = \frac{1}{2}\left[\sum_{j=1}^{n_{\text{i},0}V} m_{\text{i}} v_{\text{i},j}^2 + \sum_{j=1}^{n_{\text{e},0}V} m v_{\text{e},j}^2 - \frac{Vq^2}{\varepsilon_0}\sum_{\boldsymbol{k}} \frac{1}{k^2}|[Zn_{\text{i}}(\boldsymbol{k}) - n_{\text{e}}(\boldsymbol{k})]\,|^2\right]. \tag{2.18}$$

If the amount of particles is so high that $\ell_1\ell_2\ell_3 \ll n_{\text{e},0}V$ and $n_{\text{i},0}V$ ($n_{\text{e},0}$ and $n_{\text{i},0}$ being continuous functions), many particles contribute to each particle density sample. Individual samples are denoted $\left\{\overline{n}_1, \ldots, \overline{n}_{8\ell_1\ell_2\ell_3}\right\}$ where $8\ell_1\ell_2\ell_3$ is the amount of samples needed in a 3D space to determine the Fourier components up to $\boldsymbol{k} = 2\pi(\ell_1, \ell_2, \ell_3)/L$. The discontinuous functions $n_{\text{i}}(\boldsymbol{r})$

and $n_e(\boldsymbol{r})$ are related to the sampled values, and to find this relation we consider wave numbers $\boldsymbol{k}_{\eta_1,\eta_2,\eta_3}$ where $|\eta_j| \leq \ell_j$. For any i^{th} axes $2\ell_j + 1$ sampling points are needed. To directly quote Hagfors (1961), "Again, from information theory, it follows that the sampled values (occupation numbers) may be obtained from $n_i(\boldsymbol{r})$ and $n_e(\boldsymbol{r})$ by integration over the periodicity cube with the following weighting factor:" (coefficients from 3D Fourier transform with periodic boundary conditions)

$$w(\boldsymbol{r} - \boldsymbol{r}_{m_1,m_2,m_3}) = \prod_{j=1}^{3} \frac{\sin\left[\frac{2\ell_j+1}{L}\pi\left(x_j - \frac{m_j L}{2\ell_j+1}\right)\right]}{(2\ell_j+1)\sin\left[\frac{\pi}{L}\left(x_j - \frac{m_j L}{2\ell_j+1}\right)\right]}. \tag{2.19}$$

So, η_j is the position indices in the frequency/Fourier transformed domain. Let us define

$$\boldsymbol{r}_{m_1,m_2,m_3} = L\left[\frac{m_1}{2\ell_1+1}, \frac{m_2}{2\ell_2+1}, \frac{m_3}{2\ell_3+1}\right] \tag{2.20}$$

to be the indexed position in the spatial domain, where $m_j = \{1, 2, \ldots, 2\ell_j + 1\}$. That is, the individual samples in space can be written as $\overline{n}_i(\boldsymbol{r}_{m_1,m_2,m_3})$ (for ions, similar for electrons). By making use of the Fourier transform in its discrete form, we get

$$\overline{n}_i(\boldsymbol{r}_{m_1,m_2,m_3}) = \frac{V}{\prod_{j=1}^{3}(2\ell_j+1)} \sum_{\eta_1=-\ell_1}^{\ell_1} \sum_{\eta_2=-\ell_2}^{\ell_2} \sum_{\eta_3=-\ell_3}^{\ell_3} n_i(\boldsymbol{k}_{\eta_1,\eta_2,\eta_3}) \exp[-i\boldsymbol{k}_{\eta_1,\eta_2,\eta_3} \cdot \boldsymbol{r}_{m_1,m_2,m_3}] \tag{2.21}$$

from which we obtain

$$\sum_{m_1}\sum_{m_2}\sum_{m_3} \overline{n}_i^2(\boldsymbol{r}_{m_1,m_2,m_3}) = \frac{V^2}{8\ell_1\ell_2\ell_3} \sum_{\eta_1}\sum_{\eta_2}\sum_{\eta_3} |n_i(\boldsymbol{k}_{\eta_1,\eta_2,\eta_3})|^2 \tag{2.22}$$

where, again, $m_j = \{1, 2, \ldots, 2\ell_j + 1\}$ and $\eta_j = \{-\ell_j, -\ell_j + 1, \ldots, \ell_j - 1, \ell_j\}$.

At this point the densities (or occupation numbers) have been discretized, but how likely is any given distribution, or microstate, of sampled densities to form, compared to all possible microstates? Since it was assumed that the velocities of the individual particles are statistically unrelated to the sampled densities, it is concluded that the probability is given by Gibbs distribution (for thermal particles) as

$$\exp\left\{\left[-E(\overline{n}_{e,\zeta}, \overline{n}_{i,\xi})\right]/k_B T\right\} \tag{2.23}$$

where ζ and ξ are indices running over all sampled particles, and with E given in eq. (2.18), being the energy of a microstate. The number of permutations of these microstates are given by $(n_{\mathrm{i},0}V)!\,(n_{\mathrm{e},0}V)!\,/\prod_{j=1}^{8\ell_1\ell_2\ell_3}\overline{n}_{\mathrm{i},j}!\prod_{j=1}^{8\ell_1\ell_2\ell_3}\overline{n}_{\mathrm{e},j}!$ (Hagfors, 1961), thus the probability density is

$$\wp(\overline{n}_{\mathrm{e},\zeta},\overline{n}_{\mathrm{i},\xi}) \sim \frac{(n_{\mathrm{i},0}V)!}{\prod_j \overline{n}_{\mathrm{i},j}!}\frac{(n_{\mathrm{e},0}V)!}{\prod_j \overline{n}_{\mathrm{e},j}!}\exp[-E/k_{\mathrm{B}}T] \tag{2.24}$$

where ζ, ξ and j are dummy variables running over all sampled values. By use of Stirling's formula/approximation this can be simplified as (Hagfors, 1961)

$$\wp(\overline{n}_{\mathrm{e},\zeta},\overline{n}_{\mathrm{i},\xi}) \sim \exp[-E/k_{\mathrm{B}}T]\exp\left[-\frac{8\ell_1\ell_2\ell_3}{2n_{\mathrm{e},0}V}\sum_j(\overline{n}_{\mathrm{e},j}^2 + Z\overline{n}_{\mathrm{i},j}^2)\right]. \tag{2.25}$$

This has sampled densities in the exponent on the form as seen in eq. (2.22). When going from sampled densities to their Fourier components we see in eq. (2.22) that the right-hand side have twice the amount of terms, since $n_{\mathrm{i}}(\boldsymbol{k})$ contains both real and imaginary terms. Therefore, when changing the variables, only the directions of the wave vector $\boldsymbol{k}$ pointing into one hemisphere are accounted for if we want to use $n_{\mathrm{e},\Re}$, $n_{\mathrm{e},\Im}$, $n_{\mathrm{i},\Re}$ and $n_{\mathrm{i},\Im}$ (real and imaginary) as independent variables (Hagfors, 1961). According to section 2.1.4, the Fourier components are linearly related to the sampled densities. Due to the linear relation, the derivatives in the Jacobian of the transformation equates to constants, giving a joint probability distribution for the real and imaginary components of (Hagfors, 1961)

$$\begin{aligned}&\wp(n_{\mathrm{i},\Re}, n_{\mathrm{e},\Re}, n_{\mathrm{i},\Im}, n_{\mathrm{e},\Im})\\ &\quad\sim \exp\left[-\frac{V}{n_{\mathrm{e},0}}\sum_{\eta_1=0}^{\ell_1}\sum_{\eta_2=-\ell_2}^{\ell_2}\sum_{\eta_3=-\ell_3}^{\ell_3}\left\{2X_{\mathrm{p}}^2\left[Z^2 n_{\mathrm{i},\Re+\Im}^2 + n_{\mathrm{e},\Re+\Im}^2\right.\right.\right.\\ &\qquad\left.\left.\left. - 2Z(n_{\mathrm{i},\Re}n_{\mathrm{e},\Re} + n_{\mathrm{i},\Im}n_{\mathrm{e},\Im})\right] + Zn_{\mathrm{i},\Re+\Im}^2 + n_{\mathrm{e},\Re+\Im}^2\right\}\right]\end{aligned} \tag{2.26}$$

where $n_{\mathrm{i},\Re+\Im}^2 := n_{\mathrm{i},\Re}^2 + n_{\mathrm{i},\Im}^2$ and $n_{\mathrm{e},\Re+\Im}^2 := n_{\mathrm{e},\Re}^2 + n_{\mathrm{e},\Im}^2$, and where $n_{\mathrm{i}} = n_{\mathrm{i}}(\boldsymbol{k}_{\eta_1,\eta_2,\eta_3})$, $n_{\mathrm{e}} = n_{\mathrm{e}}(\boldsymbol{k}_{\eta_1,\eta_2,\eta_3})$. Also, $(2X_{\mathrm{p}}^2)^{-1} = (\lambda_{\mathrm{D}}\|\boldsymbol{k}_{\eta_1,\eta_2,\eta_3}\|)^2$ with $\lambda_{\mathrm{D}}^2 = \varepsilon_0 k_{\mathrm{B}}T_{\mathrm{e}}/n_{\mathrm{e},0}q^2$ being the Debye length squared and where we have defined $X_{\mathrm{p}}^2 := m_{\mathrm{e}}\omega_{\mathrm{pe}}^2/2k_{\mathrm{B}}T_{\mathrm{e}}k^2$. This can be recognized as a Gaussian multidimensional probability density. One can also find that the Fourier components enter through products of distribution functions for each wave number, therefore, the components corresponding to different wave numbers are statistically independent. The expression for the distribution of the real parts of $n_{\mathrm{i}}(\boldsymbol{k})$ and $n_{\mathrm{e}}(\boldsymbol{k})$ for one particular wave number is written down separately as (Hagfors,

1961)

$$\wp(n_{\mathrm{i},\Re}, n_{\mathrm{e},\Re}) \sim \exp\left[-\frac{V}{n_{\mathrm{e},0}}\{n_{\mathrm{i},\Re}^2 Z(1+2X_{\mathrm{p}}^2 Z) + n_{\mathrm{e},\Re}^2(1+2X_{\mathrm{p}}^2) - 4ZX_{\mathrm{p}}^2 n_{\mathrm{i},\Re} n_{\mathrm{e},\Re}\}\right]. \tag{2.27}$$

It was assumed that this can be written as a Gaussian probability density and comparing with such a function yields (Hagfors, 1961)

$$\langle n_{\mathrm{e},\Re}^2\rangle = \langle n_{\mathrm{e},\Im}^2\rangle = \frac{n_{\mathrm{e},0}}{2V}\frac{1+2X_{\mathrm{p}}^2 Z}{1+2X_{\mathrm{p}}^2(1+Z)} \tag{2.28a}$$

$$\langle n_{\mathrm{i},\Re}^2\rangle = \langle n_{\mathrm{i},\Im}^2\rangle = \frac{n_{\mathrm{e},0}}{2VZ}\frac{1+2X_{\mathrm{p}}^2}{1+2X_{\mathrm{p}}^2(1+Z)} \tag{2.28b}$$

$$\langle n_{\mathrm{e},\Re} n_{\mathrm{i},\Re}\rangle = \langle n_{\mathrm{e},\Im} n_{\mathrm{i},\Im}\rangle = \frac{n_{\mathrm{e},0}}{2V}\frac{2X_{\mathrm{p}}^2}{1+2X_{\mathrm{p}}^2(1+Z)} \tag{2.28c}$$

$$\langle n_{\mathrm{e},\Re} n_{\mathrm{i},\Im}\rangle = \langle n_{\mathrm{e},\Im} n_{\mathrm{i},\Re}\rangle = 0. \tag{2.28d}$$

Further, it can be shown that

$$\langle |n_{\mathrm{e}}(\boldsymbol{k})|^2\rangle = \langle n_{\mathrm{e},\Re}^2\rangle + \langle n_{\mathrm{e},\Im}^2\rangle = \frac{n_{\mathrm{e},0}}{V}\frac{1+2ZX_{\mathrm{p}}^2}{1+2X_{\mathrm{p}}^2(1+Z)} \tag{2.29a}$$

$$\langle |n_{\mathrm{i}}(\boldsymbol{k})|^2\rangle = \langle n_{\mathrm{i},\Re}^2\rangle + \langle n_{\mathrm{i},\Im}^2\rangle = \frac{n_{\mathrm{e},0}}{VZ}\frac{1+2X_{\mathrm{p}}^2}{1+2X_{\mathrm{p}}^2(1+Z)} \tag{2.29b}$$

and since $\|\boldsymbol{k}\|$ is related to X_{p} through $(2X_{\mathrm{p}}^2)^{-1} = (\lambda_{\mathrm{D}}\|\boldsymbol{k}_{\eta_1,\eta_2,\eta_3}\|)^2$ it can be shown that, for $Z = 1$,

$$\lim_{\|\boldsymbol{k}\|\to 0}\langle |n_{\mathrm{e}}(\boldsymbol{k})|^2\rangle = \frac{n_{\mathrm{e},0}}{2V} \tag{2.30}$$

$$\lim_{\|\boldsymbol{k}\|\to \infty}\langle |n_{\mathrm{e}}(\boldsymbol{k})|^2\rangle = \frac{n_{\mathrm{e},0}}{V}. \tag{2.31}$$

That is, for small wavenumbers the power density of the fluctuations are one-half of what they would be in a gas without particle interaction, but similar for large wavenumbers.

2.1.5 Spectral distribution

The Boltzmann equation describe the evolution of phase-space densities and as a consequence also describe how density fluctuations vary in time with the inclusion of an ambient magnetic field (Hagfors, 1961). The Boltzmann equation for the phase-space density distribution is

$$\partial_t f + \boldsymbol{v}\cdot\partial_{\boldsymbol{r}} f + \mu_\alpha\left[\boldsymbol{E} + \boldsymbol{v}\times\boldsymbol{B}\right]\cdot\partial_{\boldsymbol{v}} f = \left(\frac{\delta f}{\delta t}\right)_{\mathrm{coll}} \tag{2.32}$$

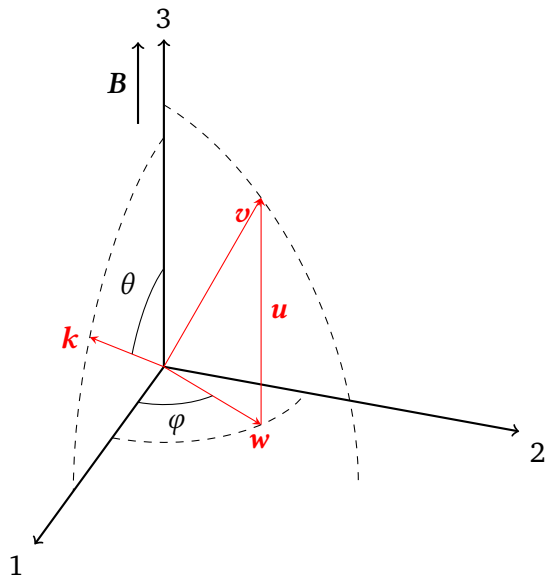


Figure 2.1: Coordinate system for solving eq. (2.35).

where $f = f(\boldsymbol{r}, \boldsymbol{v}, t)$ and $\boldsymbol{E}$ and $\boldsymbol{B}$ are functions of space and time. $\boldsymbol{B}$ is the magnetic field vector, $\alpha = \mathrm{e}, \mathrm{i}$ meaning the variables with subscript α are for electrons or ions, and where $\mu_\mathrm{e} := -q/m_\mathrm{e}$ for electrons and $\mu_\mathrm{i} := Zq/m_\mathrm{i}$ for ions. Deviations from the zeroth-order term (here: a Maxwellian) is assumed to be small and the distribution is linearized to be on the form $f = f_0(\boldsymbol{v})[1 + f_1]$ where $f_1 \ll 1$. Using the spatial Fourier transform yields

$$f_1(\boldsymbol{r}, \boldsymbol{v}, t) = \sum_{\boldsymbol{k}} f_1(\boldsymbol{k}, \boldsymbol{v}, t) \exp[-i\boldsymbol{k} \cdot \boldsymbol{r}] \tag{2.33}$$

and from the Laplace transform we have

$$f_1(\boldsymbol{k}, \boldsymbol{v}, s) = \int_0^\infty f_1(\boldsymbol{k}, \boldsymbol{v}, t) \exp[-st] \mathrm{d}t \tag{2.34}$$

which yields for the linearized Boltzmann equation

$$s' f_1 - f^{(1)} - i\boldsymbol{k} \cdot \boldsymbol{v} f_1 + \mu_\alpha \left[\frac{1}{f_0} \boldsymbol{E} \cdot \partial_{\boldsymbol{v}} f_0 - \boldsymbol{B} \left(\boldsymbol{v} \times \partial_{\boldsymbol{v}} f_1 \right) \right] = 0 \tag{2.35}$$

where $s' = s + \nu$ and ν is a collision frequency. In the succeeding the prime is omitted by letting $s' \to s$. In eq. (2.35), $f_1 = f_1(\boldsymbol{k}, \boldsymbol{v}, s)$, $f^{(1)} = f^{(1)}(\boldsymbol{k}, \boldsymbol{v}) = f_1(\boldsymbol{k}, \boldsymbol{v}, t = 0)$, $f_0 = f_0(\boldsymbol{v})$, $\boldsymbol{E} = \boldsymbol{E}(\boldsymbol{k}, s)$ and $\boldsymbol{B} = \boldsymbol{B}(\boldsymbol{k}, s)$. Figure 2.1 present a cylindrical coordinate system with $\boldsymbol{B}$ parallel to the third axis, along $\boldsymbol{u}$, θ give the angle away from parallel to $\boldsymbol{B}$ and φ is the angle away from the first axis in the plane perpendicular to $\boldsymbol{B}$. Using these coordinates the homogenous part

of eq. (2.35) is given as (Hagfors, 1961)

$$C_0(w,u) = \exp\left\{\frac{1}{\mu_\alpha B}[(s - iku\cos\theta)\varphi - ikw\sin\theta\sin\varphi]\right\}. \tag{2.36}$$

The inhomogeneous part can be found to have solution (Hagfors, 1961)

$$C(w,u,\varphi) = \frac{1}{\mu_\alpha B}\int_{\substack{\text{fixed}\\\text{limit}}}^{\varphi} \exp\left\{-\frac{1}{\mu_\alpha B}[(s - iku\cos\theta)\varphi' - ikw\sin\theta\sin\varphi']\right\} \times\left[\frac{\mu_\alpha}{f_{\alpha,0}}\partial_{\boldsymbol{v}}(f'_{\alpha,0})\boldsymbol{E} - f_\alpha^{(1)}(\boldsymbol{k},\boldsymbol{v}')\right]\mathrm{d}\varphi'. \tag{2.37}$$

The solution of eq. (2.35), where $f_1(\boldsymbol{k},\boldsymbol{v},s) = f_1(w,u,\varphi,s)$, then become

$$f_{\alpha,1}(\boldsymbol{k},\boldsymbol{v},s) = \frac{1}{\mu_\alpha B}\int_{-\infty}^{\varphi} G_\alpha(\varphi,\varphi') \left\{f_\alpha^{(1)}(\boldsymbol{k},\boldsymbol{v}') \mp \frac{i2X_\mathrm{p}^2}{f_{(\mathrm{e,i}),0}}\boldsymbol{k}\cdot\boldsymbol{v}'\,[Zn_\mathrm{i}(\boldsymbol{k},s) - n_\mathrm{e}(\boldsymbol{k},s)]\right\}\mathrm{d}\varphi' \tag{2.38}$$

and hence a solution for thermal electrons and ions is implied. In the equation above, $\mp$ refer to $\alpha = \mathrm{e},\mathrm{i}$ and is for electrons and ions, respectively. G_e and G_i are integrating factors, given as (Bernstein, 1958)

$$\begin{aligned} G_\alpha &= \exp\left[\mp\int_{\varphi'}^{\varphi}\frac{s + i\boldsymbol{k}\cdot\boldsymbol{v}}{\Omega_\alpha}\mathrm{d}\varphi\right] \\ &= \exp\left[\mp\frac{s + iku\cos\theta}{\Omega_\alpha}(\varphi - \varphi') \mp \frac{ikw\sin\theta}{\Omega_\alpha}(\sin\varphi - \sin\varphi')\right]. \end{aligned} \tag{2.39}$$

$\Omega_\alpha = \mu_\alpha B$ is the gyrofrequency, where μ_α give the charge to mass ratio.

Integrating over velocity space yields the spatial densities:

$$n_\alpha(\boldsymbol{k},s) = \int f_{\alpha,0}(\boldsymbol{v})f_{\alpha,1}(\boldsymbol{k},\boldsymbol{v},s)\mathrm{d}\boldsymbol{v} \tag{2.40}$$

which gives us (Hagfors, 1961)

$$n(\boldsymbol{k},s) = Y_\mathrm{e}(\boldsymbol{k},s) - \frac{i}{n_0}2X_\mathrm{p}^2R_\mathrm{e}(\boldsymbol{k},s)\{ZN(\boldsymbol{k},s) - n(\boldsymbol{k},s)\} \tag{2.41a}$$

$$N(\boldsymbol{k},s) = Y_\mathrm{i}(\boldsymbol{k},s) + \frac{i}{n_0}2ZX_\mathrm{p}^2R_\mathrm{i}(\boldsymbol{k},s)\{ZN(\boldsymbol{k},s) - n(\boldsymbol{k},s)\} \tag{2.41b}$$

where the expressions

$$Y_\alpha(\boldsymbol{k},s) = -\int_{\boldsymbol{v}}\int_{\mp\infty}^{\varphi} G_\alpha(\varphi,\varphi')f_{\alpha,0}(\boldsymbol{v})f_{\alpha,1}(\boldsymbol{k},\boldsymbol{v}')\mathrm{d}\boldsymbol{v}\mathrm{d}\varphi' \tag{2.42}$$

and

$$R_\alpha(\boldsymbol{k}, s) = -\int_{\boldsymbol{v}} \int_{\mp\infty}^{\varphi} \boldsymbol{k}\boldsymbol{v}' G_\alpha(\varphi, \varphi') f_{(\mathrm{e,i}),0}(\boldsymbol{v}) \mathrm{d}\boldsymbol{v}\mathrm{d}\varphi' \tag{2.43}$$

was used. The integrals in eqs. (2.42) and (2.43) are solved later. Equation (2.41) can be rewritten to

$$n_\mathrm{e}(\boldsymbol{k}, s) = \frac{Y_\mathrm{e}(\boldsymbol{k}, s)\left(1 - \frac{i}{n_{\mathrm{e},0}} 2Z^2 X_\mathrm{p}^2 R_\mathrm{i}(\boldsymbol{k}, s)\right) - Y_\mathrm{i}(\boldsymbol{k}, s)\frac{i}{n_{\mathrm{e},0}} 2ZX_\mathrm{p}^2 R_\mathrm{e}(\boldsymbol{k}, s)}{1 - \frac{i}{n_{\mathrm{e},0}} 2X_\mathrm{p}^2 [R_\mathrm{e}(\boldsymbol{k}, s) + Z^2 R_\mathrm{i}(\boldsymbol{k}, s)]} \tag{2.44a}$$

$$n_\mathrm{i}(\boldsymbol{k}, s) = \frac{Y_\mathrm{i}(\boldsymbol{k}, s)\left(1 - \frac{i}{n_{\mathrm{e},0}} 2X_\mathrm{p}^2 R_\mathrm{e}(\boldsymbol{k}, s)\right) - Y_\mathrm{e}(\boldsymbol{k}, s)\frac{i}{n_{\mathrm{e},0}} 2ZX_\mathrm{p}^2 R_\mathrm{e}(\boldsymbol{k}, s)}{1 - \frac{i}{n_{\mathrm{e},0}} 2X_\mathrm{p}^2 [R_\mathrm{e}(\boldsymbol{k}, s) + Z^2 R_\mathrm{i}(\boldsymbol{k}, s)]} \tag{2.44b}$$

which yields the variation of electron and ion density with time through an inverse Laplace transformation, i.e.

$$n_\alpha(\boldsymbol{k}, t) = \frac{1}{i2\pi} \lim_{\iota\to\infty} \int_{-i\iota+\gamma}^{i\iota+\gamma} n_\alpha(\boldsymbol{k}, s) \exp[st] \mathrm{d}s \tag{2.45}$$

where $\gamma := \Re\{s\}$ is greater than the real part of all singularities of $n_\alpha(\boldsymbol{k}, s)$. The exact densities at time t will need an initial condition for time t_0, but due to the statistical nature of the problem initial conditions cannot be fixed (Hagfors, 1961). Nevertheless, when focusing on the state of many particles a way around this can be found by forming an ensemble average:

$$\langle n_\alpha^*(\boldsymbol{k}, 0) n_\alpha(\boldsymbol{k}, t)\rangle = \frac{1}{i2\pi} \lim_{\iota\to\infty} \int_{-i\iota+\gamma}^{i\iota+\gamma} \langle n_\alpha^*(\boldsymbol{k}, 0) n_\alpha(\boldsymbol{k}, s)\rangle \exp[st] \mathrm{d}s \tag{2.46}$$

where the left-hand side is the Fourier transform of an ACF, which, due to symmetry, can be written

$$\langle n_\alpha^*(\boldsymbol{k}, 0) n_\alpha(\boldsymbol{k}, t)\rangle = 2\Re\{\langle n_\alpha^*(\boldsymbol{k}, 0) n_\alpha(\boldsymbol{k}, t)\rangle\}. \tag{2.47}$$

This, along with the Wiener-Khinchine theorem, give

$$\langle |n_\alpha(\boldsymbol{k}, \omega)|^2\rangle = \frac{1}{\pi} \lim_{\gamma\to 0} \Re\{\langle n_\alpha^*(\boldsymbol{k}, 0) n_\alpha(\boldsymbol{k}, s)\rangle\}. \tag{2.48}$$

The left-hand side is the power spectrum of interest for the IS spectrum, but still $\langle n_\alpha^*(\boldsymbol{k}, 0) n_\alpha(\boldsymbol{k}, s)\rangle$ need to be evaluated. An expression for $n_\mathrm{e}(\boldsymbol{k}, s)$ was obtained in eq. (2.44a) and is used to get

$$\begin{aligned}\langle n_\mathrm{e}^*(\boldsymbol{k}, 0) n_\mathrm{e}(\boldsymbol{k}, s)\rangle = {} & \frac{\langle n_\mathrm{e}^*(\boldsymbol{k}, 0) Y_\mathrm{e}(\boldsymbol{k}, s)\rangle \left(1 - \frac{i}{n_{\mathrm{e},0}} 2Z^2 X_\mathrm{p}^2 R_\mathrm{i}(\boldsymbol{k}, s)\right)}{1 - \frac{i}{n_{\mathrm{e},0}} 2X_\mathrm{p}^2 [R_\mathrm{e}(\boldsymbol{k}, s) + Z^2 R_\mathrm{i}(\boldsymbol{k}, s)]} \\ & - \frac{\langle n_\mathrm{e}^*(\boldsymbol{k}, 0) Y_\mathrm{i}(\boldsymbol{k}, s)\rangle \frac{i}{n_{\mathrm{e},0}} 2ZX_\mathrm{p}^2 R_\mathrm{e}(\boldsymbol{k}, s)}{1 - \frac{i}{n_{\mathrm{e},0}} 2X_\mathrm{p}^2 [R_\mathrm{e}(\boldsymbol{k}, s) + Z^2 R_\mathrm{i}(\boldsymbol{k}, s)]}.\end{aligned} \tag{2.49}$$

The expressions $\langle n_e^*(\boldsymbol{k},0)Y_e\rangle$ and $\langle n_e^*(\boldsymbol{k},0)Y_i\rangle$ contain terms $\langle n_e^*(\boldsymbol{k},0)f_e^{(1)}(\boldsymbol{k},\boldsymbol{v})\rangle$ and $\langle n_e^*(\boldsymbol{k},0)f_i^{(1)}(\boldsymbol{k},\boldsymbol{v})\rangle$. Since it has already been assumed that the spatial density fluctuations are independent of the velocities of the individual particles, eq. (2.29) can be used to obtain

$$\langle n_e^*(\boldsymbol{k},0)f_e^{(1)}(\boldsymbol{k},\boldsymbol{v})\rangle = \frac{1}{n_{e,0}}\langle|n_e(\boldsymbol{k})|^2\rangle = \frac{1}{V}\frac{1+2ZX_p^2}{1+2X_p^2(1+Z)} \tag{2.50a}$$

$$\langle n_e^*(\boldsymbol{k},0)f_i^{(1)}(\boldsymbol{k},\boldsymbol{v})\rangle = \frac{Z}{n_{e,0}}\langle n_e^*(\boldsymbol{k})n_i(\boldsymbol{k})\rangle = \frac{1}{V}\frac{2ZX_p^2}{1+2X_p^2(1+Z)}. \tag{2.50b}$$

Going back to eq. (2.43), this can be found to be

$$R_e(\boldsymbol{k},s) = in_0\left[1-\frac{s}{\Omega_e}g_e\left(\boldsymbol{k},\frac{s}{\Omega_e}\right)\right] = in_0F_e\left(\boldsymbol{k},\frac{s}{\Omega_e}\right) \tag{2.51a}$$

$$R_i(\boldsymbol{k},s) = i\frac{n_0}{Z}\left[1-\frac{s}{\Omega_i}g_i\left(\boldsymbol{k},\frac{s}{\Omega_i}\right)\right] = i\frac{n_0}{Z}F_i\left(\boldsymbol{k},\frac{s}{\Omega_i}\right). \tag{2.51b}$$

To solve the integrals in eq. (2.42), $\langle n_e^*(\boldsymbol{k})Y_e\rangle$ and $\langle n_e^*(\boldsymbol{k})Y_i\rangle$ are required since $\boldsymbol{v}'$ is stochastic. Using eq. (2.50), it can be shown that (Hagfors, 1961)

$$\begin{aligned}\langle n_e^*(\boldsymbol{k})Y_e\rangle &= -\frac{1}{V}\frac{1+2ZX_p^2}{1+2X_p^2(1+Z)}\int_{\boldsymbol{v}}\int_{-\infty}^{\varphi}G_e(\varphi,\varphi')f_{e,0}(\boldsymbol{v})\mathrm{d}\boldsymbol{v}\mathrm{d}\varphi' \\ &= \frac{n_{e,0}}{\Omega_e V}\frac{1+2ZX_p^2}{1+2X_p^2(1+Z)}g_e\left(\boldsymbol{k},\frac{s}{\Omega_e}\right)\end{aligned} \tag{2.52a}$$

$$\begin{aligned}\langle n_e^*(\boldsymbol{k})Y_i\rangle &= -\frac{1}{V}\frac{2ZX_p^2}{1+2X_p^2(1+Z)}\int_{\boldsymbol{v}}\int_{\infty}^{\varphi}G_i(\varphi,\varphi')f_{i,0}(\boldsymbol{v})\mathrm{d}\boldsymbol{v}\mathrm{d}\varphi' \\ &= \frac{n_{e,0}}{\Omega_i V}\frac{2X_p^2}{1+2X_p^2(1+Z)}g_i\left(\boldsymbol{k},\frac{s}{\Omega_i}\right)\end{aligned} \tag{2.52b}$$

where $g_\alpha(\boldsymbol{k},s/\Omega_\alpha)$ is a Gordeyev integral, given as

$$\begin{aligned}g_\alpha\left(\boldsymbol{k},\frac{s}{\Omega_\alpha}\right) = -\int_0^\infty \exp\Bigg\{&-\left(\frac{s}{\Omega_\alpha}\right)y \\ &-\left[\sin^2\theta(1-\cos y)+\frac{1}{2}y^2\cos^2\theta\right]\frac{k_BT_\alpha k^2}{m_\alpha\Omega_\alpha^2}\Bigg\}\mathrm{d}y,\end{aligned} \tag{2.53}$$

where the general form of a Gordeyev integral is given as

$$g(\omega) = \int_0^\infty I(y,\omega)\exp[\tau\omega y]\mathrm{d}y, \tag{2.54}$$

where τ is some complex number. Equations (2.50) to (2.52) are used to rewrite eq. (2.49) which in turn is related to eq. (2.48), hence

$$\langle |n_\mathrm{e}(\boldsymbol{k},\omega)|^2 \rangle = \frac{n_{\mathrm{e},0}}{\pi V \omega} \frac{\Im\{-F_\mathrm{e}\}|1+2ZX_\mathrm{p}^2 F_\mathrm{i}|^2 + 4ZX_\mathrm{p}^4 \Im\{-F_\mathrm{i}\}|F_\mathrm{e}|^2}{|1+2X_\mathrm{p}^2(F_\mathrm{e}+ZF_\mathrm{i})|^2} \tag{2.55a}$$

$$\langle |n_\mathrm{i}(\boldsymbol{k},\omega)|^2 \rangle = \frac{n_{\mathrm{e},0}}{\pi Z V \omega} \frac{\Im\{-F_\mathrm{i}\}|1+2X_\mathrm{p}^2 F_\mathrm{e}|^2 + 4ZX_\mathrm{p}^4 \Im\{-F_\mathrm{e}\}|F_\mathrm{i}|^2}{|1+2X_\mathrm{p}^2(F_\mathrm{e}+ZF_\mathrm{i})|^2} \tag{2.55b}$$

where for the functions F_e and F_i we have

$$\begin{aligned} F_\mathrm{e}(\boldsymbol{k},\omega) = 1 - \left(i\frac{X(\omega)}{X_\mathrm{e}} + \Lambda_\mathrm{e} \right) \int_0^\infty \exp\left\{ -iy\frac{X(\omega)}{X_\mathrm{e}} - y\Lambda_\mathrm{e} \right. \\ \left. -\frac{1}{2X_\mathrm{e}^2}\left[\sin^2\theta(1-\cos y) + \frac{1}{2}y^2\cos^2\theta \right] \right\} \mathrm{d}y \end{aligned} \tag{2.56a}$$

$$\begin{aligned} F_\mathrm{i}(\boldsymbol{k},\omega) = 1 - \left(i\frac{\varkappa^2 X(\omega)}{ZX_\mathrm{e}} + \Lambda_\mathrm{i} \right) \int_0^\infty \exp\left\{ -iy\frac{\varkappa^2 X(\omega)}{ZX_\mathrm{e}} - y\Lambda_\mathrm{i} \right. \\ \left. -\frac{\varkappa^2}{2Z^2X_\mathrm{e}^2}\left[\sin^2\theta(1-\cos y) + \frac{1}{2}y^2\cos^2\theta \right] \right\} \mathrm{d}y \end{aligned} \tag{2.56b}$$

where $\varkappa := (m_\mathrm{i}/m_\mathrm{e})^{1/2}$. The parameters $X(\omega)$, X_e, X_p and Λ_α are defined as

$$X(\omega)^2 := \frac{m_\mathrm{e}}{2k_\mathrm{B}T_\mathrm{e}}\frac{\omega^2}{k^2} \tag{2.57a}$$

$$X_\mathrm{e}^2 := \frac{m_\mathrm{e}}{2k_\mathrm{B}T_\mathrm{e}}\frac{\Omega_\mathrm{e}^2}{k^2} \tag{2.57b}$$

$$X_\mathrm{p}^2 := \frac{m_\mathrm{e}}{2k_\mathrm{B}T_\mathrm{e}}\frac{\omega_\mathrm{pe}^2}{k^2} = \frac{1}{2k^2\lambda_\mathrm{D}^2} \tag{2.57c}$$

$$\Lambda_\alpha := \frac{\nu_\alpha}{\Omega_\alpha} \tag{2.57d}$$

where $\Omega_\alpha = \mu_\alpha B$ is the gyrofrequency of the electrons/ions and where ν_α is the effective collision frequency. The functions F_α are closely related to the susceptibility of a dielectric, with the susceptibility function given as

$$\chi_\alpha(\boldsymbol{k},\omega) = 2X_\mathrm{p}^2 F_\alpha(\boldsymbol{k},\omega) \tag{2.58}$$

which in turn is related to the dielectric function through

$$\epsilon(\boldsymbol{k},\omega) = 1 + \sum_\alpha \chi_\alpha(\boldsymbol{k},\omega) \tag{2.59}$$

where α represents different particle species. Equation (2.55a) can then be written into the probably more familiar form

$$\langle |n_e(\boldsymbol{k},\omega)|^2 \rangle = \frac{n_{e,0}}{\pi V \omega} \frac{\Im\{-F_e\}|1+\chi_i|^2 + \Im\{-F_i\}|\chi_e|^2}{|1+\chi_e+\chi_i|^2} \tag{2.60}$$

which is similar for ions. Equation (2.14) give a relation between charge density and number density, thus for charge density variations we obtain

$$\langle |\rho(\boldsymbol{k},\omega)|^2 \rangle = \frac{n_{e,0}}{\pi Z V \omega} \frac{\Im\{-F_e\} + Z\Im\{-F_i\}}{|1+2X_p^2(F_e+ZF_i)|^2}. \tag{2.61}$$

2.2 Suprathermal electrons

In the thermosphere, the most abundant molecular constituents are N_2 and O_2, with CO_2 being a minor one. A major atomic constituent is O, produced from dissociation of O_2 by solar ultraviolet (UV) photons and by energetic particle impact (Rees, 1989). All the charged species that make up the ionosphere are produced either directly by photoionization and impact ionization of neutral atoms and molecules, or indirectly by subsequent ionic-chemical reactions (Rees, 1989).

Photoionization is the principal mechanism that produces the ionosphere, and for the three major thermospheric species we have (Rees, 1989)

$$N_2 + h\nu(< 796\,\text{Å}) \longrightarrow {N_2}^+ + e \tag{R2.1}$$
$$O_2 + h\nu(< 1026\,\text{Å}) \longrightarrow {O_2}^+ + e \tag{R2.2}$$
$$O + h\nu(< 911\,\text{Å}) \longrightarrow O^+ + e \tag{R2.3}$$

where h is the Planck's constant and ν is the frequency of a photon, i.e., $h\nu$ is the energy of a photon. The wavelengths specified in the parenthesis correspond to the ionization thresholds for the production of ions in their ground electronic state. Electrons result from these reactions, who are then called primary photoelectrons. The primary electrons often have enough energy to cause several ionizations where secondary electrons are created (Guio, 1998). Dissociative ionization is an additional source of atomic ions,

$$O_2 + h\nu(< 662\,\text{Å}) \longrightarrow O^+ + O + e \tag{R2.4}$$
$$N_2 + h\nu(< 510\,\text{Å}) \longrightarrow N^+ + N + e \tag{R2.5}$$

so photons with sufficient energy can simultaneously ionize and dissociate the molecule. Photoionization can lead to several electronically excited states of the ions and this is true also for photodissociation

$$O_2 + h\nu(< 1749\,\text{Å}) \longrightarrow O(^1D) + O(^3P). \tag{R2.6}$$

The energy corresponding to the wavelengths given in reactions (2.1) to (2.5) are threshold energies that specify the minimum photon energy required for the reaction to proceed. However, at wavelengths shorter than the threshold wavelength the photoionization cross section is larger (Rees, 1989), and the reactions therefore proceed at a higher rate in cases of excess energy. Even though it can be seen from reaction (2.6) that some excess energy may go into internal excitation of the products, a lot of the excess energy go to kinetic energy in electrons. It is possible to show that most of the excess energy go to the lighter electrons (Rees, 1989), which provide them with sufficient energy to create secondary electrons through electron impact ionization.

Secondary electrons may also be created by precipitating electrons, or primary auroral electrons. They are an external source to the atmosphere and ionize the atmosphere through collisions with gases, which again produce secondary electrons (Rees, 1989). These secondary electrons are the equivalent of the photoelectrons that are created in photoionization. When trying to obtain a description of the primary and secondary electrons one may turn the attention to the Lambert-Beer law:

$$I(\lambda, z) = I_\infty(\lambda) \exp[-\tau(\lambda, z)] \tag{2.62}$$

which states that at a point in the atmosphere, the intensity at wavelength λ is equal to $I_\infty(\lambda)$ scaled with an exponential, where τ is the optical depth. This is true for photons, but electrons do not annihilate in collisions with atoms and molecules. Instead, they scatter and loose energy and possibly cause ionization and production of secondary electrons, hence the Lambert-Beer law no longer suffices (Rees, 1989). Rather, cross sections for elastic and inelastic collisions are considered, which again can be divided up into cross sections for ionization and production of secondary electrons since the energy of degraded primary and secondary electrons is not in general the same (Rees, 1989). The angular scattering is also different, so while primary electrons are mostly scattered forward, secondary electrons are produced close to isotropically (Rees, 1989). In addition, there are electron-electron Coulomb collisions between energetic and thermal electrons, giving a friction-like energy transfer. These considerations give an expression describing the energy transfer for primary and secondary electrons. While primary and secondary electrons are in the process of loosing energy, they have more energy than the thermal electrons and are denoted suprathermal electrons.

2.3 Numerical description of suprathermal distributions

The theory describing scattering off magnetized electrons with the inclusion of collisions and an ambient magnetic field is described by for example Hagfors (1961). The result obtained there (eq. (2.55a)), however, is only considering thermal electrons with a velocity distribution modelled by a Maxwellian. With better techniques for observing the plasma lines, this part of the IS spectrum drew more attention. For example, an electron density-aspect angle dependency in the plasma line power was observed, but to interpret and explain these new findings, suprathermal electrons would have to be included in the theory. Electrons from photoionization and auroral precipitation contribute to make the plasma line detectable with more radars (Vierinen et al., 2017), but they also change the electron velocity distribution making them more difficult to represent in the IS theory. The suprathermal electrons are seen in the velocity distribution function as a high energy tail and loose energy to the larger population of thermal electrons (Rees, 1989). The velocity distribution of electrons with higher energy has a more complex variation in energy than the thermal electrons and are therefore harder to model.

Guio (1998) focused on obtaining a better model for the plasma line including the contributions from suprathermal electrons. The resulting model was made for the case of observations along the magnetic field lines. It was based on a velocity distribution where the thermal and suprathermal electron populations was spilt in two. The thermal population was represented by a Spitzer function, while the suprathermal population was pitch angle resolved by considering an electron transport model providing calculations of the angular energy flux of suprathermal electrons (Guio, 1998). Moments of the velocity distribution function can be calculated from the angular moments of the intensity in the transport equation, and the derivation of the first four moments are presented in Guio (1998). Applying such velocity distribution functions and extend the calculations of the dielectric function presented in Guio (1998) to the generalized case of radar observations at oblique angles to the magnetic field line was suggested by Guio (1998) for future work. It is of interest to try to combine the theory presented in Hagfors (1961) with the work by Guio (1998).

AURORA is the name of a time-dependant multi-stream electron transport code that is able to calculate the electron distribution in the ionosphere dependent on altitude, phase velocity and pitch angle along a magnetic field line (Gustavsson, personal communication). That is, it calculates the electron flux using the electron transport equation based on a solar spectrum, similar to the approach by Guio (1998).

3

Derivation of dielectric functions

The kinetic modelling of density fluctuations in a plasma that gives us the equations for the IS spectra will eventually depend on the velocity distribution function that is used. The theory presented in chapter 2 assumes a Maxwellian distribution. Here, the theory will be expanded by deriving the dielectric function for both a kappa velocity distribution function and for arbitrary isotropic velocity distribution functions, which are then substituted into the derivation of the IS spectrum. The subscripts α, e and i, used in the previous chapter to indicate particle species, are omitted here. Instead, it is assumed that the particle species under consideration is the electron.

3.1 The kappa distribution function

It has been observed through satellite experiments that the electron population in the magnetosheath may be better fitted by a kappa velocity distribution function that feature a high-energy tail rather than a Maxwellian (Olbert, 1968). Plasmas that are best represented by velocity distributions that feature a high-energy tail include solar flares, the solar wind and plasmas in a suprathermal radiation field (Mace and Hellberg, 1995). The Maxwellian distribution and

the kappa distribution are given as

$$f_{0,\mathrm{M}}(v) = (2\pi v_{\mathrm{th}}^2)^{-3/2} \exp\left\{-\frac{v^2}{2v_{\mathrm{th}}^2}\right\} \tag{3.1}$$

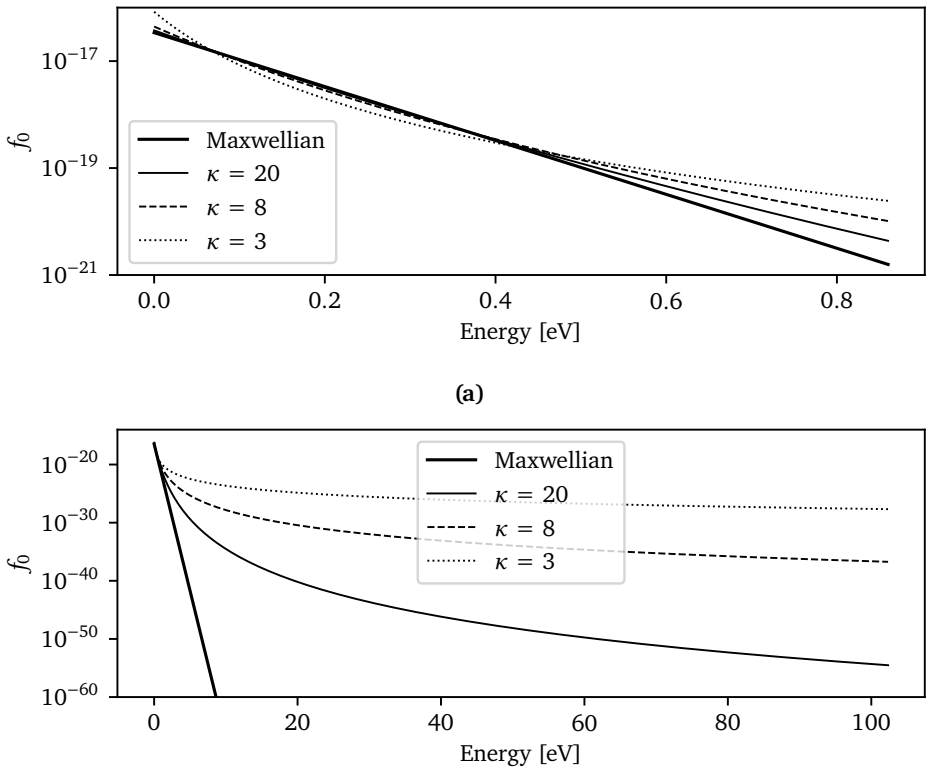
and

$$f_{0,\kappa}(v) = (\pi\kappa\Theta^2)^{-3/2} \frac{\Gamma(\kappa+1)}{\Gamma(\kappa-1/2)} \left(1 + \frac{v^2}{\kappa\Theta^2}\right)^{-(\kappa+1)} \tag{3.2}$$

where Γ is the gamma function, $\Theta^2 = v_{\mathrm{th}}^2(2\kappa - 3)/\kappa$ is the characteristic speed (Hellberg et al., 2009), $v_{\mathrm{th}}^2 = k_{\mathrm{B}}T/m$ is the thermal speed, and where k_{B}, T and m are the Boltzmann constant, temperature and mass, respectively. The subscript 0 signify an unperturbed distribution, while the subscripts M and κ are indicative of a Maxwellian distribution and a kappa distribution, respectively. Both functions are normalized so that $\int f_0 \mathrm{d}^3\boldsymbol{v} = 1$. An advantage of using the kappa distribution as given in eq. (3.2) is that it gives a family of distribution functions with longer tails, which in the limit as κ tends to infinity approaches the Maxwellian distribution (Mace, 2003).

Livadiotis and McComas (2011) give an overview of the kind of plasmas that may be represented by a kappa distribution and how varying the kappa index will lead the distribution to represent the different plasmas, first shown in Livadiotis and McComas (2010). The figure presented in Livadiotis and McComas (2011) show that a value of $\kappa = 2.5$ is assumed to set the boundary between what they denote the "near-equilibrium" region, $\kappa \in (2.5, \infty]$, and the "far-equilibrium" region, $\kappa \in (1.5, 2.5]$. Here, "equilibrium" refer to thermal equilibrium and for a plasma to be in the "near-equilibrium" region the "thermodynamic distance" must be sufficiently small. For reference, the plasma associated with X-rays and nanoflares are assumed to be in the near-equilibrium region ($\kappa > 2.5$) while the plasma associated with the inner heliosheath and solar flares are assumed to be in the far-equilibrium region ($\kappa \leq 2.5$).

A comparison between the Maxwellian distribution and the kappa distribution for different values for the κ index is presented in fig. 3.1. The interesting part that make the kappa distribution different from the Maxwellian distribution is that the kappa distribution represent an electron population with much higher phase-space densities at high phase velocity and energy. The increased phase-space density at high energy is shown in fig. 3.1b, where the tails of the kappa distributions greatly deviates from the Maxwellian tail.



(a)

(b)

Figure 3.1: Comparison between the Maxwellian and kappa distribution functions with varying kappa index. The thick, straight line represent the Maxwellian distribution, while the kappa distributions follow with decreasing kappa index. **(a)** shows the distributions at low energy, where mostly thermal electrons contribute and **(b)** shows the high-energy part where the tail representing suprathermal electrons is found.

3.2 Dielectric function for the kappa distribution

The Hagfors-theory for calculating the IS spectrum for a Maxwellian velocity distribution was derived in chapter 2. This derivation includes an expression for the dielectric function, obtained in eq. (2.59), in which we find the function $F(\boldsymbol{k}, \omega)$ given in eq. (2.56). Mace (2003) derive the dielectric function for both a Maxwellian distribution in eq. (3.1) and the kappa distribution in eq. (3.2), and by comparing our expression for the dielectric function for a Maxwellian distribution to the expression used by Mace (2003), our theory can be expanded by following the derivation by Mace (2003) for the kappa distribution.

To compare the dielectric functions for a Maxwellian distribution eq. (2.56) is rewritten using the substitutions $y' = y/\Omega$ and $\omega' = -\omega$. Mace (2003) assume

a collisionless plasma, and for the sake of comparison the collision term ν is omitted, which yields

$$F(\boldsymbol{k},\omega) = 1 + i\omega \int_0^\infty \exp\left\{ i\omega y - \frac{k_\mathrm{B} T k^2}{m\Omega}\left[\sin^2\theta(1-\cos(y\Omega)) + \frac{1}{2}y^2\Omega^2\cos^2\theta\right]\right\} \mathrm{d}y \tag{3.3}$$

where y and ω have been substituted back in for y' and ω'. The above can then be written in short as

$$F(\boldsymbol{k},\omega) = 1 + i\omega g(\boldsymbol{k},\omega), \tag{3.4}$$

again referring to a Gordeyev integral on the form of eq. (2.54) when using g. According to eqs. (2.58) and (2.59) and with the use of eq. (2.57c) the dielectric function become

$$\epsilon(\boldsymbol{k},\omega) = 1 + \sum_\alpha \frac{1}{k^2\lambda_\mathrm{D}^2}(1 + i\omega g(\boldsymbol{k},\omega)) = 1 + \sum_\alpha \chi(\boldsymbol{k},\omega). \tag{3.5}$$

Comparing this to eq. (16) in Mace (2003) it is evident that they are indeed identical when considering the definition of the Gordeyev integral in eq. (15) of Mace (2003), where they include the angular frequency, ω, in the Gordeyev integral.

The extension to a kappa distribution is then just a matter of substituting in the Gordeyev integral for a kappa distribution, defined by Mace (2003) as

$$g(\boldsymbol{k},\omega) = \frac{1}{2^{\kappa-1/2}\Gamma(\kappa+\frac{1}{2})}\int_0^\infty \exp\{i\omega y\}\, z(\boldsymbol{k},y)^{\kappa+1/2} K_{\kappa+1/2}[z(\boldsymbol{k},y)]\mathrm{d}y \tag{3.6}$$

where

$$z(\boldsymbol{k},y) = (2\kappa)^{1/2}\left[\frac{k^2\Theta^2\sin^2\theta}{\Omega^2}[1-\cos(y\Omega)] + \frac{1}{2}k^2y^2\cos^2\theta\frac{k_\mathrm{B}T}{m}\right] \tag{3.7}$$

and

$$\Theta^2 = 2\frac{\kappa-\frac{3}{2}}{\kappa}\frac{k_\mathrm{B}T}{m}. \tag{3.8}$$

K_β is the modified Bessel function of the second kind of real order β.

The Debye shielding in a plasma with kappa distributed electron velocities is modified such that the Debye length is decreased. The Debye length related to a kappa distribution is defined by Mace (2003) as

$$\lambda_{\mathrm{D},\kappa} = \lambda_{\mathrm{D,M}}\left(\frac{\kappa-\frac{3}{2}}{\kappa-\frac{1}{2}}\right)^{\frac{1}{2}} = \left[\frac{\varepsilon_0 k_\mathrm{B}T}{n_0 q^2}\frac{\kappa-\frac{3}{2}}{\kappa-\frac{1}{2}}\right]^{\frac{1}{2}}. \tag{3.9}$$

This Debye length is substituted with $\lambda_{\mathrm{D,M}}$ which can be found in eq. (2.55), if we write $X_{\mathrm{p}}^2 = 1/2k^2\lambda_{\mathrm{D}}^2$ as in eq. (2.57c).

Extending the derivation of the dielectric function to include a constant collision term, ν, was done for the Maxwellian case by expanding the Laplace parameter, s, as explained below eq. (2.35). The same can be done in the case of using the kappa distribution, i.e. we let $s = i\omega \longrightarrow i\omega + \nu$, yielding

$$g(\boldsymbol{k},\omega) = \frac{1}{2^{\kappa-1/2}\Gamma(\kappa+\frac{1}{2})}\int_0^\infty \exp\{i\omega y + \nu y\}\, z(\boldsymbol{k},y)^{\kappa+1/2}K_{\kappa+1/2}[z(\boldsymbol{k},y)]\mathrm{d}y, \tag{3.10}$$

$$\chi(\boldsymbol{k},\omega) = \frac{1}{k^2\lambda_{\mathrm{D}}^2}F(\boldsymbol{k},\omega) = \frac{1}{k^2\lambda_{\mathrm{D}}^2}[1 + (i\omega+\nu)g(\boldsymbol{k},\omega)] \tag{3.11}$$

which is substituted into eq. (3.5).

3.3 Dielectric function for isotropic distributions

For an arbitrary isotropic velocity distribution function the derivation of Mace (2003) can be used to obtain an expression for the dielectric function. The derivation starts from the Vlasov equation, similar to eq. (2.32), but without the collision term:

$$\partial_t f_{\alpha 1} + \boldsymbol{v}\cdot\partial_{\boldsymbol{x}} f_{\alpha 1} + \mu_\alpha \boldsymbol{v}\times\boldsymbol{B}_0\cdot\partial_{\boldsymbol{v}} f_{\alpha 1} = -\mu_\alpha \boldsymbol{E}_1\cdot\partial_{\boldsymbol{v}} f_{\alpha 0} \tag{3.12}$$

where μ is the charge to mass ratio and α denote particle species, but this subscript is dropped from here onward. The subscripts 0 and 1 denote zeroth-order and first-order terms, respectively. Again the Poisson's equation is used to get a description of the electric field (eq. (2.15)) through an electrostatic potential ϕ_1

$$\varepsilon_0 k^2\phi_1(\boldsymbol{k},s) = \sum_\alpha n_0 q\int f_1(\boldsymbol{k},\boldsymbol{v},s)\mathrm{d}^3v = \sum_\alpha \rho_1(\boldsymbol{k},s). \tag{3.13}$$

The parameter $\boldsymbol{k}$ appear from doing a Fourier transform in space while the parameter s appear through a Laplace transform in time. When applying the Fourier and Laplace transforms, eq. (3.12) yields (Mace, 2003)

$$f_1(\boldsymbol{k},\boldsymbol{v},s) = \frac{1}{\exp[-(2\pi/\Omega)(s+ik_\parallel v_\parallel)]-1}\int_\varphi^{\varphi+2\pi}\exp[P(\varphi')-P(\varphi)]Q(\varphi')\mathrm{d}\varphi' \tag{3.14}$$

where

$$P(\varphi) = -\frac{1}{\Omega}[(s+ik_\parallel v_\parallel)\varphi + ik_\perp v_\perp\sin\varphi] \tag{3.15}$$

$$Q(\varphi) = -\frac{1}{\Omega}\left[f_1(\boldsymbol{k},\boldsymbol{v},t=0) + i\frac{q}{m}\phi_1(\boldsymbol{k},s)\boldsymbol{k}\cdot\partial_{\boldsymbol{v}} f_0\right] \tag{3.16}$$

where $\parallel$ and $\perp$ refer to the parallel and perpendicular component of a vector relative to the magnetic field and φ is the gyro phase angle found in fig. 2.1, i.e., $\boldsymbol{v} = \boldsymbol{v}(\varphi) = (v_\perp \cos\varphi, v_\perp \sin\varphi, v_\parallel)^T$. Further, it is shown that $f_1(\boldsymbol{k}, \boldsymbol{v}, s)$ can be written on the form

$$f_1(\boldsymbol{k}, \boldsymbol{v}, s) = \int_{-\infty}^{\varphi} \exp[P(\varphi') - P(\varphi)] Q(\varphi') \mathrm{d}\varphi' \tag{3.17}$$

which is the same as eq. (2.38) except from the difference in notation. Substituting this expression for $f_1(\boldsymbol{k}, \boldsymbol{v}, s)$ into eq. (3.13) give an expression for $\rho_1(\boldsymbol{k}, s)$ on the form

$$\rho_1(\boldsymbol{k}, s) =: \psi(\boldsymbol{k}, s) + \chi(\boldsymbol{k}, s)\phi_1(\boldsymbol{k}, s) \tag{3.18}$$

where

$$\psi(\boldsymbol{k}, s) = n_0 q \int_{-\infty}^{0} \int f_1(\boldsymbol{k}, \boldsymbol{v}, t = 0) \exp[sy + i\boldsymbol{p}(y) \cdot \boldsymbol{v}] \mathrm{d}^3\boldsymbol{v}\mathrm{d}y \tag{3.19}$$

$$\chi(\boldsymbol{k}, s) = i\frac{n_0 q^2}{m} \int_{-\infty}^{0} \int \boldsymbol{p}'(y) \cdot \partial_{\boldsymbol{v}} f_0 \exp[sy + i\boldsymbol{p}(y) \cdot \boldsymbol{v}] \mathrm{d}^3\boldsymbol{v}\mathrm{d}y, \tag{3.20}$$

and where

$$\boldsymbol{p}(y) = \left(\tfrac{k_\perp}{\Omega} \sin(\Omega y), \tfrac{k_\perp}{\Omega}[1 - \cos(\Omega y)], k_\parallel y\right)^T \tag{3.21}$$

$$\boldsymbol{p}'(y) = \left(k_\perp \cos(\Omega y), k_\perp \sin(\Omega y), k_\parallel\right)^T. \tag{3.22}$$

While eq. (3.19) contains information about the initial charge perturbation, eq. (3.20) takes part in determining the long time behaviour of the plasma (Mace, 2003) and is recognized as the susceptibility function.

An integration by parts with respect to $\boldsymbol{v}$ yields for eq. (3.20) (Mace, 2003)

$$\chi(\boldsymbol{k}, s) = \frac{n_0 q^2}{m} \int_{-\infty}^{0} \int \boldsymbol{p}(y) \cdot \boldsymbol{p}'(y) \exp[sy + i\boldsymbol{p}(y) \cdot \boldsymbol{v}] f_0(\boldsymbol{v}) \mathrm{d}^3\boldsymbol{v}\mathrm{d}y. \tag{3.23}$$

Under the assumption that the distribution is isotropic and with a change of coordinates from cartesian to spherical, the above equation can be simplified further to the form (Mace, 2003)

$$\chi(\boldsymbol{k}, s) = 4\pi v_{\mathrm{th}}^2 \frac{\varepsilon_0}{\lambda_{\mathrm{D}}^2} \int_{-\infty}^{0} \exp[sy] p'(y) \int_0^{\infty} v \sin[p(y)v] f_0(v) \mathrm{d}v\mathrm{d}y \tag{3.24}$$

where $p'(y) = \mathrm{d}p(y)/\mathrm{d}y = \mathrm{d}[\boldsymbol{p}(y) \cdot \boldsymbol{p}(y)]^{1/2}/\mathrm{d}y$ and $n_0 q^2/m = v_{\mathrm{th}}^2 \varepsilon_0/\lambda_{\mathrm{D}}^2$ was used to rewrite the fraction in eq. (3.23). With the relations in eqs. (2.58) and (2.59) the dielectric function become

$$\epsilon(\boldsymbol{k}, \omega) = 1 + \sum_\alpha \chi(\boldsymbol{k}, \omega), \tag{3.25}$$

and the IS spectrum can be calculated for an arbitrary isotropic velocity distribution, $f_0(v)$.

In eq. (3.9) the change in the Debye length is taken care of with regard to the kappa distribution and this must also be done in the general case. Since we are now working with any arbitrary distribution, the Debye length cannot be derived analytically and a numerical calculation of the scaling is needed to correct for the change in Debye length. To see the effect of the Debye length on the susceptibility function, it is useful to look at the derivation of the Gordeyev integral for the kappa distribution, eq. (3.10), since the correction have already been pointed out in this case. This derivation is carried out by Mace (2003) and will only be outlined here.

The distribution in eq. (3.2) is inserted into eq. (3.24) and further consideration is made of the velocity integral:

$$I(y) = A(\kappa\theta^2)^{\kappa+1} \int_0^\infty \frac{v \sin[p(y)v]}{(\kappa\theta^2 + v^2)^{\kappa+1}} \mathrm{d}v \tag{3.26}$$

where A is the normalization constant in eq. (3.2). This expression can be further developed and is then substituted back into eq. (3.24) for the susceptibility function. The desired form of the susceptibility function is obtained after yet another rewriting, yielding

$$\chi(\boldsymbol{k}, s) = -\frac{\varepsilon_0}{\lambda_{\mathrm{D,M}}^2} \left(\frac{\kappa - \frac{1}{2}}{\kappa - \frac{3}{2}} \right) \left[1 - s \frac{\int_{-\infty}^0 \exp[sy] z^{\kappa+1/2} K_{\kappa+1/2}(z) \mathrm{d}y}{2^{\kappa-1/2} \Gamma(\kappa + \frac{1}{2})} \right]. \tag{3.27}$$

The Gordeyev integral used for the kappa distribution in eq. (3.10) is recognized and so is the correction of the Debye length defined in eq. (3.9).

With this in mind, the general case should be corrected for by evaluating the velocity integral and comparing to the value of the integral for a Maxwellian distribution. That is, in the same way we get the kappa correction from

$$\frac{\lambda_{\mathrm{D},\kappa}^2}{\lambda_{\mathrm{D,M}}^2} = \frac{\kappa - \frac{3}{2}}{\kappa - \frac{1}{2}} \tag{3.28}$$

any general Debye length can be found through

$$\frac{\lambda_{\mathrm{D,S}}^2}{\lambda_{\mathrm{D,M}}^2} = \frac{\int_{-\infty}^0 \int_0^\infty v \sin[p(y)v] f_{0,\mathrm{M}} \mathrm{d}v \mathrm{d}y}{\int_{-\infty}^0 \int_0^\infty v \sin[p(y)v] f_{0,\mathrm{S}} \mathrm{d}v \mathrm{d}y} \tag{3.29}$$

where S represent an arbitrary isotropic distribution and M the Maxwellian distribution.

3.4 Alternative derivation of the dielectric function for isotropic distributions

The susceptibility function for isotropic distributions (eq. (3.24), eq. (12) of Mace (2003)) found in the dielectric function can be expressed as

$$\chi(\boldsymbol{k}, s) = -4\pi \frac{n_0 q^2}{m} \left[\int_0^\infty f_0 \mathrm{d}v - \int_0^\infty f_0 \int_{-\infty}^0 s \exp[sy] \cos(pv) \mathrm{d}y \mathrm{d}v \right], \quad (3.30)$$

where we have used integration by parts with respect to y to rewrite eq. (3.24), i.e.

$$\begin{aligned} &\int_{-\infty}^0 \exp[sy] p' \sin(pv) \mathrm{d}y \\ &\qquad = \int_{-\infty}^0 \exp[sy] \left(-\frac{1}{v} \cos(pv) \right)' \mathrm{d}y \\ &\qquad = -\exp[sy] \left. \frac{1}{v} \cos(pv) \right|_{-\infty}^0 + \int_{-\infty}^0 s \exp[sy] \frac{1}{v} \cos(pv) \mathrm{d}y \\ &\qquad = -\frac{1}{v} + \frac{1}{v} s \int_{-\infty}^0 \exp[sy] \cos(pv) \mathrm{d}y \end{aligned} \quad (3.31)$$

with the assumption that $\Re\{s\} > 0$. In the above equations, $f_0 = f_0(v)$, $p = p(y)$ and $p' = p'(y) = \mathrm{d}p(y)/\mathrm{d}y$.

Mace (2003) argues that the form of eq. (3.30) is useful because one can factor out the term $\int_0^\infty f_0(v)\mathrm{d}v$. On this form you are more likely to find analytical solutions to expressions (e.g. the integral $\int_0^\infty f_0(v)\mathrm{d}v$) that are part of the evaluation of the susceptibility function, which would be more precise and provide faster computation of the IS spectrum.

3.5 Alternative versions of the kappa distribution

Even though the kappa distribution give more flexibility in representing the particle velocity distributions, it is not capable of representing an arbitrary population, hence there might still be cases where it falls short. To this end, we may want to look at more flexible distributions of a similar family or different distributions altogether. Gaelzer et al. (2016) derive the general dielectric tensor for a bi-kappa distribution for the case of a magnetized plasma with an anisotropic population of electrons and ions. A comprehensive analysis is

given of this bi-kappa distribution defined as

$$f_s^{(\alpha)}(v_{||}, v_\perp) = A_s^{(\sigma_s)}\left(1 + \frac{v_{||}^2}{\kappa_s w_{||s}^2} + \frac{v_\perp^2}{\kappa_s w_{\perp s}^2}\right)^{-\sigma_s} \tag{3.32}$$

where A_s is a normalization constant and $w_{||s}$ and $w_{\perp s}$ are proportional to the parallel and perpendicular thermal speeds, $v_{||}$ and $v_\perp$, but also functions of κ. s is the particle species and α_s, σ_s and κ_s are indices defining the distribution function. An implementation of this distribution would provide better chances of being able to fit the theoretical IS spectrum to real measurements, but the susceptibility function that the dielectric function depends on have no known implementation in computer code (Gaelzer et al., 2016), making this an issue for future work.

Ziebell et al. (2017) give derivations of the dispersion relation for two isotropic and four anisotropic kappa distributions, where one of the two isotropic distributions is the one given in eq. (3.2). With the derivation of the susceptibility function for arbitrary isotropic distributions, the second isotropic kappa distribution can also be used to calculate the IS spectrum and will provide more flexibility of choice, but without the same analytical development as for the kappa distribution in eq. (3.2) it was not of much interest.

4

Implementation in computer code

The equations taking part in the derivation of the IS spectrum, carried out in the preceding chapters, was implemented in computer code for numerical computation. Two algorithms with different advantages was implemented to solve the Gordeyev integrals. The Simpson's rule was slow, but easy on memory. In addition, the implementation of the Simpson's rule to solve integrals accepts an array representing the samples and an array representing the values at the sampled points, which makes it easy to customize a good and efficient sampling for a given integrand. The chirp z-transform algorithm was chosen due to its computational efficiency yielding high numerical accuracy, but at the cost of using a lot of memory. This algorithm was found to produce inconsistent results, and most of the focus was therefore on the implementation of the Simpson's rule.

4.1 Evaluating the Gordeyev integral using the Simpson's rule

The theory presented by Hagfors (1961) was used to calculate the IS spectrum, specifically eq. (2.55) for $\langle |n(\boldsymbol{k}, \omega)|^2 \rangle$, which in turn is a function of the suscep-

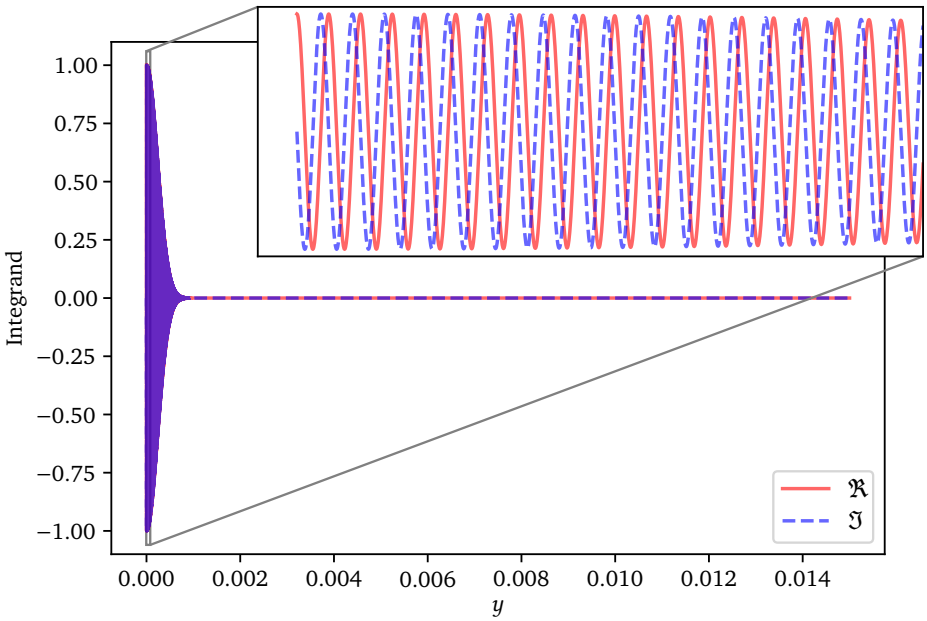


Figure 4.1: Shape of the integrand $I(\boldsymbol{k}, y)\exp[\tau\omega y]$ in eq. (4.1) with $\omega = 1.5 \times 10^6$ Hz ($f \approx 2.4 \times 10^5$ Hz) as a function of y. The red solid line is the real part of the integrand, while the blue dashed line is the imaginary part.

tibility function. Equation (2.58) is the susceptibility function for a Maxwellian distribution and eq. (3.11) is the susceptibility function for a kappa distribution, where both integrals in the expressions are on the form of a Gordeyev integral, that is

$$g(\boldsymbol{k}, \omega) = \int_0^\infty I(\boldsymbol{k}, y)\exp[\tau\omega y]\mathrm{d}y = \int_0^{y_{\max}} I(\boldsymbol{k}, y)\exp[\tau\omega y]\mathrm{d}y \tag{4.1}$$

where τ is a complex number. A lot of computation can be omitted when realizing that the integrand $I(\boldsymbol{k}, y)\exp[\tau\omega y]$ approaches zero very quickly, shown in fig. 4.1. Therefore, instead of integrating to infinity using a quadrature algorithm that handles such a function, a finite upper boundary $y_{\max}$ was chosen. To further take advantage of the shape of the integrand the integral was sampled according to the formula

$$y = (y')^a \tag{4.2}$$

where a is an integer. The sampling is illustrated in fig. 4.2, where the sampled value is given by the y axis and the number of sampling points goes along the x axis. Such a chirp-like sampling ensures that more points close to zero are used when evaluating the integral. The same idea can be applied to the sampling in frequency to make the IS spectra plots. Since the ion line lie in the kHz range, when plotting between frequencies in the MHz range, a lot of

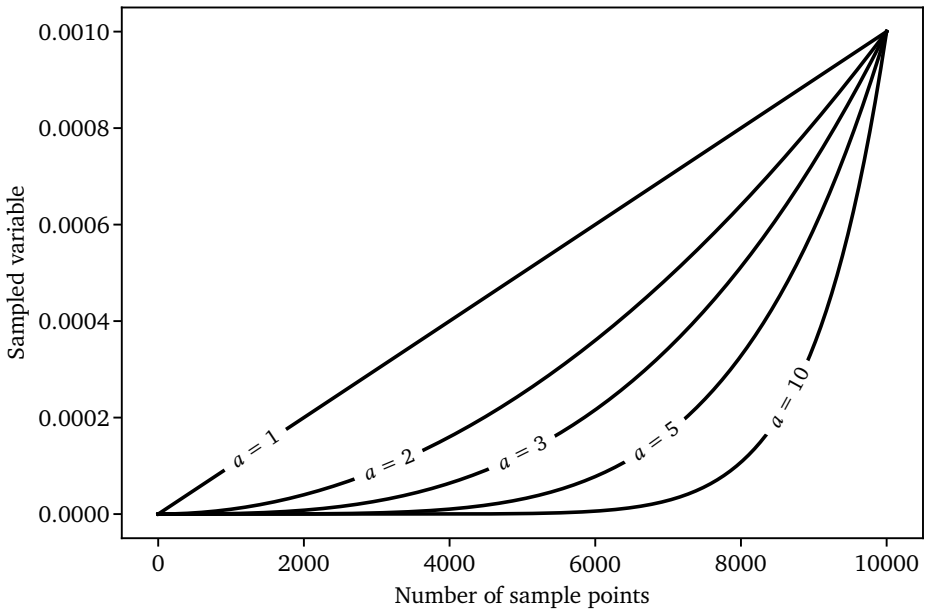


Figure 4.2: Sampling was done such that many points close to zero was chosen, with less emphasis put on larger values of the integration variable.

detail is lost if the number of sampling points at low frequency is not increased. The sampling in frequency was done according to eq. (4.2) with $a = 3$ (and should in general be done with a being odd) to preserve the order of a linear sampling on the real number line.

Equation (3.24) present the susceptibility function for an arbitrary isotropic distribution, and is also given on the form of a Gordeyev integral. The difference from the evaluation of the susceptibility functions for Maxwellian and kappa distributions discussed above is that first the velocity integral which is a function of the distribution function must be evaluated. Equation (3.24) is written as

$$\chi(\boldsymbol{k}, s) = 4\pi \frac{n_0 q^2}{m} \int_{-\infty}^{0} \exp[sy] p'(\boldsymbol{k}, y) \urcorner(\boldsymbol{k}, y) \mathrm{d}y \tag{4.3}$$

where

$$\urcorner(\boldsymbol{k}, y) = \int_0^{\infty} v \sin[p(\boldsymbol{k}, y)v] f_0(v) \mathrm{d}v \tag{4.4}$$

and it is clear that the velocity integral, $\urcorner(\boldsymbol{k}, y)$, only need to be calculated once for all y before substituting it into eq. (3.24). Equation (4.4) was evaluated in the same way as the Gordeyev integral by using the Simpson's rule for numerical integration and with an upper boundary $v_{\max}$. The value of $v_{\max}$ was chosen based on the available energies and subsequently velocities in the calculated electron fluxes. The maximum available energy from the calculated

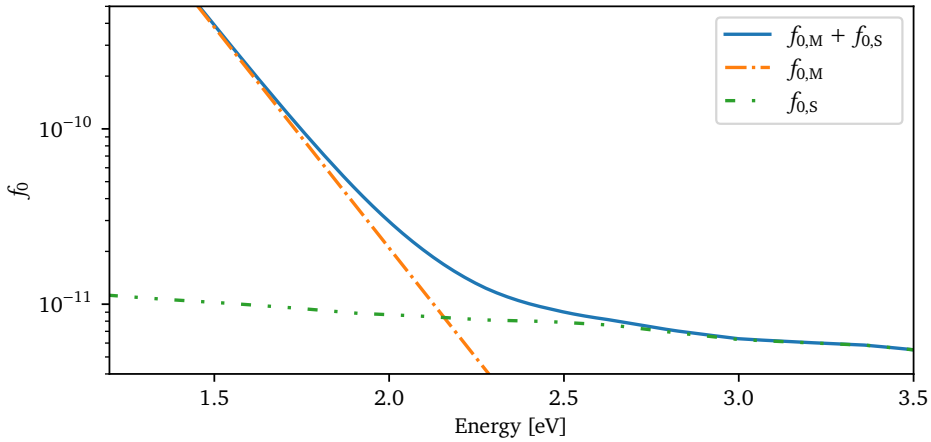


Figure 4.3: The construction of an electron velocity distribution from a calculated suprathermal distribution and a thermal distribution (Maxwellian) was done by adding the two arrays together. The intersection between the distributions can be seen in the figure and the sum of the two arrays is shown by the blue solid line, denoted $f_{0,M} + f_{0,S}$, while the two distributions are shown by the orange "dash-dot" line denoted $f_{0,M}$ (thermal) and the green "dash-dot-dot" line denoted $f_{0,S}$ (suprathermal).

fluxes was $E = 110\,\text{eV}$, and according to the formula

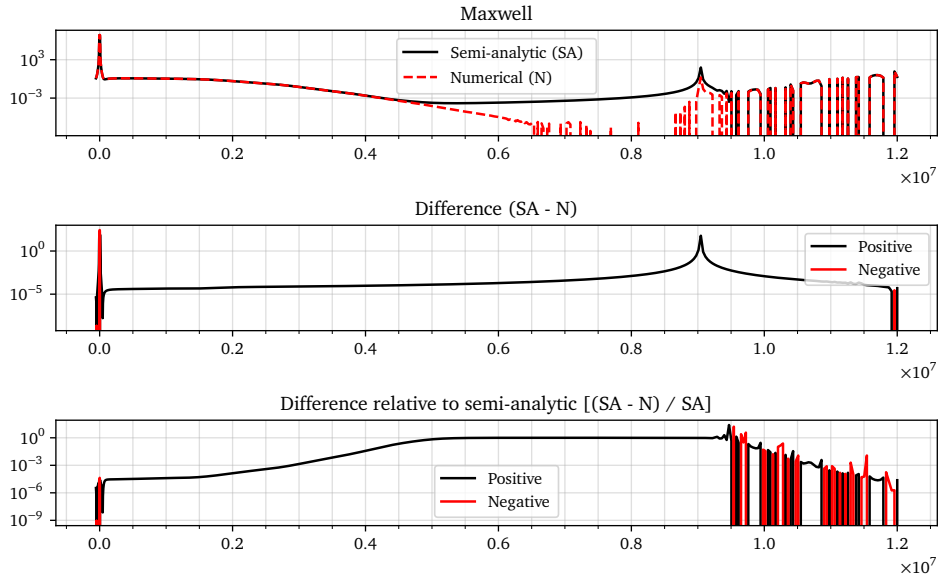
$$E = \frac{1}{2}mv^2, \tag{4.5}$$

the upper boundary was set to $v_{\text{max}} = 6 \times 10^6\,\text{ms}^{-1}$ (electrons with energy $E = 110\,\text{eV}$ has velocity $v \approx 6.22 \times 10^6\,\text{ms}^{-1}$).

This does not yield the same precision as an analytic derivation, for starters because a finite upper boundary is used in the integration in place of infinity, but also because the integration is done numerically. Nevertheless, with high enough sampling points, the difference in the subsequent numerical calculations will be small.

4.2 Implementation of calculated electron distributions

Equation (4.4) accepts an arbitrary isotropic distribution. To take advantage of this, suprathermal electron distributions was calculated for photoelectron production above the Arecibo Observatory and the magnetic conjugate ionosphere from solar spectra with the electron transport code AURORA (Gustavsson,



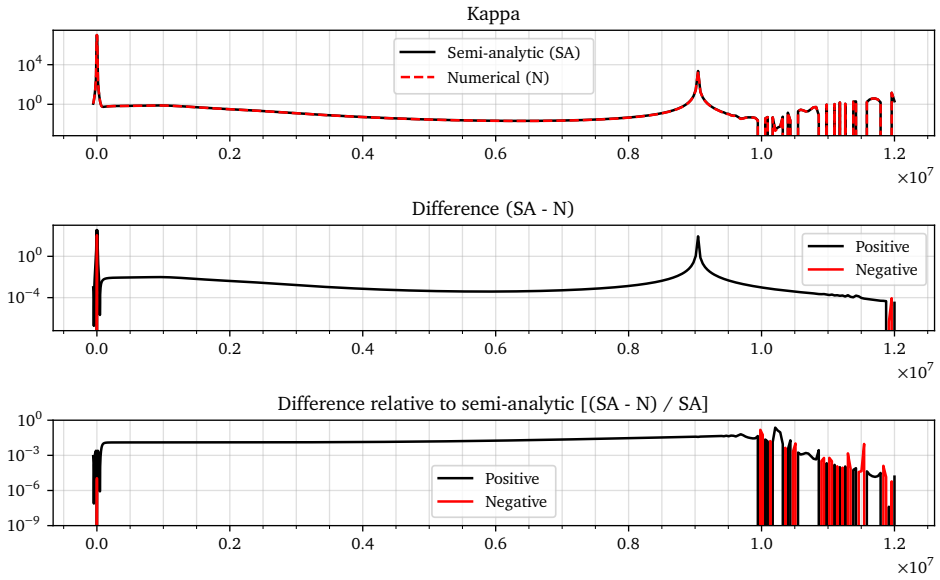
(a)

Figure 4.4: Comparison between the semi-analytic implementation and the numerical implementation of the IS spectrum calculation. Here, $N_y = 8 \times 10^4$ and $N_v = 4 \times 10^4$. **(a)** show the spectra from a Maxwellian distribution. (Continues on the next page.)

personal communication). The suprathermal distribution covered the interval from $E = 1\,\text{eV}$ to $E = 110\,\text{eV}$ and was interpolated to cover energies down to $E = 0\,\text{eV}$. Interpolation was carried out using the `interp` algorithm provided by `numpy` with the default setting, which give the value at $E = 1\,\text{eV}$ to all samples in the region $E = [0, 1)\,\text{eV}$. This suprathermal distribution was added to a Maxwellian distribution representing thermal electrons, implying an assumption of a superposition property to the distributions. The result of the summation of the thermal distribution with the suprathermal distribution is presented in fig. 4.3 as the blue solid line labelled $f_{0,\text{M}} + f_{0,\text{S}}$, where the Maxwellian distribution for the thermal electrons is shown by the orange "dash, dot" line labelled $f_{0,\text{M}}$, and the suprathermal distribution is shown by the green "dash, dot, dot" line labelled $f_{0,\text{S}}$.

4.3 Testing the numerical precision

To test the precision of the numerical implementation based on eq. (3.24), both the Maxwellian and the kappa distribution was included in the form they are given in eqs. (3.1) and (3.2). This was done to be able to compare with the semi-



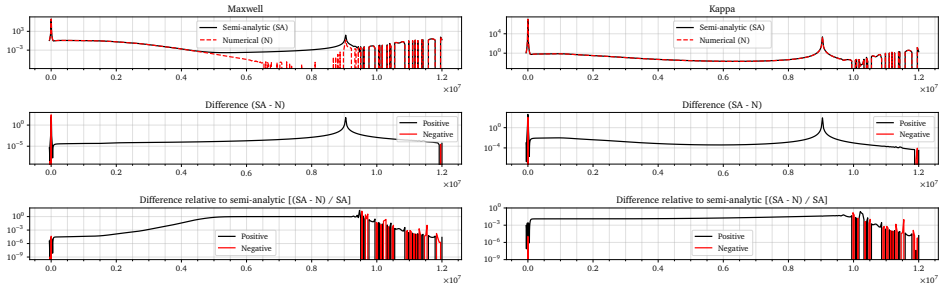
(b)

Figure 4.4: (Continued.) Comparison between the semi-analytic implementation and the numerical implementation of the IS spectrum calculation. Here, $N_y = 8 \times 10^4$ and $N_v = 4 \times 10^4$. **(b)** show the spectra from a kappa distribution where $\kappa = 3$.

analytic implementations based on eqs. (2.59) and (3.5) for the Maxwellian distribution and the kappa distribution, respectively. Figure 4.4 show IS spectra from a Maxwellian distribution (fig. 4.4a) and a kappa distribution (fig. 4.4b). In figs. 4.4a and 4.4b, the top panel show the spectra obtained from the two implementations plotted on top of each other, the second panel show the difference between the semi-analytic and the numerical implementation, while the third panel show the difference between the implementations normalized by the spectrum from the semi-analytic implementation.

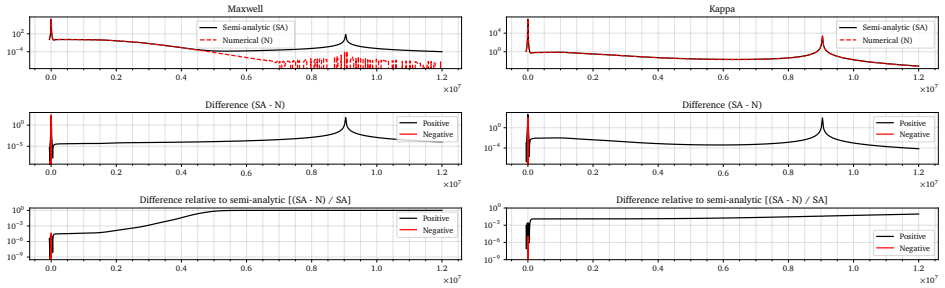
Figure 4.4 was made using $N_y = 8 \times 10^4$ samples in the Gordeyev integral and $N_v = 4 \times 10^4$ samples in the velocity integral. From fig. 4.4a it is clear that the precision at frequencies above 6 MHz for the numerical implementation is poor, while the calculated spectra for the kappa distribution in fig. 4.4b show similar results for the two implementations up to about 9 MHz. The difference between the implementations, seen in panel two, is larger around the ion line and plasma line, but their relative difference is almost constant, suggesting that the general shape of the spectrum is preserved from the semi-analytic to the numerical implementation.

Increasing the number of samples in the velocity integral to $N_v = 4 \times 10^5$



(a) **(b)**

Figure 4.5: Comparison between the semi-analytical implementation and the numerical implementation of the IS spectrum calculation. Here, $N_y = 8 \times 10^4$ and $N_v = 4 \times 10^5$.



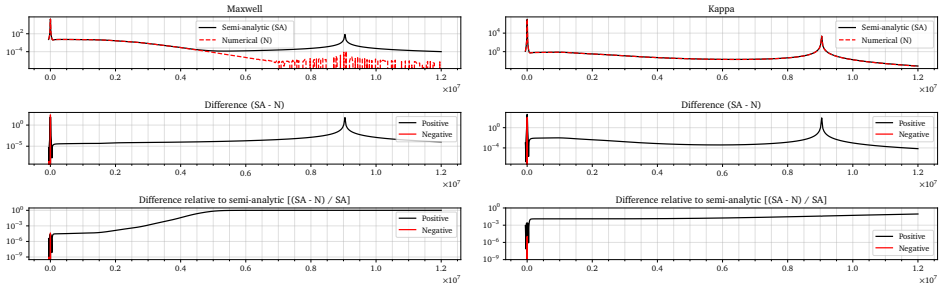
(a) **(b)**

Figure 4.6: Comparison between the semi-analytical implementation and the numerical implementation of the IS spectrum calculation. Here, $N_y = 8 \times 10^5$ and $N_v = 4 \times 10^4$.

did not do much of a difference. Figure 4.5 show the same comparison as fig. 4.4, but with $N_v = 4 \times 10^5$ samples in the velocity integral instead of $N_v = 4 \times 10^4$ samples as in fig. 4.4. The figures are almost indistinguishable when using either of the two sample sizes, suggesting that the sampling of velocity is good enough with $N_v = 4 \times 10^4$ samples and that the reason for the poor numerical precision in fig. 4.4 was not caused by the value of N_v.

In fig. 4.6, the sampling of the velocity was reset down to $N_v = 4 \times 10^4$, while the sampling of y in the Gordeyev integral was increased to $N_y = 8 \times 10^5$ from $N_y = 8 \times 10^4$. This change significantly improved the accuracy of the spectra when using a kappa distribution (i.e., from fig. 4.5b to fig. 4.6b). The precision of the semi-analytic implementation using a Maxwellian distribution was also significantly improved from fig. 4.5a to fig. 4.6a, while the numerical implementation still had poor precision above 6 MHz.

When the sampling of the velocity was again set to $N_v = 4 \times 10^5$, shown in



(a) **(b)**

Figure 4.7: Comparison between the semi-analytical implementation and the numerical implementation of the IS spectrum calculation. Here, $N_y = 8 \times 10^5$ and $N_v = 4 \times 10^5$.

fig. 4.7, the spectra did not change significantly from fig. 4.6. This strengthens the idea that the sampling of velocity in the velocity integral (eq. (4.4)) is sufficient for $N_v = 4 \times 10^4$, and that for frequency $f < 9.5\,\text{MHz}$, $N_y = 8 \times 10^4$ is sufficient (see for example the black solid line for the semi-analytic implementation from the Maxwellian in the top panel of fig. 4.5a). $N_y = 8 \times 10^5$ yields good results up to at least frequency $f = 12\,\text{MHz}$. The spectra calculated using the numerical implementation with a Maxwellian distribution was, nevertheless, still poor at frequencies larger than 6 MHz when using $N_v = 4 \times 10^5$ and $N_y = 8 \times 10^5$.

Since the calculation of the spectra with a kappa distribution result in similar plots from the semi-analytic and numerical implementations given high enough N_y, and we have seen that the sampling of velocity does not yield significant improvements for $N_v > 4 \times 10^4$, a potential reason for the poor results obtained with the Maxwellian distribution lies in the decimal precision. Figure 3.1 show that the magnitude of the kappa distributions in the high-energy tail is many orders higher than the magnitude of the Maxwellian distribution in the high-energy tail. Already at $E = 10\,\text{eV}$, the magnitude of the Maxwellian is about 1×10^{-40} times any of the kappa distributions presented. Also, the kappa distribution for kappa index $\kappa < 8$ never reach a magnitude of less than 1×10^{-34} on the whole energy range up to $E = 110\,\text{eV}$.

To see if the decimal precision is the issue, the upper limit, v_{max}, was lowered to $v_{\text{max}} = 2 \times 10^6\,\text{ms}^{-1}$ from $v_{\text{max}} = 6 \times 10^6\,\text{ms}^{-1}$, where $v = 2 \times 10^6\,\text{ms}^{-1}$ give $E \approx 11.4\,\text{eV}$. This was done to force the magnitude of the distribution functions at velocities higher than v_{max} to be equal to zero. Reducing the upper boundary will increase the sampling on the remaining velocity interval, but as we have seen, increasing the sampling above $N_v = 4 \times 10^4$ do not provide a significant improvement.

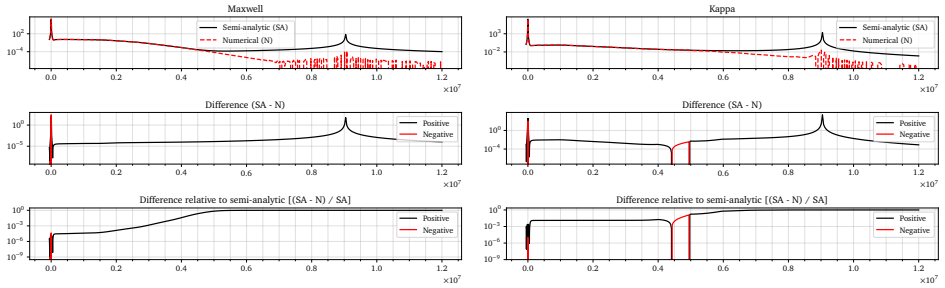


Figure 4.8: Comparison between the semi-analytical implementation and the numerical implementation of the IS spectrum calculation for low upper boundary in the velocity integral. Here, $N_y = 8 \times 10^4$, $N_v = 4 \times 10^4$ and $v_{\max} = 2 \times 10^6\,\mathrm{ms}^{-1}$.

Figure 4.8 show a comparison between the semi-analytic implementation and numerical implementation using the Maxwellian distribution and the kappa distribution. As expected, the spectra from the Maxwellian distribution obtained by the numerical implementation is not significantly changed, while the equivalent spectra from the kappa distribution was much worse compared to the case of using $v_{\max} = 6 \times 10^6\,\mathrm{ms}^{-1}$ as the upper boundary. Also, the shape of the spectra from the two distributions obtained by the numerical implementation is similar in shape when using an upper boundary of $v_{\max} = 2 \times 10^6\,\mathrm{ms}^{-1}$, indicating that the decimal precision is indeed the cause of the poor results obtained by the numerical implementation for the Maxwellian distribution.

Increasing the decimal precision is therefore important when the magnitude of the distribution function is small, and one should consider using for example the `mpmath` Python library or similar to improve the decimal precision when working with distribution functions that get vanishingly small at high phase velocity/energy. The `mpmath` library does not, however, include the Simpson's rule for integration, but a quadrature algorithm that accepts a functional as its argument rather than an array. This significantly slows down the calculation of the integrals found in the susceptibility functions but with the same numerical precision.

To make sure the different distribution functions used in the velocity integral (eq. (4.4)) was correctly implemented, a test was made. The test is listed in appendix A.9 in the `TestVDF` class (line 83), and it takes advantage of what is stated in section 3.1, namely that the integral of the distribution function over

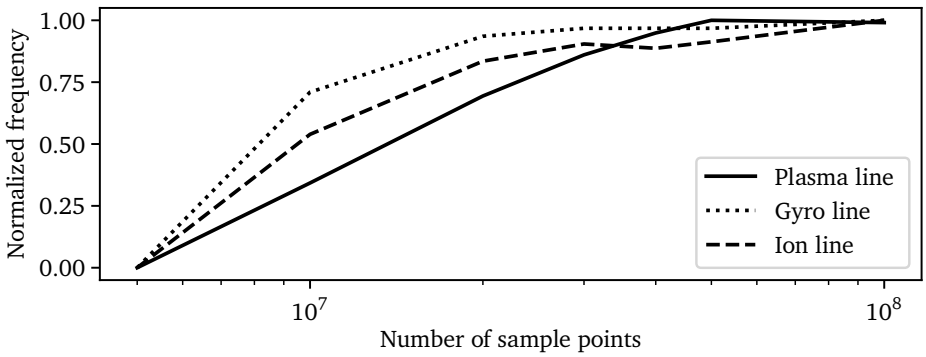


Figure 4.9: Visual of how the peak frequencies changed as a function of number of sampling points, N ($= N_f = N_y$). The lines show the peak frequency of the plasma line (solid), gyro line (dotted) and ion line (dashed) along the y axis against number of sampling points on the x axis. The lines have been shifted to zero and normalized.

velocity space should be equal to one:

$$\int f_0 \mathrm{d}^3\boldsymbol{v} = 1. \tag{4.6}$$

The test compare the result from the integral to the known result, 1, and the test passes if the value of the integral is equal to 1 to six decimal places.

4.4 Evaluating the Gordeyev integral using the chirp z-transform

Figures 4.4 to 4.7 shows that increasing the sampling of the y parameter of the Gordeyev integral was an efficient way of increasing the precision in the calculation of the IS spectrum. The chirp z-transform is an alternative way of solving the Gordeyev integral using the fast Fourier transform (FFT). The Gordeyev integral is rewritten with a finite upper boundary along the same lines as for the Simpson's rule algorithm, but then further rewritten as a finite sum and evaluated using the chirp z-transform algorithm described by Li et al. (1991). This algorithm is computationally much more efficient than the method of using the Simpson's rule and it is therefore possible to increase the number of samples in the Gordeyev integral, N_y, and along the frequency axis, N_f, by orders of magnitude.

Unfortunately, the chirp z-transform algorithm was found to lead to some artefacts where the number of sampling points would influence the frequency of

the peaks in the spectrum. This is shown in fig. 4.9, where the peak frequencies of the upshifted ion line, gyro line and plasma line are plotted against number of sampling points used in the calculation. The number of samples along the frequency axis, N_f, and in the Gordeyev integral, N_y, was equal in all numerical tests, i.e., $N = N_f = N_y$. The frequency lines have all been shifted to start at zero and then normalized to make the lines span the same range. In reality, however, the ion line is in the kHz range while the plasma line lie in the MHz range, with the gyro line in between on the order of 1×10^5 Hz. Because of these numerical errors, the chirp z-transform was put aside.

5

Results from model calculations of IS spectra

In the preceding chapters the theory and computational model for calculating the IS spectrum from radar observations at oblique angles to the magnetic field was developed. The derivation included electron distributions described by a Maxwellian distribution, kappa distributions and arbitrary isotropic distributions. The motivation behind this was that the theory and program should be able to reproduce real observations in greater detail, thus enabling us to derive plasma parameters in more interesting plasmas in general and more turbulent plasmas in particular, and to examine observed phenomena both analytically and numerically. This chapter will present the results achieved by the numerical model and compare the spectra calculated from different distributions. In all calculations, a Maxwellian distribution was used to represent the ions.

5.1 Spectra from Maxwellian and kappa distributions

When moving to a kappa distribution from a Maxwellian distribution, we move to a representation of a population that has larger fluxes in the high-energy tail. As a result of this increased high-energy electron population, the Landau

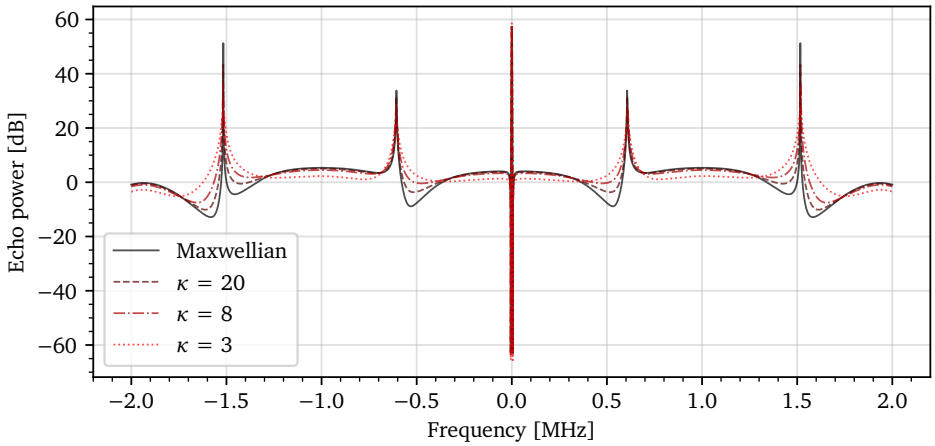


Figure 5.1: IS spectra for a Maxwellian distribution and three kappa distributions, with $\kappa = \{20, 8, 3\}$.

damping at large phase velocities, corresponding to large frequency shifts, is increased. This will in turn widen the plasma line, similar to how it is broadened in the kinetic description compared to the fluid description where Landau damping is not considered.

Figure 5.1 shows the result of plotting the IS spectrum from a Maxwellian distribution and different kappa distributions, with the plasma parameters used in the numerical model presented in table 5.1. The plot contains three pairs of peaks; one pair at such low frequency that they look like a single peak at zero frequency referred to as the ion line, one pair at ±0.6 MHz referred

Table 5.1: Plasma parameters for fig. 5.1. f_r is the radar frequency, n_0 is the electron number density, B is the magnetic field strength, m_i is the ion mass, ν is collision frequency, T is temperature and θ is the angle between the radar beam and the magnetic field line.

Parameter	Unit	Value
f_r	[Hz]	430×10^6
n_0	[m^{-3}]	2.0×10^{10}
B	[T]	3.5×10^{-5}
m_i	[amu]	29
ν_e	[Hz]	0
ν_i	[Hz]	0
T_e	[K]	200
T_i	[K]	200
θ	[°]	135.0

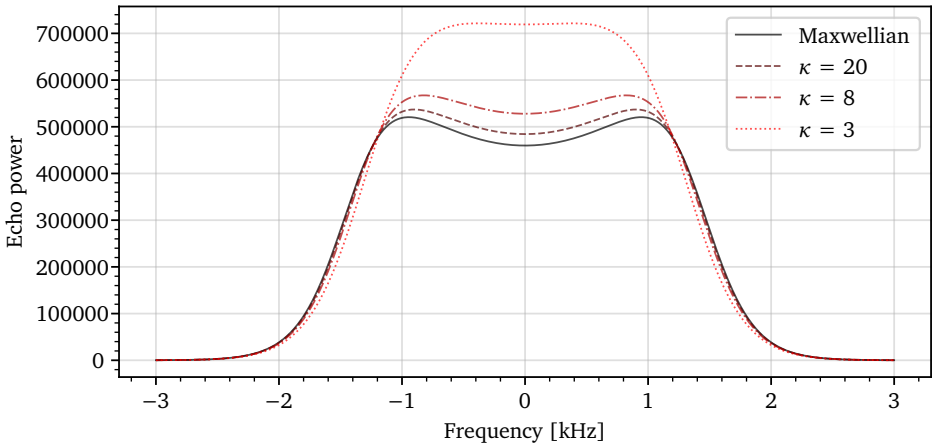


Figure 5.2: Ion line of the IS spectrum, calculated using a Maxwellian distribution and different kappa distributions, where $\kappa = \{20, 8, 3\}$.

to as the gyro line and one pair at ±1.5 MHz referred to as the plasma line. The two latter pairs are due to backscatter from plasma waves, and from the power spectrum in eq. (2.60) it is evident that when the denominator decrease, the power density increase, thus the peaks appear where χ_e approach zero. Similarly, the ion line appear where χ_i approach zero.

It was stated in section 3.1 that as the kappa index increase, the kappa distribution approach the Maxwellian distribution. Therefore, it is expected that the IS spectrum calculated from a kappa distribution with relatively high kappa index is akin to the spectrum calculated from a Maxwellian distribution. In fig. 5.1, the solid black line show the IS spectrum from a Maxwellian distribution, while the dashed dark red line show the spectrum from a kappa distribution with $\kappa = 20$. Even for such relatively small kappas, the deviation from the Maxwellian spectrum is small. The gyro lines and plasma lines in the spectrum from the kappa distribution can be seen to be slightly wider, with shoulders containing more power, while the peak frequency power of the gyro lines and plasma lines are greater in the spectrum from the Maxwellian distribution.

The "dash-dot" line in fig. 5.1 is the IS spectrum from a kappa distribution with $\kappa = 8$ and the dotted line is the IS spectrum from a kappa distribution with $\kappa = 3$. Here, the effect of the high-energy tail become more distinct as the kappa index decreases, which is seen in that the gyro lines and plasma lines are further widened with more power in the shoulders, in addition to that the peak frequencies decrease in power.

Figure 5.2 is a closer look at the low frequency part of fig. 5.1—that is, the same plasma parameters presented in table 5.1 apply—known as the ion line. Three

features are of interest in the figure, which is that the peak power is increasing with decreasing kappa index, the resonance frequencies where the peaks are found are downshifted as the kappa index is decreased and the valley between the resonance frequencies is decreasing with decreasing kappa index.

Going back to fig. 3.1a, the magnitude of the kappa distributions is seen to increase in the low-energy region as the kappa index decrease. This means more electrons, hence more scatterers, are present at the phase velocity of the ion acoustic wave, leading to more received power (Saito et al., 2000). The ion and electron temperature was set equal, $T_e = T_i = 200\,\mathrm{K}$, and in such plasmas, ion acoustic waves are heavily Landau-damped (Chen, 1984). An increased Landau damping is related to the slope of the distributions. It is clear from fig. 3.1 that the kappa distributions have slopes that get steeper in the low-energy regions with decreasing kappa index, thus leading to an increased Landau damping (Chen, 1984). When the ion acoustic waves are damped, the valley between the peaks is reduced. This was shown quantitatively by Saito et al. (2000), who numerically solved the dispersion relation for electrostatic waves from Thorne and Summers (1991). By solving the dispersion relation, Saito et al. (2000) found that the frequency of the ion acoustic wave is downshifted from the Maxwellian electron distribution to the kappa distribution, and that damping rates are increased for the same change of electron distribution, in accordance with fig. 5.2.

5.2 The plasma lines

Figure 5.3 look at the peak at the highest frequency, the plasma line, with plasma parameters presented in table 5.2. It is clear from fig. 5.3 that the resonance frequency of the plasma wave is downshifted as the kappa index is decreased. In fig. 5.1 the plasma lines was seen to be getting wider due to increased Landau damping caused by the larger population of electrons at high phase velocity. The downshift of the resonance frequency of the plasma line can also be explained by the change of the electron velocity distribution, since it causes the theoretical plasma resonance frequency to change. The real part of the plasma resonance frequency is defined as

$$\omega_{\Re,e} = \left[\omega_{pe}^2(1 + 3k^2\lambda_D^2) + \Omega_e^2 \sin^2\theta\right]^{1/2}. \tag{5.1}$$

This is dependent on the Debye length, and in eq. (3.9) a Debye length for the kappa distribution that decrease as the kappa index decreases was introduced. From this, it is consistent that the plasma resonance frequency is downshifted.

Figure 5.4 shows the plasma line obtained from a Maxwellian distribution

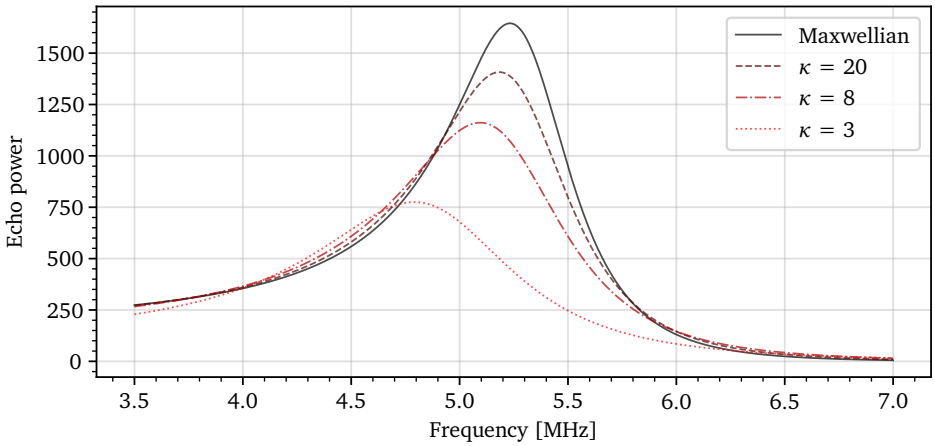


Figure 5.3: Plasma line of the IS spectrum, calculated using a Maxwellian distribution and different kappa distributions, where $\kappa = \{20, 8, 3\}$.

and two kappa distributions, with kappa indices of 20 and 3. The plasma parameters are the same as in fig. 5.3 and given in table 5.2, except from the electron temperature which is changed from 2000 K to 10 000 K in steps of 1000 K. We notice how the width and power changes. For small kappa indices, the peak of the plasma line from the kappa distribution is strongly damped at low temperature compared to the peak associated with the Maxwellian distribution. Then, as temperature increases, the damping of the plasma line from the Maxwellian distribution become similar to the damping seen in the plasma line for both kappa distributions. This is also reported by Saito et al. (2000), which points to the Debye length to explain this phenomenon. When the

Table 5.2: Plasma parameters for fig. 5.3. f_r is the radar frequency, n_0 is the electron number density, B is the magnetic field strength, m_i is the ion mass, ν is collision frequency, T is temperature and θ is the angle between the radar beam and the magnetic field line.

Parameter	Unit	Value
f_r	[Hz]	933×10^6
n_0	[m^{-3}]	2.0×10^{11}
B	[T]	5×10^{-5}
m_i	[amu]	16
ν_e	[Hz]	0
ν_i	[Hz]	0
T_e	[K]	5000
T_i	[K]	2000
θ	[°]	180.0

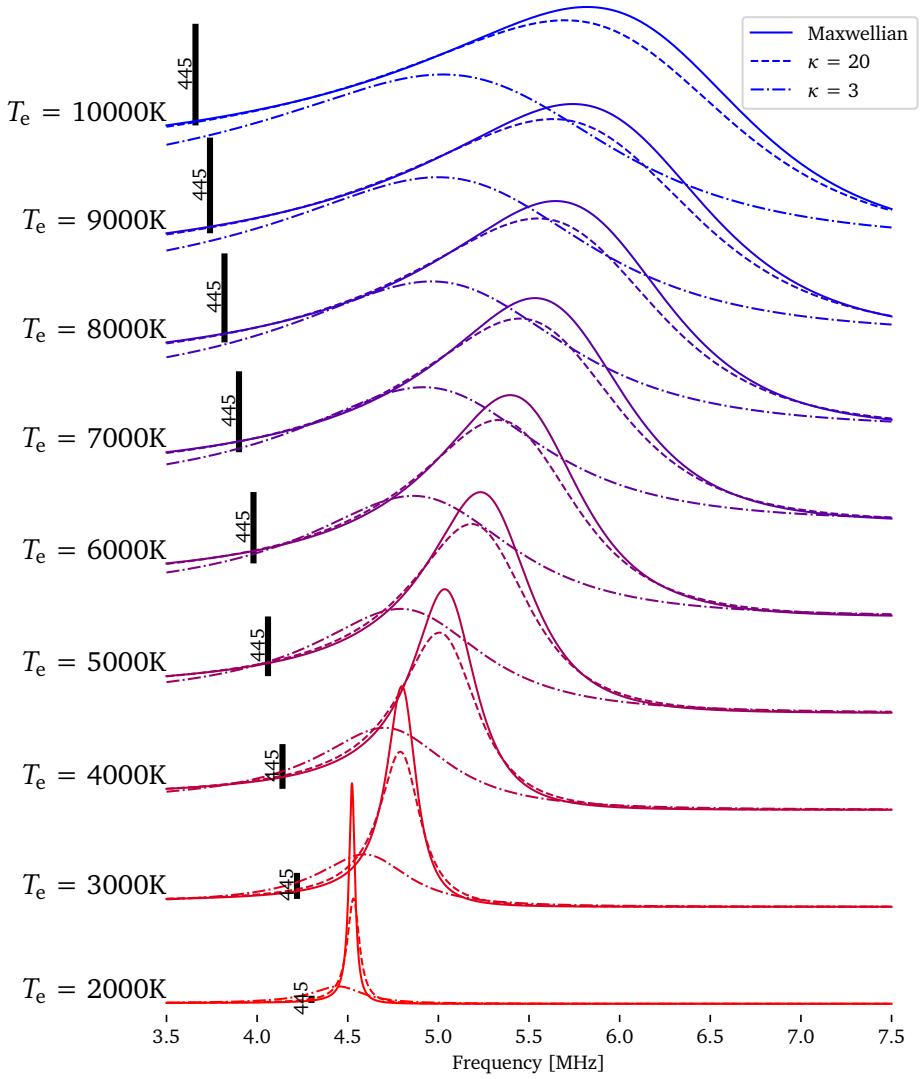


Figure 5.4: Plasma line with changing electron temperature. The electron temperature was changed from 2000 K to 10 000 K in increments of 1000 K. The y axis is a linear scale representing the returned power from the scattering, and the black bar represent equal power at the different temperatures, spanning 445 power-units.

electron temperature is small, the assumption of weak Landau damping is valid, i.e. $k^2\lambda_D^2 \ll 1$. In such a situation it is expected of the enhancement in electron population at high phase velocities, represented by a kappa distribution, to result in a significant change in the width of the plasma line compared to a spectrum from a Maxwellian distribution. But when the electron temperature is increased, the expression $k^2\lambda_D^2$ approaches unity and the assumption of weak

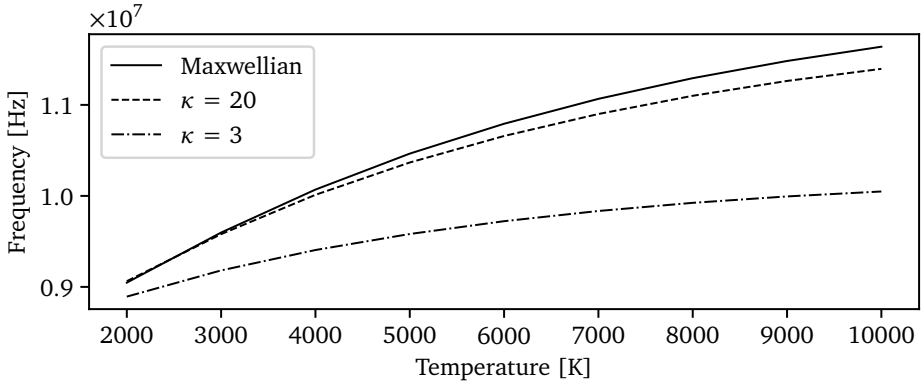


Figure 5.5: Difference between up- and downshifted plasma line peak frequency. The peak frequencies are the same as presented in fig. 5.4.

damping is no longer valid, resulting in a wide plasma line. With increased damping and a plasma line that get wider, power is distributed to the shoulders from the peak and the peak power decrease as seen in fig. 5.4.

In real measurements, it is easier to accurately measure the resonance frequency of the plasma line rather than the correct received power or other measures that give the shape of the plasma line due to receiver gains and system losses (Nicolls et al., 2006). Because of this, the resonance frequency of the plasma line is important to obtain information about the plasma line, and a much used parameter is the difference between the up- and downshifted resonance frequencies. This parameter is given as

$$\Delta f_{\mathfrak{R}} = f_{\mathfrak{R}+} - f_{\mathfrak{R}-}, \tag{5.2}$$

and is plotted in fig. 5.5 for the peaks found in fig. 5.4. As seen in fig. 5.4, the resonance frequency is increased as the electron temperature increase. Figure 5.5 present a clearer view of how the frequency changes with temperature when the IS spectrum is calculated from the three distributions used in fig. 5.4. All three plasma resonance frequency lines plotted in fig. 5.5 change as a function of temperature, and they do so with similar shape across all three distributions.

While fig. 5.5 show the difference between the up- and downshifted resonance frequencies, the sum is also a widely used parameter to be able to look at the asymmetry between the frequencies. The up- and downshifted frequencies taking part in eq. (5.2) are generally not the same, and the value of the wave vector k is obtained through the mean of the transmitted and received frequency

(Showen, 1979; Nicolls et al., 2006), i.e.

$$k_{\pm} = \frac{2\pi}{c}[f_{\mathrm{r}} + (f_{\mathrm{r}} \pm f_{\mathfrak{R}})] \tag{5.3}$$

assuming $f_{\mathfrak{R}}/f_{\mathrm{r}} \ll 1$ and where ± refer to the up- and downshifted frequencies. For the Arecibo radar the asymmetry parameter ($f_{\mathfrak{R}+} + f_{\mathfrak{R}-}$) is on the order of kHz for typical plasma parameters (Showen, 1979).

The frequency difference parameter in eq. (5.2) was studied by Djuth et al. (2018), with particular emphasis on the altitude region where the suprathermal electron distribution contain structure, $E = 14\,\mathrm{eV}$ to $E = 27\,\mathrm{eV}$. In addition, they looked at the power received from the plasma line and noted that there were good agreement between the structure observed in the received power as a function of aspect angle, and the spectral structure in the ionosphere for the energy interval 14 eV to 27 eV. They were able to derive a pitch angle dependence between the energy corresponding to a spectral structure and the structures seen in the plasma line power measurement:

$$E(\theta) = D\cos(\theta)^{1.94} \tag{5.4}$$

where D is a normalization constant. Djuth et al. (2018) argued that the pitch angle, referring to the angle between the velocity vector of the electrons to the magnetic field line, would be the same as the aspect angle, hence the energy was written as a function of aspect angle. Djuth et al. (2018) also provide a pitch angle formula for the resonance frequency of the plasma line:

$$f_{\mathfrak{R}}(\theta) = A\cos(\theta)^{0.97} \tag{5.5}$$

where A is a normalization constant.

5.3 Plasma line power structures at Arecibo Observatory

5.3.1 Measurements

Similar observations to those of Djuth et al. (2018) has been made by Vierinen (personal communication) of a plasma line power dependence on aspect angle and altitude. These measurements were made at the Arecibo Observatory during evening time, on 17 March 2015 between 16 (20) and 18 (22) local time (UT), and are presented in fig. 5.6. The method used to do the measurements was with the coded long-pulse technique, where the radar frequency was set to 430 MHz and with transmit pulses of length 440 µs with bits of 2 µs

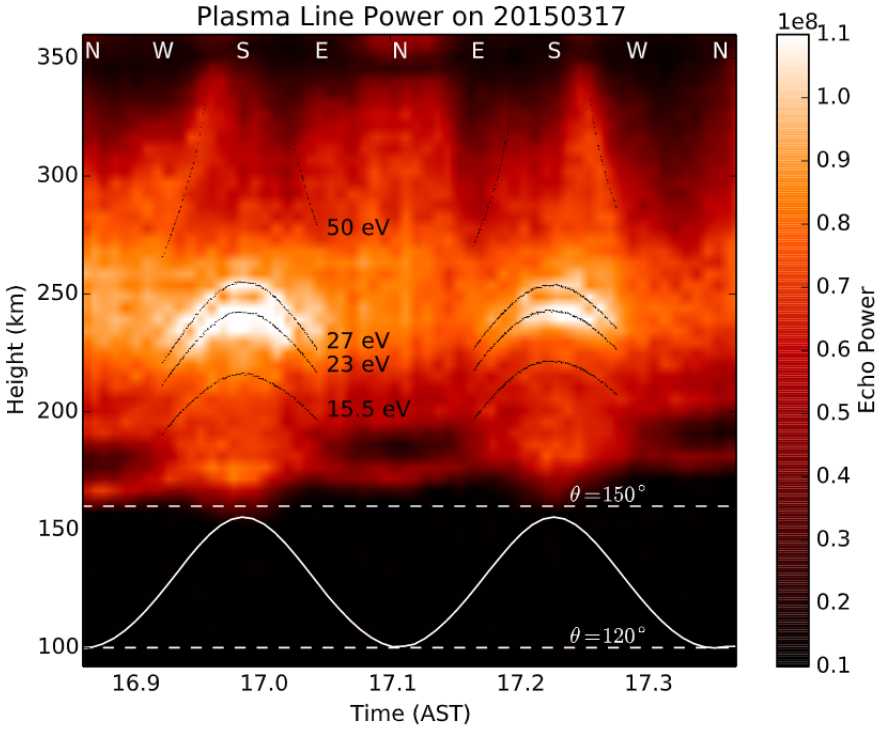


Figure 5.6: Measurement of plasma line power as a function of aspect angle, time and altitude, at the Arecibo Observatory. (Vierinen, personal communication.)

length (Vierinen, personal communication). A more detailed description of the measurement technique has been given by Djuth et al. (1994). The Arecibo Observatory is located in Arecibo, Puerto Rico, with coordinates 18°20′39″ N, 66°45′10″ W, and has got a 350 m diameter dish (LaLonde, 1974), with its magnetic conjugate point located near Mar del Plata, Argentina (Djuth et al., 2018). The zenith angle of the antenna during the experiment was 15°, and the antenna was rotated 720° in azimuth during the experiment, hence the aspect angle variation between $\theta = 120°$ and $\theta = 150°$ seen at the bottom of fig. 5.6.

Figure 5.6 show measured plasma line echo power as a function of altitude and aspect angle changing with time. The echo power is seen to change with both altitude and aspect angle and overlaid are black isolines showing constant energy calculated according to the equation

$$E = \frac{1}{2} m_e \left(\frac{f_{\mathfrak{R}}}{\cos\theta} \frac{\lambda}{2} \right)^2 . \tag{5.6}$$

Plasma wave phase velocity is defined as $v_\phi = f_{\mathfrak{R}} \lambda / 2$ (Yngvesson and Perkins,

1968; Djuth et al., 2018), where f_R is the resonance frequency of the plasma wave measured by the radar, λ is the wavelength of the radar beam and the factor 1/2 on the radar wavelength is the Bragg condition (e.g. Kudeki and Milla (2011) or Djuth et al. (2018)). The classical energy related to this phase velocity is then $E = m_e v_\phi^2/2$. Assuming the main contributing factor to the plasma wave resonance frequency to come from electrons moving close to parallel to the magnetic field line, the measured frequency/phase velocity is a decomposition of the resonance frequency and a factor $1/\cos\theta$ is obtained. E in eq. (5.6) is therefore the energy of an electron moving along the magnetic field line with the plasma wave phase velocity.

The plasma line intensity is usually represented as a plasma line temperature, and in presence of suprathermal electrons but with no ambient magnetic field the temperature of the plasma line is given as (Perkins and Salpeter, 1965; Yngvesson and Perkins, 1968)

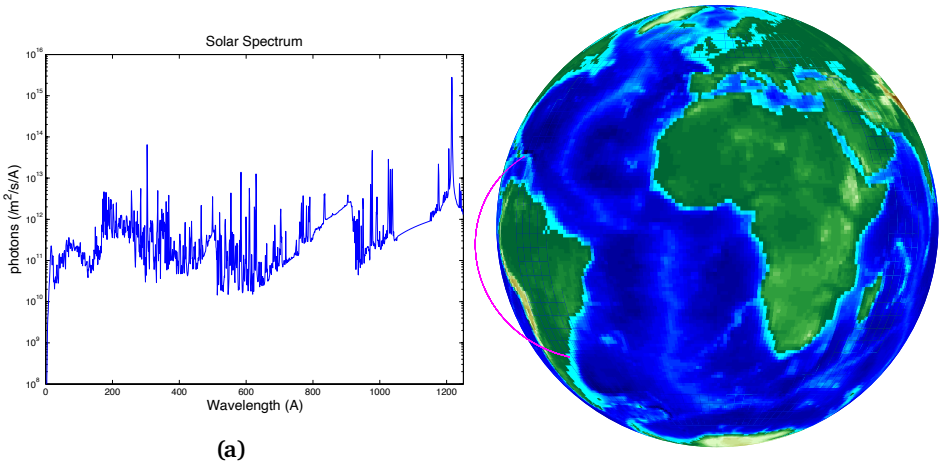
$$T_P(v_\phi) = T_e \frac{f_M(v_\phi) + f_S(v_\phi) + \chi_{coll}}{f_M(v_\phi) - k_B T_e \frac{d}{dE} f_S(v_\phi) + \chi_{coll}} \tag{5.7}$$

where T_e is the thermal electron temperature, f_M is the isotropic Maxwellian distribution, f_S is the isotropic distribution for the suprathermal electrons and χ_{coll} represent electron-ion collisional excitation and damping (Yngvesson and Perkins, 1968). For a magnetized plasma the thermal distribution and corresponding thermal Landau damping need to be modified (Yngvesson and Perkins, 1968; Fredriksen et al., 1992).

When large photoelectron fluxes are present, the term $-k_B T_e \frac{d}{dE} f_S(v_\phi)$ dominates the plasma wave damping in eq. (5.7) (Djuth et al., 2018). Because of this, the enhanced power seen in fig. 5.6 was assumed to be due to features in the suprathermal distribution originating from spectral features in the solar UV spectrum, and specific constant energies associated with the features in the solar spectrum was used to mark the isolines in fig. 5.6.

5.3.2 Comparison with numerical model

An electron distribution calculated for photoelectron production above Arecibo and the magnetic conjugate ionosphere from solar UV spectra was used to reproduce the measurements in fig. 5.6. The electron distribution was calculated with the AURORA electron transport code which used the solar spectrum shown in fig. 5.7a and calculated the electron transport along the magnetic field line shown in fig. 5.7b as the magenta line to the left in the figure. The solar spectrum and magnetic field line in fig. 5.7 are from 17 March 2015, at 12:00 UT, the same day the measurement in fig. 5.6 was made. An example of



(a) **(b)**

Figure 5.7: Input to electron transport code. **(a)** show the spectrum between ~0 Å and ~1250 Å of solar UV flux and **(b)** show the magnetic field line. (Gustavsson, personal communication.)

a calculated distribution for a specific altitude averaged over all pitch angles can be seen in fig. 5.8, where it is compared to the Maxwellian distribution and the kappa distribution with $\kappa = 3$.

In section 4.3, the precision of the numerical implementation was tested against the semi-analytic implementation, and the Maxwellian distribution was found to yield poor results in the high frequency part of the IS spectrum, shown in fig. 4.8. Figure 5.8 show that the calculated electron distribution used in the numerical implementation has, in the high-energy region, magnitude comparable to the kappa distribution with $\kappa = 3$, and it is therefore expected that the calculated IS spectrum from the program yields reasonable results. The level of precision was the same as used in fig. 4.4, i.e., $N_v = 4 \times 10^4$ and $N_y = 8 \times 10^4$, since the spectrum was calculated for frequency $f < 9.5\,\text{MHz}$.

The plots made to reproduce the measurement in fig. 5.6 was obtained through a different cross-section through parameter space. Temporal variation was assumed to be negligible over the approximately five minutes the experiment lasted, thus only aspect angle was changed along the x axis. Also, the distribution function that was used was calculated for one specific altitude/height, and instead the electron number density was varied to mimic altitude variation along the y axis. In the bottomside ionosphere (below the F region peak at about 300 km altitude), the electron number density is increasing with altitude (Djuth et al., 2018), thus making it a comparable cross-section.

Figure 5.9 was made with the parameters presented in table 5.3. The figure

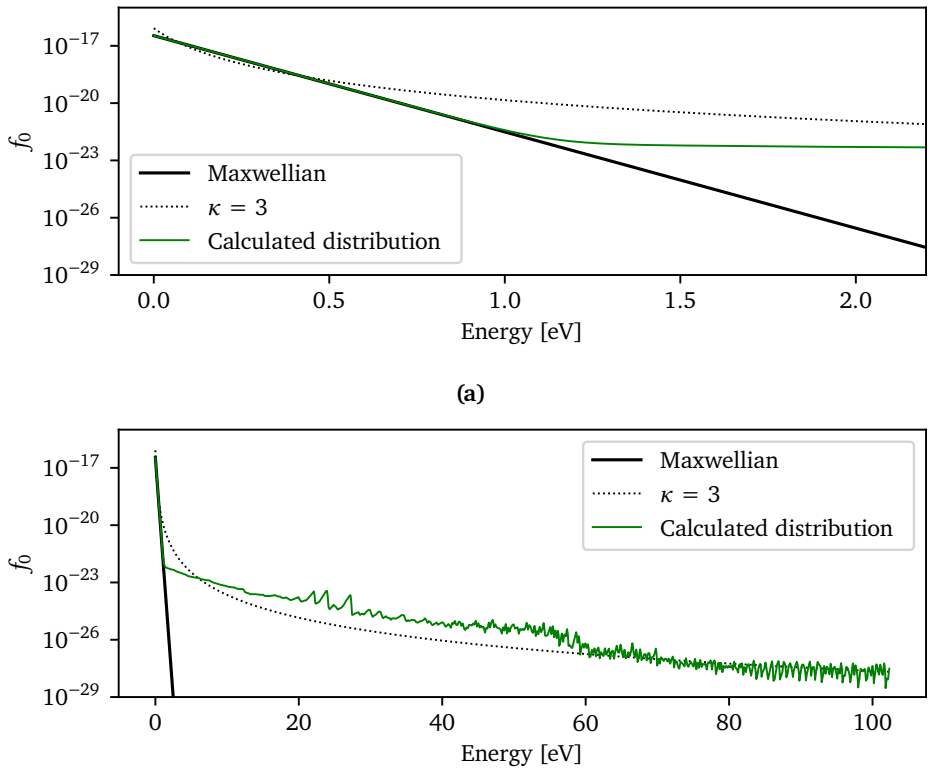


Figure 5.8: Calculated distribution compared to Maxwellian and kappa distribution ($\kappa = 3$). The calculated suprathermal distribution is the same as the one used in fig. 5.11, shown in fig. 5.12.

shows plasma line power as a function of electron number density, n_e, along the y axis and as a function of aspect angle, θ, along the x axis. The power of the plasma line was calculated through a Lorentzian fit around the plasma line peak frequency, with a total width of 1 kHz. The green shaded area on top of the surface plot in fig. 5.9 represent plasma line peak frequencies that map to the energy intervals $E = (15.58, 18.42)$ eV or $E = (22.47, 23.75)$ eV, calculated according to eq. (5.6).

The energy intervals was chosen because the distribution function that was used in the calculation had large positive slopes approximately at these two energy intervals. Figure 5.10 shows the distribution that represent the suprathermal electrons, and in the enlarged box are the two enhancements that was believed to cause the structures seen in fig. 5.9, marked with green shading. The lower shaded area in fig. 5.9, marked by the label "1", correspond to the energy interval labelled "1" in fig. 5.10, and similarly for the label "2".

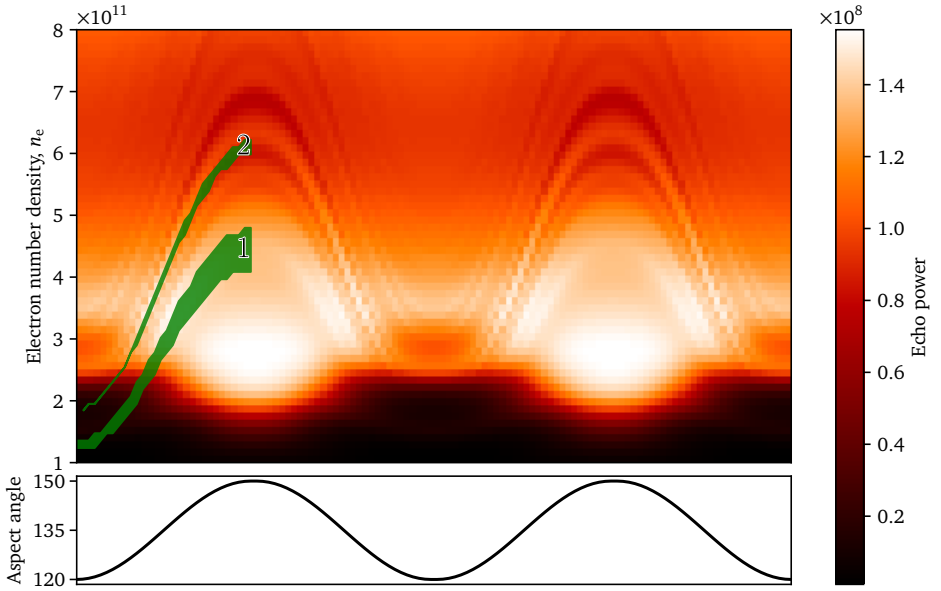


Figure 5.9: Plasma line power as a function of aspect angle, θ, and electron number density, n_e. Table 5.3 give the plasma parameters used in the computation of the IS spectrum, while the green shaded regions represent plasma peak frequencies that map to $E = (15.58, 18.42)$ eV or $E = (22.47, 23.75)$ eV, shown in fig. 5.10. The figure was made by calculating the spectra needed for the left quarter (first quarter of the sine wave), before the data points were mirrored and copied to make the structures clearer.

Figure 5.9 show how the plasma line power enhancements maps nicely to the expected energies. At aspect angles close to $\theta = 135°$ the shaded region fit the

Table 5.3: Plasma parameters for fig. 5.9. f_r is the radar frequency, B is the magnetic field strength, m_i is the ion mass, ν is the collision frequency, T is the temperature and height and ToD is the altitude and time of day corresponding to the calculated suprathermal distribution shown in fig. 5.10.

Parameter	Unit	Value
f_r	[Hz]	430×10^6
B	[T]	$35\,000 \times 10^{-9}$
m_i	[amu]	16
ν_e	[Hz]	100
ν_i	[Hz]	100
T_e	[K]	2000
T_i	[K]	1500
Height	[km]	599
ToD	[UT]	12:00

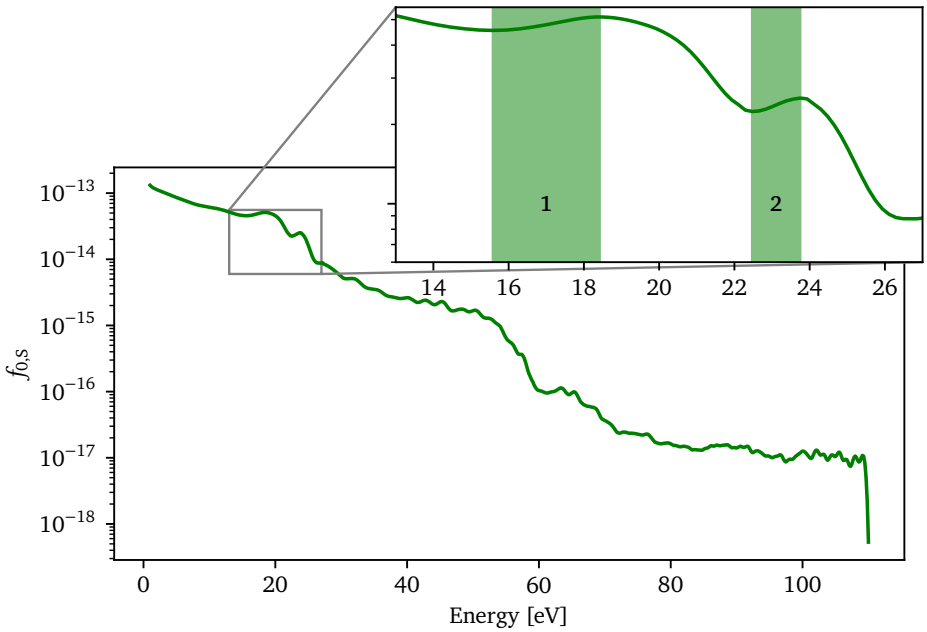


Figure 5.10: Distribution representing the suprathermal electrons in fig. 5.9. The energy intervals that correspond to the dots in fig. 5.9 can be seen as the two bumps where the distribution is enhanced, shown in the enlarged rectangle as the two shaded areas.

plasma line power structures best, lying nearly on top, while for larger aspect angles the shaded region lie slightly below the structures. The green shaded area in fig. 5.9 can be seen to get wider with larger aspect angle, which might be what causes the mapping to seem worse at large aspect angle.

Equation (5.7) for the power of the plasma line is dependent on the distribution for the suprathermal electrons in two ways. In the numerator, the value of the distribution is added, while in the denominator the important term is the derivative. When the distribution contain enhanced features as seen in fig. 5.10 the derivative increases to above zero. This makes the denominator of eq. (5.7) smaller while the ratio increase, leading to increased power. From fig. 5.9 it can be seen that it is the structure labelled “2” that is most prominent and a possible explanation is found in the distribution in fig. 5.10 in combination with eq. (5.7).

Two features in fig. 5.10 of interest are that enhancement “1” is wider than enhancement “2” and that enhancement “2” come right after enhancement “1”. Since enhancement “1” is wider, the magnitude of the derivative is smaller and therefore affect the value in the denominator of eq. (5.7) less. The second point, that enhancement “2” appear right after enhancement “1”, means that

the derivative will change quickly with energy around the energy associated with enhancement "2". This has the effect that the echo power also change in magnitude quickly at this energy, which is consistent with the prominent change in power seen in fig. 5.9 at structure "2". The echo power is dependent on the value of the distribution itself in the numerator, but the echo power in the structures in fig. 5.9 give an indication that the more important term coming from the distribution of the suprathermal electrons is the derivative in the denominator, in accordance with the argument by Djuth et al. (2018).

To further investigate the results from fig. 5.9, suggesting a relation between resonance frequency and energy according to eq. (5.6), the program was run using a different suprathermal distribution with more sharp features. Figure 5.11 shows a similar plot as in fig. 5.9, of plasma line power as a function of electron number density and aspect angle, but now with the plasma parameters given in table 5.4 and with the distribution for the suprathermal electrons shown in fig. 5.12. The peaks are found at higher energies in fig. 5.12 compared to fig. 5.10, and the scale of the electron number density in fig. 5.11 was therefore increased somewhat compared to fig. 5.9.

The energy intervals marked by the shaded areas in fig. 5.12 are covering the whole rising ridge where the derivative is positive, and maps to the shaded structures in fig. 5.11. All three shadings in fig. 5.11 fits very well to the structures of enhanced plasma line power seen in the figure. One can even distinguish the slight increase in power between structure "2" and "3" in fig. 5.11 that most likely come from the small enhancement in the electron distribution at $E \approx 25\,\text{eV}$, seen in fig. 5.12.

Table 5.4: Plasma parameters for fig. 5.11. f_r is the radar frequency, B is the magnetic field strength, m_i is the ion mass, ν is the collision frequency, T is the temperature and height and ToD is the altitude and time of day corresponding to the calculated suprathermal distribution shown in fig. 5.12.

Parameter	Unit	Value
f_r	[Hz]	430×10^6
B	[T]	$35\,000 \times 10^{-9}$
m_i	[amu]	16
ν_e	[Hz]	100
ν_i	[Hz]	100
T_e	[K]	2000
T_i	[K]	1500
Height	[km]	300
ToD	[UT]	12:00

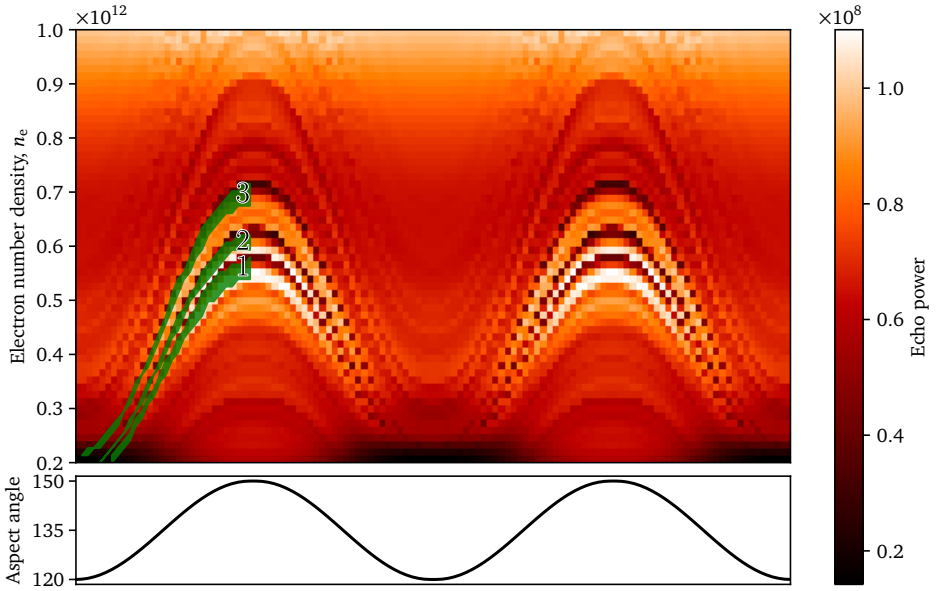


Figure 5.11: Plasma line power as a function of aspect angle, θ, and electron number density, n_e. Table 5.4 give the plasma parameters used in the computation of the IS spectrum, while the green shaded regions represent plasma peak frequencies that map to $E = (20.29, 22.05)$ eV, $E = (22.45, 23.87)$ eV or $E = (25.38, 27.14)$ eV, shown in fig. 5.10. The figure was made by calculating the spectra needed for the left quarter (first quarter of the sine wave), before the data points were mirrored and copied to make the structures clearer.

The shading of the structures in fig. 5.11 also somewhat cover the darker decrease in power on the topside of the structures. This might be due to the shading in fig. 5.12 reaching all the way up to the peak of the electron distribution enhancements. At the peak, the derivative is again changing sign from positive to negative and the power is expected to be reduced. In addition, since the distributions decrease very sharply, the resolution in electron density and aspect angle might not be high enough to capture this sharp change.

Nevertheless, the match between the enhancements of the distribution in fig. 5.12 and the structures seen in fig. 5.11 is good, and give an indication that the explanation provided by eqs. (5.6) and (5.7) is reasonable. The relation shown between the enhancements in the electron distribution and the structures in the plasma line power as a function of number density and aspect angle facilitate for finding the energy of suprathermal electron flux enhancements based on the power returned from the plasma line of the IS spectrum.

The dependence on aspect angle of the plasma line power is also in accordance

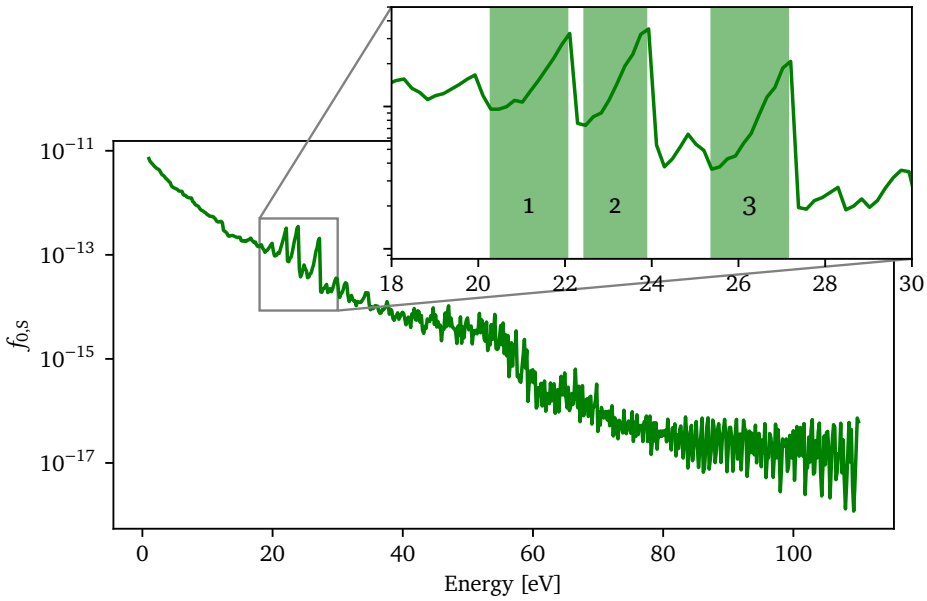


Figure 5.12: Distribution representing the suprathermal electrons in fig. 5.11. The energy intervals that correspond to the dots in fig. 5.11 can be seen as the three bumps where the distribution is enhanced, shown in the enlarged rectangle as the three shaded areas.

with observations made by Fredriksen et al. (1992) with the EISCAT UHF radar. They showed that the received power in the plasma line was reduced as the radar pointing direction was moved away from parallel to the magnetic field line. Due to the structures caused by the enhancements in the distribution function this is not the case at all electron number densities (e.g. at $n_e = 0.4 \times 10^{12}$ in fig. 5.11), but the general trend is that power is reduced when the aspect angle decreases.

5.3.3 Results compared to measurements by Djuth

The energy and frequency formulas presented in eqs. (5.4) and (5.5) that were empirically derived by Djuth et al. (1994) are similar to the formula in eq. (5.6) that was used to trace the plasma line power structures in figs. 5.9 and 5.11. If the aspect angle formula by Djuth et al. (2018) for resonance frequency in eq. (5.5) is substituted into the expression for constant energy given in eq. (5.6), we obtain

$$E = \frac{1}{2} m_e \left(\frac{f_{\text{R}}}{\cos\theta} \frac{\lambda}{2} \right)^2 = \frac{1}{2} m_e \left(\frac{A \cos(\theta)^{0.97}}{\cos\theta} \frac{\lambda}{2} \right)^2 \tag{5.8}$$

or, without the scaling for the angle to the magnetic field:

$$E(\theta) = \frac{1}{2} m_e \left(A \cos(\theta)^{0.97} \frac{\lambda}{2} \right)^2 = D \cos(\theta)^{1.94} \tag{5.9}$$

which is the same as eq. (5.4) that was empirically derived by Djuth et al. (2018). That is, the energy formula derived by Djuth et al. (2018) is changing with aspect angle and the related "phase energy" is the energy for measuring along the field line, i.e., $\theta = 0$ or $\cos\theta = 1$. The empirically derived formula for energy and how it is related to the plasma line power structures is similar to what was used here, except from the exponent on the cosine. In the numerical analyses carried out here, the best fit was achieved for exponents of 2 and 1 in eqs. (5.4) and (5.5), respectively.

6 Conclusion

In this thesis derivations of dielectric functions have been carried out which are a fundamental part of the derivation of the incoherent scatter spectrum. This was done for a Maxwellian distribution, a kappa distribution and arbitrary isotropic distributions, and subsequently implemented in computer code. The program that was developed includes an ambient geomagnetic field and as such accepts a radar beam pointing at oblique angles to the magnetic field. The derivations were based on the work by Hagfors (1961) and Mace (2003).

A method for calculating the IS spectra for isotropic distributions was presented in chapter 4. The Simpson's algorithm was used to solve the integrals, and a chirp-sampling was used in addition to a finite upper boundary to calculate the integrals more efficiently. To validate the accuracy of the extension to a general method from the semi-analytic implementation, the Maxwellian distribution and kappa distribution was included in both methods for comparison. This analysis showed the limitations of using distributions with vanishing magnitude in the high-energy tail caused by the decimal precision. The importance of sampling with high enough density in the Gordeyev integral was also evident, where a sample size of $N_y = 8 \times 10^4$ was found to be sufficient up to about 9.5 MHz in the IS spectrum. A sample size in the Gordeyev integral of $N_y = 8 \times 10^5$ was also used, which provided good results up to 12 MHz. At such high sampling points, however, the Simpson's algorithm quickly become very slow and another algorithm, the chirp z-transform, was suggested. The chirp z-transform algorithm did not yield consistent results, but the calculated peak

frequencies seemed to converge as the sampling was increased. Since the chirp z-transform is many times more efficient than the Simpson's algorithm, a working implementation of the chirp z-transform or similar algorithms should be sought if higher sampling is needed.

The primary result in this thesis is the derivation of the IS spectra to include arbitrary isotropic distributions. This was done to be able to consider suprathermal electrons which substantially change the velocity distribution of electrons away from a Maxwellian. The derivation of the dielectric function also account for radar pointing direction at oblique angles to the magnetic field. This was important to enable analysis of observations and measurements made by radars located at low latitude, since these radars cut through the magnetic field at an angle when probing the ionosphere.

This includes the radar at the Arecibo Observatory, and recent measurements made by the Arecibo radar was analysed using the program developed here. Specifically, simulations of structures in the ionosphere in presence of a multi-peaked suprathermal electron distribution was carried out to reproduce the measurements by varying similar plasma parameters. It was shown that the peaks/enhancements in the velocity distribution function for the suprathermal electrons map to structures seen in the plasma line power as a function of aspect angle and electron number density. This mapping was done according to a formula relating the plasma wave phase velocity along the radar pointing direction, scaled by the cosine of the angle to the magnetic field, to the energy of enhancements in the suprathermal velocity distribution function.

Further, it was shown that the program was able to reproduce known results for an electron distribution with a high-energy tail, thus showing the consistency between previous results and the program. This includes the increased Landau-damping of both the ion lines and the plasma lines in response to electron distributions with high-energy tails and a downshift of the resonance frequency of both ion lines and plasma lines.

6.1 Future work

The dielectric function that was derived here for the calculation of the IS spectrum was restricted to isotropic distributions, both in regard to the kappa distribution and the arbitrary distribution. A natural next step is to extend this to include anisotropic distributions. One such anisotropic distribution is the kappa distribution in eq. (3.32) which has been studied by Gaelzer et al. (2016), but that was found by Gaelzer et al. (2016) to have no known implementation in computer code.

The implementations of the algorithms used here was found to have some limitations, and improving the numerical precision is another suggested future work. One Python library that implement high numerical precision is the `mpmath` library, but without an implementation of the Simpson's algorithm, different algorithms for calculating the integrals would have to be studied. Increasing the number of samples used in the integrals would also provide better numerical precision, and taking advantage of the FFT through for example the chirp z-transform is a possible approach to achieve higher sampling.

A

Source code

The computer code used in this thesis is listed in the sections below. The code was written in Python 3.8.2 64-bit using the Visual Studio Code Insiders editor, and the environment was macOS Catalina version 10.15.5. A GitHub Pages site for the repository can be found at `https://engeir.github.io/isr_spectrum/`. Alternatively a release (`v1.0`) can be downloaded as a `.zip` file of the complete repository with the correct file structure as it was at the time the thesis was finalized: `https://github.com/engeir/isr_spectrum/archive/v1.0.zip`.

The electron power density spectrum referred to as the IS spectrum, derived in eq. (2.55a), can be found in appendix A.5, line 81. Appendix A.6 contain all the integrands of the Gordeyev integrals (eqs. (2.56), (3.6) and (3.24)) as classes, while the scaling of the integrals is done in appendix A.10, line 42. The velocity integral (eq. (4.4)) was solved in appendix A.11, line 43, while the distribution functions are given in the different classes in appendix A.7.

To generate the data needed for both plots of plasma line power, run the `run()` method (uncomment line 170 in appendix A.1) of the `HelloKitty` class found in appendix A.4. Plots of the data are generated by the `PlotHK` class in appendix A.3, line 607. (The measurements in fig. 5.6 resemble a kittens eyes, hence the nickname "HelloKitty".) Plots of the IS spectra used in this thesis are generated and plotted from the remaining classes in appendix A.3 and ran from the `main.py` file, appendix A.1 line 44.

A.1 main.py

```
"""Main script for controlling the calculation of the IS spectrum.

Calculate spectra from specified parameters as shown in the
examples given in the class methods, create a new set-up with
the `Reproduce` abstract base class in `reproduce.py` or use
one of the pre-defined classes from `reproduce.py`.
"""

# The start method of the multiprocessing module was changed from python3.7
# to python3.8. Instead of using 'fork', 'spawn' is the new default.
# To be able to use global variables across all parallel processes,
# the start method must be reset to 'fork'. See
# https://tinyurl.com/yyxxfxst for more info.
import multiprocessing as mp
mp.set_start_method('fork')

import matplotlib  # pylint: disable=C0413
import matplotlib.pyplot as plt  # pylint: disable=C0413
import numpy as np  # pylint: disable=C0413

from plotting import hello_kitty as hk  # pylint: disable=C0413
from plotting import reproduce  # pylint: disable=C0413
from plotting.plot_class import PlotClass  # pylint: disable=C0413


# Customize matplotlib
matplotlib.rcParams.update({
    'text.usetex': True,
    'font.family': 'DejaVu Sans',
    'axes.unicode_minus': False,
    'pgf.texsystem': 'pdflatex'
})


class Simulation:
    def __init__(self):
        self.from_file = False
        self.f = np.ndarray([])
        self.data = []
        self.meta_data = []
        self.legend_txt = []
        self.ridge_txt = []
        self.plot = PlotClass()
        # self.r = reproduce.PlotNumerical(self.plot)
        # self.r = reproduce.PlotTestDebye(self.plot)
        # self.r = reproduce.PlotSpectra(self.plot)
        # self.r = reproduce.PlotIonLine(self.plot)
        # self.r = reproduce.PlotPlasmaLine(self.plot)
        self.r = reproduce.PlotTemperature(self.plot)
        # self.r = reproduce.PlotHKExtremes(self.plot)

```

```
52      def create_data(self):
53          """Create IS spectra.
54
55          The spectra should be appended to the `self.data` list, giving a
56          list of spectra that are themselves `np.ndarrays`, or into a list
57          of such lists as the aforementioned.
58
59          A list of spectra can be plotted in `plot_normal`, while a list of
60          lists can be plotted by `plot_ridge`. When using `plot_ridge`, it is
61          assumed that all the lists in the outer list is of equal length.
62
63          The list `self.ridge_txt` should be the same length as the length
64          of the outer list when plotting with `plot_ridge`, since this text
65          will go on the left of every ridge. The list `self.legend_txt` should
66          be the same length as the length of the inner lists, and will give
67          the legend for the spectra given in the inner lists.
68
69          Notes:
70          ::
71          Possible items in the sys_set dictionary include:
72              K_RADAR -- Radar wavenumber
73                         (= -4pi(radar frequency)/(speed of light)) [m^(-1)]
74              B -- Magnetic field strength [T]
75              MI -- Ion mass in atomic mass units [u]
76              NE -- Electron number density [m^(-3)]
77              NU_E -- Electron collision frequency [Hz]
78              NU_I -- Ion collision frequency [Hz]
79              T_E -- Electron temperature [K]
80              T_I -- Ion temperature [K]
81              T_ES -- Temperature of suprathermal electrons in the
82                      gauss_shell VDF [K]
83              THETA -- Aspect angle [1]
84              Z -- Height of real data [100, 599] [km]
85              mat_file -- Important when using real data and decides
86                          the time of day
87              pitch_angle -- list of integers that determine which slices
88                             of the pitch angles are used. 'all' uses all
89
90          Examples:
91          ::
92          ```
93              TEMPS = [2000, 5000]
94              methods = ['maxwell', 'kappa']
95              sys_set = {'B': 5e-4, 'MI': 16, 'NE': 2e11, 'NU_E': 0, 'NU_I': 0,
96                         'T_E': 5000, 'T_I': 2000, 'T_ES': 90000,
97                         'THETA': 40 * np.pi / 180, 'Z': 599,
98                          'mat_file': 'fe_zmuE-01.mat'}
99              params = {'kappa': 3, 'vdf': 'kappa', 'area': False}
100             for T in TEMPS:
101                 ridge = []
102                 sys_set['T_E'] = T
103                 self.ridge_txt.append(f'$T_e = {T}$ K')
104                 for m in methods:
105                     self.f, s, meta_data = isr.isr_spectrum(m, sys_set, **params)
```

```
106                     self.meta_data.append(meta_data)
107                     ridge.append(s)
108                 self.data.append(ridge)
109 
110             # For a nicer legend, this is added manually
111             self.legend_txt.append('Maxwellian')
112             self.legend_txt.append('Kappa')
113         ```
114         """
115         # self.from_file = True
116         self.r.create_it('../figures/temp_ridge.npz',
            ↪  from_file=self.from_file)
117         self.f = self.r.f
118         self.data = self.r.data
119         self.legend_txt = self.r.legend_txt
120         self.ridge_txt = self.r.ridge_txt
121         self.meta_data = self.r.meta_data
122 
123     def plot_data(self):
124         """Plot the created data from `self.data`.
125 
126         If you want to only plot the plasma line, set
127         ```
128             self.plot.plasma = True
129         ```
130 
131         `self.plot.plot_normal()` accepts a list of `np.ndarray`s and
132         `self.plot.plot_ridge()` accepts a list of lists of `np.ndarray`s,
133         i.e. a list of the type you send to `self.plot.plot_normal()`.
134 
135         Examples:
136         ::
137         ```
138             # Given the example in self.create_data()
139             # self.plot.plasma = True
140             self.plot.plot_normal(self.f, self.data[0], 'plot',
141                                   self.legend_txt)
142             self.plot.plot_normal(self.f, self.data[0], 'semilogy',
143                                   self.legend_txt)
144             self.plot.plot_ridge(self.f, self.data, 'plot', self.legend_txt,
145                                  self.ridge_txt)
146             self.plot.plot_ridge(self.f, self.data, 'semilogy',
147                                  self.legend_txt, self.ridge_txt)
148         ```
149         """
150         self.r.plot_it()
151 
152     def save_handle(self, mode):
153         if mode == 'setUp':
154             if self.plot.save in ['y', 'yes']:
155                 self.plot.save_it(self.f, self.data, self.legend_txt,
                    ↪  self.ridge_txt, self.meta_data)
156         elif mode == 'tearDown':
157             if self.plot.save in ['y', 'yes']:
```

```
158                 self.plot.pdffig.close()
159             plt.show()
160
161     def run(self):
162         self.create_data()
163         self.save_handle('setUp')
164         self.plot_data()
165         self.save_handle('tearDown')
166
167
168 if __name__ == '__main__':
169     Simulation().run()
170     # hk.HelloKitty(1).run()
```

A.2 config.py

```
1  """Constants used system wide.
2  """
3
4  import os
5  import sys
6
7  import numpy as np
8
9
10 # Check if a test is running. Potential paths are
11 # ['pytest.py', 'pytest', 'test_ISR.py', '__main__.py', 'python3.7 -m
   ↪ unittest']
12 # or check if 'main.py' was used.
13 if os.path.basename(os.path.realpath(sys.argv[0])) != 'main.py':
14     # DO NOT EDIT
15     F_N_POINTS = 1e1
16     Y_N_POINTS = 1e1
17     V_N_POINTS = 1e1
18 else:
19     F_N_POINTS = 1e4  # Number of sample points in frequency, N_f
20     Y_N_POINTS = 8e4  # Number of sample points in integral variable, N_y
21     V_N_POINTS = 4e4  # Number of sample points in velocity integral
       ↪ variable, N_v
22 # Adds one sample to get an even number of bins, which in
23 # turn give better precision in the Simpson integration.
24 Y_N_POINTS += 1
25 V_N_POINTS += 1
26 Y_MAX_e = 1.5e-4  # Upper limit of integration (= infinity)
27 Y_MAX_i = 1.5e-2
28 # Based on E = 110 eV -> 6.22e6 m/s
29 V_MAX = 6e6
30 ORDER = 3
31
32 I_P = {'F_MIN': 2.5e6, 'F_MAX': 9.5e6}
```

```
f = np.linspace(I_P['F_MIN'], I_P['F_MAX'], int(F_N_POINTS))
f = (f / I_P['F_MAX'])**1 * I_P['F_MAX']
w = 2 * np.pi * f  # Angular frequency
```

A.3 reproduce.py

```
"""Reproduce the plots used in the thesis, and/or create new
"experiments" based on the abstract base class `Reproduce`.

Run from `main.py`.
"""

import sys
from abc import ABC, abstractmethod

import matplotlib
import matplotlib.pyplot as plt
from matplotlib import gridspec
import matplotlib.patheffects as PathEffects
import numpy as np
import scipy.constants as const

# from inputs import config as cf

# Customize matplotlib
matplotlib.rcParams.update({
    'text.usetex': True,
    'font.family': 'DejaVu Sans',
    'axes.unicode_minus': False,
    'pgf.texsystem': 'pdflatex'
})

if __name__ != '__main__':
    from utils import spectrum_calculation as isr


class Reproduce(ABC):
    """Abstract base class to reproduce figures.

    Arguments:
        ABC {class} -- abstract base class
    """

    def __init__(self, p):
        self.f = np.ndarray([])
        self.data = []
        self.meta_data = []
        self.legend_txt = []
        self.ridge_txt = []
        self.p = p
```

```
45
46      def create_it(self, *args, from_file=False):
47          if not from_file:
48              self.create_from_code()
49          else:
50              self.create_from_file(*args)
51
52      @abstractmethod
53      def create_from_code(self):
54          """Method that create needed data.
55          """
56
57      def create_from_file(self, *args):
58          """Accepts zero, one or two arguments.
59
60          If zero arguments are given, a default path is used to look for files.
61          ::
62          If one argument is given, it should include
63          the full path (with or without file ending).
64          ::
65          If two arguments are given, the first should be the path to
66          the directory where the file is located, and the second
67          argument must be the name of the file.
68          """
69          if len(args) != 0:
70              if len(args) == 1:
71                  args = args[0]
72                  parts = args.split('/')
73                  path = '/'.join(parts[:-1]) + '/'
74                  name = parts[-1]
75              elif len(args) == 2:
76                  path = args[0]
77                  name = args[1]
78          else:
79              path = '../../figures/'
80              name = 'hello_kitty_2020_6_9_2--28--4.npz'
81          name = name.split('.')[0]
82          try:
83              f = np.load(path + name + '.npz', allow_pickle=True)
84          except Exception:
85              sys.exit(print(f'Could not open file {path + name}.npz'))
86          sorted(f)
87          self.f, self.data, self.meta_data = f['frequency'],
            ↪  list(f['spectra']), list(f['meta'])
88          self.legend_txt, self.ridge_txt = list(f['legend_txt']),
            ↪  list(f['ridge_txt'])
89
90          if self.p.save in ['y', 'yes']:
91              self.p.save_path = name
92
93      @abstractmethod
94      def plot_it(self):
95          """Method that plot relevant plots.
96          """
```

```
97
98
99  class PlotNumerical(Reproduce):
100     """Reproduce figure with a comparison between the semi-analytic
101     and numerical implementation.
102
103     In config, set
104     ```
105         'F_MIN': - 2e6, 'F_MAX': 9e6
106     ```
107     Also, using
108     ```
109         F_N_POINTS = 1e3
110     ```
111     is sufficient.
112     """
113     def create_from_code(self):
114         F0 = 430e6
115         K_RADAR = - 2 * F0 * 2 * np.pi / const.c  # Radar wavenumber
116         sys_set = {'K_RADAR': K_RADAR, 'B': 35000e-9, 'MI': 16,
117                    'NE': 1e12, 'NU_E': 100, 'NU_I': 100, 'T_E': 2000,
118                    'T_I': 1500, 'T_ES': 90000,
119                    'THETA': 30 * np.pi / 180, 'Z': 300,
120                    'mat_file': 'fe_zmuE-07.mat',
121                    'pitch_angle': 'all'}
122         params = {'kappa': 3, 'vdf': 'maxwell', 'area': False}
123
124         ridge = []
125         self.f, s1, meta_data = isr.isr_spectrum('maxwell', sys_set,
            ↪ **params)
126         ridge.append(s1)
127         self.meta_data.append(meta_data)
128         _, s2, _ = isr.isr_spectrum('a_vdf', sys_set, **params)
129         ridge.append(s2)
130         self.data.append(ridge)
131
132         ridge = []
133         params['vdf'] = 'kappa'
134         self.f, s1, meta_data = isr.isr_spectrum('kappa', sys_set, **params)
135         ridge.append(s1)
136         self.meta_data.append(meta_data)
137         _, s2, _ = isr.isr_spectrum('a_vdf', sys_set, **params)
138         ridge.append(s2)
139         self.data.append(ridge)
140
141     def plot_it(self):
142         for maxwell, data in enumerate(self.data):
143             self.plotter(maxwell, data)
144
145     def plotter(self, maxwell, data):
146         s1 = data[0]
147         s2 = data[1]
148         plot = plt.semilogy
149         xlim = [1e3, self.f[-1]]
```

```
150         d = s1 - s2
151         rd = d / s1
152         plt.figure(figsize=(8, 5))
153         plt.subplot(3, 1, 1)
154         if maxwell == 0:
155             plt.title('Maxwell')
156         else:
157             plt.title('Kappa')
158         plot(self.f, s1, 'k', label='Semi-analytic (SA)')
159         plot(self.f, s2, 'r--', label='Numerical (N)')
160         plt.legend()
161         # plt.xlim(xlim)
162         plt.minorticks_on()
163         plt.grid(True, which="both", ls="-", alpha=0.4)
164         plt.subplot(3, 1, 2)
165         plt.title('Difference (SA - N)')
166         plot(self.f, d, 'k', label='Positive')
167         plot(self.f, - d, 'r', label='Negative')
168         plt.legend()
169         # plt.xlim(xlim)
170         plt.minorticks_on()
171         plt.grid(True, which="both", ls="-", alpha=0.4)
172         plt.subplot(3, 1, 3)
173         plt.title('Difference relative to semi-analytic [(SA - N) / SA]')
174         plot(self.f, rd, 'k', label='Positive')
175         plot(self.f, - rd, 'r', label='Negative')
176         plt.legend()
177         # plt.xlim(xlim)
178         plt.minorticks_on()
179         plt.grid(True, which="both", ls="-", alpha=0.4)
180         plt.yticks([1e-9, 1e-6, 1e-3, 1e0])
181
182         plt.tight_layout()
183
184         if self.p.save in ['y', 'yes']:
185             self.p.pdffig.attach_note('numerical precision test')
186             plt.savefig(self.p.pdffig, bbox_inches='tight', format='pdf',
                ↪  dpi=600)
187             plt.savefig(str(self.p.save_path) + f'_page_{self.p.page}.pgf',
                ↪  bbox_inches='tight')
188             self.p.page += 1
189
190
191 class PlotTestDebye(Reproduce):
192     """Reproduce figure of IS spectra using two kappa
193     dist with and without Debye length correction.
194
195     In config, set
196     ```
197         'F_MIN': - 2e6, 'F_MAX': 2e6
198     ```
199     Also, using
200     ```
201         F_N_POINTS = 5e5
```

```
    ```
    is sufficient.
    """

    def create_from_code(self):
        F0 = 430e6
        K_RADAR = - 2 * F0 * 2 * np.pi / const.c # Radar wavenumber
        self.legend_txt = [r'$\lambda_{\mathrm{D}} = \lambda_{\mathrm{D},\kappa}$', r'$\lambda_{\mathrm{D}} = \lambda_{\mathrm{D,M}}$']
        sys_set = {'K_RADAR': K_RADAR, 'B': 35000e-9, 'MI': 29, 'NE': 2e10, 'NU_E': 0, 'NU_I': 0, 'T_E': 200, 'T_I': 200, 'T_ES': 90000,
                   'THETA': 45 * np.pi / 180, 'Z': 599, 'mat_file': 'fe_zmuE-07.mat'}
        params = {'kappa': 3, 'vdf': 'real_data', 'area': False}
        self.f, s, meta_data = isr.isr_spectrum('kappa', sys_set, **params)
        self.data.append(s)
        self.meta_data.append(meta_data)
        params['debye'] = 'maxwell'
        self.f, s, meta_data = isr.isr_spectrum('kappa', sys_set, **params)
        self.data.append(s)
        self.meta_data.append(meta_data)

    def plot_it(self):
        self.p.plot_normal(self.f, self.data, 'semilogy', self.legend_txt)

class PlotSpectra(Reproduce):
    """Reproduce figure with ridge plot over different temperatures.

    In config, set
    ```
        'F_MIN': - 2e6, 'F_MAX': 2e6
    ```
    Also, using
    ```
        F_N_POINTS = 1e5
    ```
    is sufficient.
    """
    def create_from_code(self):
        F0 = 430e6
        K_RADAR = - 2 * F0 * 2 * np.pi / const.c # Radar wavenumber
        self.legend_txt = ['Maxwellian', r'$\kappa = 20$', r'$\kappa = 8$', r'$\kappa = 3$']
        kappa = [20, 8, 3]
        sys_set = {'K_RADAR': K_RADAR, 'B': 35000e-9, 'MI': 29, 'NE': 2e10, 'NU_E': 0, 'NU_I': 0, 'T_E': 200, 'T_I': 200, 'T_ES': 90000,
                   'THETA': 45 * np.pi / 180, 'Z': 599, 'mat_file': 'fe_zmuE-07.mat'}
        params = {'kappa': 20, 'vdf': 'real_data', 'area': False}
        self.f, s, meta_data = isr.isr_spectrum('maxwell', sys_set, **params)
        self.data.append(s)
```
```

```
248         for k in kappa:
249             params['kappa'] = k
250             self.f, s, meta_data = isr.isr_spectrum('kappa', sys_set,
                ↪  **params)
251             self.data.append(s)
252         meta_data['version'] = 'both'
253         self.meta_data.append(meta_data)
254 
255     def plot_it(self):
256         self.p.plot_normal(self.f, self.data, 'semilogy', self.legend_txt)
257 
258 
259 class PlotIonLine(Reproduce):
260     """Reproduce figure with ridge plot over different temperatures.
261 
262     In config, set
263     ```
264         'F_MIN': - 3e3, 'F_MAX': 3e3
265     ```
266     Also, using
267     ```
268         F_N_POINTS = 1e3
269     ```
270     is sufficient.
271     """
272     def create_from_code(self):
273         F0 = 430e6
274         K_RADAR = - 2 * F0 * 2 * np.pi / const.c
275         self.legend_txt = ['Maxwellian', r'$\kappa = 20$', r'$\kappa = 8$',
            ↪  r'$\kappa = 3$']
276         kappa = [20, 8, 3]
277         sys_set = {'K_RADAR': K_RADAR, 'B': 35000e-9, 'MI': 29, 'NE': 2e10,
            ↪  'NU_E': 0, 'NU_I': 0, 'T_E': 200, 'T_I': 200, 'T_ES': 90000,
278                    'THETA': 45 * np.pi / 180, 'Z': 599, 'mat_file':
                       ↪  'fe_zmuE-07.mat'}
279         params = {'kappa': 20, 'vdf': 'real_data', 'area': False}
280         self.f, s, meta_data = isr.isr_spectrum('maxwell', sys_set,
            ↪  **params)
281         self.data.append(s)
282         for k in kappa:
283             params['kappa'] = k
284             self.f, s, meta_data = isr.isr_spectrum('kappa', sys_set,
                ↪  **params)
285             self.data.append(s)
286         meta_data['version'] = 'both'
287         self.meta_data.append(meta_data)
288 
289     def plot_it(self):
290         self.p.plot_normal(self.f, self.data, 'plot', self.legend_txt)
291 
292 
293 class PlotPlasmaLine(Reproduce):
294     """Reproduce figure with ridge plot over different temperatures.
295 
```

```
296     In config, set
297     ```
298         'F_MIN': 3.5e6, 'F_MAX': 7e6
299     ```
300     Also, using
301     ```
302         F_N_POINTS = 1e3
303     ```
304     is sufficient.
305     """
306     def create_from_code(self):
307         F0 = 933e6
308         K_RADAR = - 2 * F0 * 2 * np.pi / const.c
309         self.legend_txt = ['Maxwellian', r'$\kappa = 20$', r'$\kappa = 8$',
            ↪  r'$\kappa = 3$']
310         kappa = [20, 8, 3]
311         sys_set = {'K_RADAR': K_RADAR, 'B': 50000e-9, 'MI': 16, 'NE': 2e11,
            ↪  'NU_E': 0, 'NU_I': 0, 'T_E': 5000, 'T_I': 2000, 'T_ES': 90000,
312                    'THETA': 0 * np.pi / 180, 'Z': 599, 'mat_file':
                       ↪  'fe_zmuE-07.mat'}
313         params = {'kappa': 20, 'vdf': 'real_data', 'area': False}
314         self.f, s, meta_data = isr.isr_spectrum('maxwell', sys_set,
            ↪  **params)
315         self.data.append(s)
316         for k in kappa:
317             params['kappa'] = k
318             self.f, s, meta_data = isr.isr_spectrum('kappa', sys_set,
                ↪  **params)
319             self.data.append(s)
320         meta_data['version'] = 'both'
321         self.meta_data.append(meta_data)
322
323     def plot_it(self):
324         self.p.plot_normal(self.f, self.data, 'plot', self.legend_txt)
325
326
327 class PlotTemperature(Reproduce):
328     """Reproduce figure with ridge plot over different temperatures.
329
330     In config, set
331     ```
332         'F_MIN': 3.5e6, 'F_MAX': 7.5e6
333     ```
334     Also, using
335     ```
336         F_N_POINTS = 5e3
337     ```
338     is sufficient.
339     """
340     def __init__(self, p):
341         super(PlotTemperature, self).__init__(p)
342         self.f_list = [[], [], []]
343
344     def create_from_file(self, *args):
```

```
345         """Accepts zero, one or two arguments.
346 
347         If zero arguments are given,
348         a default path is used to look for files.
349         ::
350         If one argument is given, it should include
351         the full path (with or without file ending).
352         ::
353         If two arguments are given, the first should be the path to
354         the directory where the file is located, and the second
355         argument must be the name of the file.
356         """
357         if len(args) != 0:
358             if len(args) == 1:
359                 args = args[0]
360                 parts = args.split('/')
361                 path = '/'.join(parts[:-1]) + '/'
362                 name = parts[-1]
363             elif len(args) == 2:
364                 path = args[0]
365                 name = args[1]
366         else:
367             path = '../../figures/'
368             name = 'hello_kitty_2020_6_9_2--28--4.npz'
369         name = name.split('.')[0]
370         try:
371             f = np.load(path + name + '.npz', allow_pickle=True)
372         except Exception:
373             sys.exit(print(f'Could not open file {path + name}.npz'))
374         sorted(f)
375         self.f, self.data, self.meta_data = f['frequency'],
            ↪  list(f['spectra']), list(f['meta'])
376         self.legend_txt, self.ridge_txt = list(f['legend_txt']),
            ↪  list(f['ridge_txt'])
377 
378         for r in self.data:
379             peak = int(np.argwhere(r[0] == np.max(r[0])))
380             self.f_list[0].append(self.f[peak])
381             peak = int(np.argwhere(r[1] == np.max(r[1])))
382             self.f_list[1].append(self.f[peak])
383             peak = int(np.argwhere(r[2] == np.max(r[2])))
384             self.f_list[2].append(self.f[peak])
385 
386         if self.p.save in ['y', 'yes']:
387             self.p.save_path = name
388 
389     def create_from_code(self):
390         F0 = 933e6
391         K_RADAR = - 2 * F0 * 2 * np.pi / const.c
392         T = [2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000, 10000]
393         self.ridge_txt = [r'$T_{\mathrm{e}} = %d \mathrm{K}$' % j for j in
            ↪  T]
394         self.legend_txt = ['Maxwellian', r'$\kappa = 20$', r'$\kappa = 3$']
```

```
395         sys_set = {'K_RADAR': K_RADAR, 'B': 50000e-9, 'MI': 16, 'NE': 2e11,
            ↪  'NU_E': 0, 'NU_I': 0, 'T_E': 2000, 'T_I': 2000, 'T_ES': 90000,
396                    'THETA': 0 * np.pi / 180, 'Z': 599, 'mat_file':
                       ↪  'fe_zmuE-07.mat'}
397         params = {'kappa': 8, 'vdf': 'real_data', 'area': False}
398         kappa = [20, 3]
399         for t in T:
400             ridge = []
401             sys_set['T_E'] = t
402             self.f, s, meta_data = isr.isr_spectrum('maxwell', sys_set,
                ↪  **params)
403             ridge.append(s)
404             for k in kappa:
405                 params['kappa'] = k
406                 self.f, s, meta_data = isr.isr_spectrum('kappa', sys_set,
                    ↪  **params)
407                 ridge.append(s)
408             self.data.append(ridge)
409         self.meta_data.append(meta_data)
410
411         for r in self.data:
412             peak = int(np.argwhere(r[0] == np.max(r[0])))
413             self.f_list[0].append(self.f[peak])
414             peak = int(np.argwhere(r[1] == np.max(r[1])))
415             self.f_list[1].append(self.f[peak])
416             peak = int(np.argwhere(r[2] == np.max(r[2])))
417             self.f_list[2].append(self.f[peak])
418
419     def plot_it(self):
420         self.p.plot_ridge(self.f, self.data, 'plot', self.legend_txt,
            ↪  self.ridge_txt)
421
422         T = [2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000, 10000]
423         plt.figure(figsize=(6, 3))
424         plt.plot(T, self.f_list[0], 'k', label='Maxwellian')
425         plt.plot(T, self.f_list[1], 'k--', label=r'$\kappa = 20$')
426         plt.plot(T, self.f_list[2], 'k:', label=r'$\kappa = 3$')
427         plt.legend()
428
429         if self.p.save in ['y', 'yes']:
430             self.p.pdffig.attach_note('freq change')
431             plt.savefig(self.p.pdffig, bbox_inches='tight', format='pdf',
                ↪  dpi=600)
432             plt.savefig(str(self.p.save_path) + f'_page_{self.p.page}.pgf',
                ↪  bbox_inches='tight')
433             self.p.page += 1
434
435
436 class PlotHKExtremes(Reproduce):
437     """Reproduce figure with ridge plot over the extremes from
438     the Hello Kitty plot.
439
440     In config, set
441     ```
```

```
        'F_MIN': 2.5e6, 'F_MAX': 9.5e6
    ```
    Also, using
    ```
        F_N_POINTS = 1e4
    ```
    is sufficient.
    """
    def create_from_code(self):
        F0 = 430e6
        K_RADAR = - 2 * F0 * 2 * np.pi / const.c # Radar wavenumber
        sys_set = {'K_RADAR': K_RADAR, 'B': 35000e-9, 'MI': 16, 'NE': 1e11,
                   'NU_E': 100, 'NU_I': 100, 'T_E': 2000, 'T_I': 1500,
                   ↪ 'T_ES': 90000,
                   'THETA': 30 * np.pi / 180, 'Z': 599, 'mat_file':
                   ↪ 'fe_zmuE-07.mat',
                   'pitch_angle': list(range(10))}
        params = {'kappa': 8, 'vdf': 'real_data', 'area': False}
        # Ridge 1
        ridge = []
        # Line 1
        self.f, s, meta_data = isr.isr_spectrum('a_vdf', sys_set, **params)
        ridge.append(s)
        self.meta_data.append(meta_data)
        # Line 2
        sys_set['NE'] = 1e12
        self.f, s, meta_data = isr.isr_spectrum('a_vdf', sys_set, **params)
        ridge.append(s)
        self.data.append(ridge)
        self.meta_data.append(meta_data)

        # Ridge 2
        ridge = []
        # Line 1
        sys_set['THETA'] = 60 * np.pi / 180
        sys_set['NE'] = 1e11
        self.f, s, meta_data = isr.isr_spectrum('a_vdf', sys_set, **params)
        ridge.append(s)
        self.meta_data.append(meta_data)
        # Line 2
        sys_set['NE'] = 1e12
        self.f, s, meta_data = isr.isr_spectrum('a_vdf', sys_set, **params)
        ridge.append(s)
        self.data.append(ridge)
        self.meta_data.append(meta_data)

        self.legend_txt = ['1e11', '1e12']
        self.ridge_txt = ['30', '60']

    def plot_it(self):
        self.p.plot_ridge(self.f, self.data, 'semilogy', self.legend_txt,
        ↪ self.ridge_txt)
```
```

```
493 class PlotHK:
494     """Reproduce the Hello Kitty figures from saved data."""
495     def __init__(self, *args):
496         """Accepts zero, one or two arguments.
497 
498         If zero arguments are given, a default path is used to look for files.
499         ::
500         If one argument is given, it should include
501         the full path (with or without file ending).
502         ::
503         If two arguments are given, the first should be the path to
504         the directory where the file is located, and the second
505         argument must be the name of the file.
506         """
507         if len(args) != 0:
508             if len(args) == 1:
509                 args = args[0]
510                 parts = args.split('/')
511                 path = '/'.join(parts[:-1]) + '/'
512                 self.name = parts[-1]
513             elif len(args) == 2:
514                 path = args[0]
515                 self.name = args[1]
516         else:
517             path = '../../figures/'
518             # Old
519             # self.name = 'hello_kitty_2020_6_9_2--28--4.npz'
520             self.name = 'hello_kitty_2020_6_8_22--1--51.npz'
521             # New
522             # self.name = 'hello_kitty_2020_6_15_22--27--16.npz'
523             # self.name = 'hello_kitty_2020_6_15_15--50--18.npz'
524         self.name = self.name.split('.')[0]
525         try:
526             self.file = np.load(path + self.name + '.npz')
527         except Exception:
528             sys.exit(print(f'Could not open file {path + self.name}'))
529         self.g = self.file['power']
530 
531     def shade(self):
532         dots_x = []
533         dots_y = []
534         for i, d in enumerate(self.file['dots'][1]):
535             arg = np.argwhere(self.file['angle'] ==
                ↪  self.file['angle'][int(d)])
536             dots_x = np.r_[dots_x, arg[:1, 0]]
537             dots_y = np.r_[dots_y, np.ones(len(arg[:1, 0])) *
                ↪  self.file['dots'][2][i]]
538 
539         s = set(self.file['dots'][0])
540         for i in s:
541             mask = np.argwhere(self.file['dots'][0]==i)
542             xs = []
543             y_min = []
544             y_max = []
```

```
            for x in range(30):
                arg = np.argwhere(dots_x[mask].flatten() == x)
                if bool(arg.any()):
                    xs.append(x)
                    y_min.append(np.min(dots_y[mask][arg]))
                    y_max.append(np.max(dots_y[mask][arg]))
            plt.fill_between(xs, y_min, y_max, color='g', alpha=.8)
            x, y = xs[-1], (y_max[-1] + y_min[-1]) / 2
            txt = plt.text(x, y, r'$\mathrm{}$'.format(int(i)), color='k',
            ↪  va='center', ha='right', fontsize=15)
            txt.set_path_effects([PathEffects.withStroke(linewidth=1,
            ↪  foreground='w')])

    def shade2p0(self, *args):
        """Mark points on the plasma line power plot
        that map to any number of energy (eV) intervals.

        *args can be any number of lists
        or tuples of length 2 (E_min, E_max)
        """
        l = const.c / 430e6
        deg = self.file['angle'][:self.file['fr'].shape[1]]
        E_plasma = .5 * const.m_e * (self.file['fr'] * l / (2 * np.cos(deg *
        ↪  np.pi / 180)**(1)))**2 / const.eV
        for a in args:
            try:
                if len(a) == 2:
                    m = (a[0] < E_plasma) & (E_plasma < a[1])
                    self.g[:, :30][m] = np.nan
            except Exception:
                pass

    def plot_it(self):
        # self.shade2p0([15.88, 18.72], [22.47, 23.75], [60, 64])
        # self.shade2p0([20.29, 21.99], [22.45, 23.82], (25.38, 27.03),
        ↪  [32.82, 34.33], [46, 47], [61.55, 65])
        f = plt.figure(figsize=(8, 5))
        gs = gridspec.GridSpec(2, 1, height_ratios=[4, 1])
        ax0 = plt.subplot(gs[0])
        im = ax0.imshow(self.g,
                        extent=[0, len(self.file['angle']) - 1,
                        ↪  np.min(self.file['density']),
                        ↪  np.max(self.file['density'])],
                        origin='lower', aspect='auto', cmap='gist_heat')
        current_cmap = im.get_cmap()
        current_cmap.set_bad(color='green', alpha=.6)
        self.shade()
        plt.ylabel(r'Electron number density, $n_{\mathrm{e}}$')
        plt.tick_params(axis='x', which='both', bottom=False,
                        top=False, labelbottom=False)
        ax1 = plt.subplot(gs[1])
        ax1.plot(180 - self.file['angle'], 'k')
        plt.xlim([0, len(self.file['angle']) - 1])
        plt.yticks([150, 135, 120])
```

```
593         plt.ylabel('Aspect angle')
594         axs = []
595         axs += [ax0]
596         axs += [ax1]
597         gs.update(hspace=0.05)
598         f.colorbar(im, ax=axs).ax.set_ylabel('Echo power')
599         plt.tick_params(axis='x', which='both', bottom=False,
600                         top=False, labelbottom=False)
601         plt.savefig(f'{self.name}.pgf', bbox_inches='tight',
            ↪  transparent=True)
602
603         plt.show()
604
605
606 if __name__ == '__main__':
607     PlotHK().plot_it() #
```

A.4 hello_kitty.py

```
1  """Script for calculating the peak power of the plasma line
2  at different aspect angles, height and time of day.
3
4  Already implemented are two versions, vol.1 and vol.2.
5  Run from `main.py`.
6  """
7
8  import os
9  import sys
10 import time
11 import datetime
12
13 import numpy as np
14 import matplotlib
15 import matplotlib.pyplot as plt
16 from matplotlib.backends.backend_pdf import PdfPages
17 from matplotlib import gridspec
18 import scipy.integrate as si
19 import scipy.constants as const
20 from lmfit.models import LorentzianModel
21 from tqdm import tqdm
22
23 from utils import spectrum_calculation as isr
24 from inputs import config as cf
25
26 # Customize matplotlib
27 matplotlib.rcParams.update({
28     'text.usetex': True,
29     'font.family': 'DejaVu Sans',
30     'axes.unicode_minus': False,
31     'pgf.texsystem': 'pdflatex'
```

```
})


class HelloKitty:
    def __init__(self, vol):
        """Create the data and a "Hello Kitty" plot.

        Both the plots and the raw data is saved to file, and the
        ``PlotHK`` class can reproduce the plots based on the
        saved data.

        In `config`, set
        ```
            'F_MIN': 2.5e6, 'F_MAX': 9.5e6
        ```
        Also, using
        ```
            F_N_POINTS = 1e4
        ```
         is sufficient.

        Args:
            vol {int or float} -- choose between two different
            input settings, creating two different HK plots
        """
        self.vol = int(vol)
        if self.vol == 1:
            self.Z = np.linspace(1e11, 8e11, 60)
        else:
            self.Z = np.linspace(2e11, 1e12, 60)
        self.A = 45 + 15 * np.cos(np.linspace(0, np.pi, 30))
        self.fr = np.zeros((len(self.Z), len(self.A)))
        self.g = np.zeros((len(self.Z), len(self.A)))
        self.dots = [[], [], []]
        self.meta = []
        self.F0 = 430e6
        self.K_RADAR = - 2 * self.F0 * 2 * np.pi / const.c  # Radar
        ↪  wavenumber
        save = input('Press "y/yes" to save plot, ' + \
                     'any other key to dismiss.\t').lower()
        if save in ['y', 'yes']:
            self.save = True
        else:
            self.save = False

    def create_data(self):
        if self.vol == 1:
            sys_set = {'K_RADAR': self.K_RADAR, 'B': 35000e-9, 'MI': 16,
                       'NE': 2e10, 'NU_E': 100, 'NU_I': 100, 'T_E': 2000,
                       'T_I': 1500, 'T_ES': 90000,
                       'THETA': 60 * np.pi / 180, 'Z': 599,
                       'mat_file': 'fe_zmuE-07.mat',
                       'pitch_angle': list(range(10))}
        else:
```

```
85              sys_set = {'K_RADAR': self.K_RADAR, 'B': 35000e-9, 'MI': 16,
86                         'NE': 2e10, 'NU_E': 100, 'NU_I': 100, 'T_E': 2000,
87                         'T_I': 1500, 'T_ES': 90000,
88                         'THETA': 60 * np.pi / 180, 'Z': 300,
89                         'mat_file': 'fe_zmuE-07.mat',
90                         'pitch_angle': 'all'}
91          params = {'kappa': 8, 'vdf': 'real_data', 'area': False}
92          with tqdm(total=len(self.Z) * len(self.A)) as pbar:
93              for i, z in enumerate(self.Z):
94                  sys_set['NE'] = z
95                  plasma_freq = (sys_set['NE'] * const.elementary_charge**2 /
96                                 (const.m_e * const.epsilon_0))**.5 / (2 *
                                   ↪ np.pi)
97                  cf.I_P['F_MIN'] = plasma_freq
98                  cf.I_P['F_MAX'] = plasma_freq + 4e5
99                  cf.f = np.linspace(cf.I_P['F_MIN'], cf.I_P['F_MAX'],
                    ↪ int(cf.F_N_POINTS))
100                 cf.w = 2 * np.pi * cf.f  # Angular frequency
101                 for j, a in enumerate(self.A):
102                     sys_set['THETA'] = a * np.pi / 180
103                     old_stdout = sys.stdout
104                     f = open(os.devnull, 'w')
105                     sys.stdout = f
106                     f, s, meta_data = isr.isr_spectrum('a_vdf', sys_set,
                        ↪ **params)
107                     sys.stdout = old_stdout
108                     plasma_power, energy_interval, fr = self.check_energy(f,
                        ↪ s, a)
109                     if energy_interval != 0:
110                         self.dots[0].append(energy_interval)
111                         self.dots[1].append(j)
112                         self.dots[2].append(z)
113                     self.fr[i, j] = fr
114                     self.g[i, j] = plasma_power
115                     pbar.update(1)
116         self.meta.append(meta_data)
117
118     def check_energy(self, f, s, deg):
119         p = int(np.argwhere(s==np.max(s)))
120         freq = f[p]
121         f_mask = (freq - 5e2 < f) & (f < freq + 5e2)
122         x = f[f_mask]
123         y = s[f_mask]
124         mod = LorentzianModel()
125         pars = mod.guess(y, x=x)
126         out = mod.fit(y, pars, x=x)
127         power = si.simps(out.best_fit, x)
128
129         l = const.c / self.F0
130         # Calculate corresponding energy with formula:
            ↪ E = 0.5 m_e [f_r λ_R / (2 cos θ)]^2
131         E_plasma = .5 * const.m_e * (freq * l / (2 * np.cos(deg * np.pi /
            ↪ 180)))**2 / const.eV
132         res = 0
```

```
        if self.vol == 1:
            if bool(15.58 < E_plasma < 18.42):
                res = 1
            elif bool(22.47 < E_plasma < 23.75):
                res = 2
        else:
            if bool(20.29 < E_plasma < 22.05):
                res = 1
            elif bool(22.45 < E_plasma < 23.87):
                res = 2
            elif bool(25.38 < E_plasma < 27.14):
                res = 3
        return power, res, freq

    def plot_data(self):
        # Hello kitty figure duplication
        self.g = np.c_[self.g, self.g[:, ::-1], self.g, self.g[:, ::-1]]
        self.A = np.r_[self.A, self.A[::-1], self.A, self.A[::-1]]
        dots_x = []
        dots_y = []
        for i, d in enumerate(self.dots[1]):
            arg = np.argwhere(self.A == self.A[d])
            dots_x = np.r_[dots_x, arg[:2, 0]]
            dots_y = np.r_[dots_y, np.ones(len(arg[:2, 0])) *
            ↪  self.dots[2][i]]

        f = plt.figure(figsize=(6, 4))
        gs = gridspec.GridSpec(2, 1, height_ratios=[4, 1])
        ax0 = plt.subplot(gs[0])
        im = ax0.imshow(self.g, extent=[0, len(self.A) - 1,
                                        np.min(self.Z), np.max(self.Z)],
                        origin='lower', aspect='auto', cmap='gist_heat')
        plt.scatter(dots_x, dots_y, s=3)
        plt.ylabel(r'Electron number density, $n_{\mathrm{e}}$')
        plt.tick_params(axis='x', which='both', bottom=False,
                        top=False, labelbottom=False)
        ax1 = plt.subplot(gs[1])
        ax1.plot(self.A)
        plt.xlim([0, len(self.A) - 1])
        plt.yticks([30, 45, 60])
        plt.ylabel('Aspect angle')
        axs = []
        axs += [ax0]
        axs += [ax1]
        gs.update(hspace=0.05)
        f.colorbar(im, ax=axs).ax.set_ylabel('Echo power')
        plt.tick_params(axis='x', which='both', bottom=False,
                        top=False, labelbottom=False)

        if self.save:
            save_path = '../../../report/master-thesis/figures'
            if not os.path.exists(save_path):
                save_path = '../figures'
                os.makedirs(save_path, exist_ok=True)
```

```
186             tt = time.localtime()
187             the_time = f'{tt[0]}_{tt[1]}_{tt[2]}_{tt[3]}--{tt[4]}--{tt[5]}'
188             save_path = f'{save_path}/hello_kitty_{the_time}'
189             self.meta.insert(0, {'F_MAX': cf.I_P['F_MAX'], 'V_MAX':
                ↪  cf.V_MAX,
190                                  'F_N_POINTS': cf.F_N_POINTS, 'Y_N_POINTS':
                                     ↪  cf.Y_N_POINTS,
191                                  'V_N_POINTS': cf.V_N_POINTS})
192
193             pdffig = PdfPages(str(save_path) + '.pdf')
194             metadata = pdffig.infodict()
195             metadata['Title'] = f'Hello Kitty plot'
196             metadata['Author'] = 'Eirik R. Enger'
197             metadata['Subject'] = f"Plasma line power as a function of ' + \
198                                    'electron number density and aspect angle."
199             metadata['Keywords'] = f'{self.meta}'
200             metadata['ModDate'] = datetime.datetime.today()
201             pdffig.attach_note('max(s), 100percent power')
202             plt.savefig(pdffig, bbox_inches='tight', format='pdf', dpi=600)
203             pdffig.close()
204             plt.savefig(f'{save_path}.pgf', bbox_inches='tight',
                ↪  metadata=self.meta)
205             np.savez(f'{save_path}', angle=self.A, density=self.Z,
                ↪  power=self.g, dots=self.dots, fr=self.fr)
206
207         plt.show()
208
209     def run(self):
210         self.create_data()
211         self.plot_data()
```

A.5 spectrum_calculation.py

```
1  """Script containing the calculation of the power density spectrum
2  and other plasma parameters.
3  """
4
5  import os
6  import sys
7
8  import numpy as np
9  import scipy.constants as const
10 import scipy.integrate as si
11
12 from inputs import config as cf
13 from utils import integrand_functions as intf
14 from utils.parallel import gordeyev_int_parallel
15
16
```

```
17 def isr_spectrum(version, system_set, kappa=None, vdf=None, area=False,
   ↪ debye=None):
18     """Calculate an ISR spectrum using the theory
19     presented by Hagfors [1961] and Mace [2003].
20
21     Arguments:
22         version {str} -- decide which integral to use when
23         calculating ISR spectrum
24         system_set {dict} -- all plasma parameters and other parameters
25         needed in the different calculation methods
26
27     Keyword Arguments:
28         kappa {int} -- kappa index used in any kappa distribution
29         (default: {None})
30         vdf {str} -- gives the VDF used in the a_vdf calculation
31         (default: {None})
32         area {bool} -- if True, calculates the area under the ion line
33         (default: {False})
34         debye {str} -- if set to `maxwell`, the Maxwellian Debye length
35         is used (default: {None})
36
37     Returns:
38         f {np.ndarray} -- 1D array giving the frequency axis
39         Is {np.ndarray} -- 1D array giving the spectrum at
40         the sampled frequencies
41         meta_data {dict} -- all parameters used to calculate
42         the returned spectrum
43     """
44     sys_set, p = correct_inputs(version, system_set.copy(), {'kappa': kappa,
       ↪ 'vdf': vdf})
45     kappa, vdf = p['kappa'], p['vdf']
46     func = version_check(version, vdf, kappa)
47     w_c = w_e_gyro(np.linalg.norm([sys_set['B']], 2))
48     M_i = sys_set['MI'] * (const.m_p + const.m_n) / 2
49     W_c = w_ion_gyro(np.linalg.norm([sys_set['B']], 2), M_i)
50
51     # Ions
52     params = {'K_RADAR': sys_set['K_RADAR'], 'THETA': sys_set['THETA'],
53               'nu': sys_set['NU_I'], 'm': M_i, 'T': sys_set['T_I'], 'w_c':
                  ↪ W_c}
54     y = np.linspace(0, cf.Y_MAX_i**(1 / cf.ORDER), int(cf.Y_N_POINTS),
       ↪ dtype=np.double)**cf.ORDER
55     f_ion = intf.INT_MAXWELL()
56     f_ion.initialize(y, params)
57     Fi = gordeyev_int_parallel.integrate(M_i, sys_set['T_I'],
       ↪ sys_set['NU_I'], y, function=f_ion, kappa=kappa)
58
59     # Electrons
60     params = {'K_RADAR': sys_set['K_RADAR'], 'THETA': sys_set['THETA'],
61               'nu': sys_set['NU_E'], 'm': const.m_e, 'T': sys_set['T_E'],
62               'T_ES': sys_set['T_ES'], 'w_c': w_c, 'kappa': kappa, 'vdf':
                  ↪ vdf,
63               'Z': sys_set['Z'], 'mat_file': sys_set['mat_file'],
64               'pitch_angle': sys_set['pitch_angle']}
```

```
65      y = np.linspace(0, cf.Y_MAX_e**(1 / cf.ORDER), int(cf.Y_N_POINTS),
        ↪  dtype=np.double)**cf.ORDER
66      func.initialize(y, params)
67      Fe = gordeyev_int_parallel.integrate(const.m_e, sys_set['T_E'],
        ↪  sys_set['NU_E'], y, function=func, kappa=kappa)
68
69      Xp_i = np.sqrt(1 / (2 * L_Debye(sys_set['NE'], sys_set['T_E'],
        ↪  kappa=None)**2 * \
70             sys_set['K_RADAR']**2))
71      if func.the_type == 'maxwell' or debye == 'maxwell':
72          Xp_e = np.sqrt(1 / (2 * L_Debye(sys_set['NE'], sys_set['T_E'])**2 *
            ↪  \
73                 sys_set['K_RADAR']**2))
74      elif func.the_type == 'kappa':
75          Xp_e = np.sqrt(1 / (2 * L_Debye(sys_set['NE'], sys_set['T_E'],
            ↪  kappa=kappa)**2 * \
76                 sys_set['K_RADAR']**2))
77      elif func.the_type == 'a_vdf':
78          Xp_e = np.sqrt(1 / (2 * L_Debye(sys_set['NE'], sys_set['T_E'],
            ↪  char_vel=func.char_vel)**2 * \
79                 sys_set['K_RADAR']**2))
80
81      # In case we have ω = 0 in our frequency array, we just ignore this
        ↪  warning message
82      with np.errstate(divide='ignore', invalid='ignore'):
83          Is = sys_set['NE'] / (np.pi * cf.w) * (np.imag(- Fe) * abs(1 + 2 *
            ↪  Xp_i**2 * Fi)**2 + (
84              4 * Xp_e**4 * np.imag(- Fi) * abs(Fe)**2)) / abs(1 + 2 * Xp_e**2
                ↪  * Fe + 2 * Xp_i**2 * Fi)**2
85
86      if area:
87          if cf.I_P['F_MAX'] < 1e4:
88              area = si.simps(Is, cf.f)
89              print('The area under the ion line is %1.6e.' % area)
90          else:
91              print('F_MAX is set too high. The area was not calculated.')
92
93      sys_set['THETA'] = round(params['THETA'] * 180 / np.pi, 1)
94      sys_set['version'] = version
95      return cf.f, Is, dict(sys_set, **p)
96
97
98  def L_Debye(*args, kappa=None, char_vel=None):
99      """Calculate the Debye length.
100
101     Input args may be
102         n_e -- electron number density
103         T_e -- electron temperature
104         T_i -- ion temperature
105
106     Returns:
107         float -- the Debye length
108     """
109     nargin = len(args)
```

```
110     if nargin == 1:
111         n_e = args[0]
112     elif nargin == 2:
113         n_e = args[0]
114         T_e = args[1]
115     elif nargin == 3:
116         n_e = args[0]
117         T_e = args[1]
118         T_i = args[2]
119 
120     Ep0 = 1e-09 / 36 / np.pi
121 
122     if nargin < 3:
123         if kappa is not None:
124             LD = np.sqrt(Ep0 * const.k * T_e / (max(0, n_e) * const.e**2)
125                          ) * np.sqrt((kappa - 3 / 2) / (kappa - 1 / 2))
126         elif char_vel is not None:
127             LD = np.sqrt(Ep0 * const.k * T_e / (max(0, n_e) * const.e**2)
128                          ) * np.sqrt(char_vel)
129         else:
130             LD = np.sqrt(Ep0 * const.k * T_e /
131                          (max(0, n_e) * const.e**2))
132     else:
133         LD = np.sqrt(Ep0 * const.k /
134                      ((max(0, n_e) / T_e + max(0, n_e) / T_i) / const.e**2))
135 
136     return LD
137 
138 
139 def w_ion_gyro(B, m_ion):
140     """Ion gyro frequency as a function of
141     magnetic field strength and ion mass.
142 
143     Arguments:
144         B {float} -- magnetic field strength
145         m_ion {float} -- ion mass
146 
147     Returns:
148         float -- ion gyro frequency
149     """
150     w_e = const.e * B / m_ion
151 
152     return w_e
153 
154 
155 def w_e_gyro(B):
156     """Electron gyro frequency as a function of magnetic field strength.
157 
158     Arguments:
159         B {float} -- magnetic field strength
160 
161     Returns:
162         float -- electron gyro frequency
163     """
```

```
164     w_e = const.e * B / const.m_e
165
166     return w_e
167
168
169 def correct_inputs(version, sys_set, params):
170     """Extra check suppressing the parameters
171     that was given but is not necessary.
172     """
173     if version != 'kappa' and not (version == 'a_vdf' and params['vdf'] in
        ↪  ['kappa', 'kappa_vol2']):
174         params['kappa'] = None
175     if version != 'a_vdf':
176         params['vdf'] = None
177     if version != 'a_vdf' or params['vdf'] != 'gauss_shell':
178         sys_set['T_ES'] = None
179     if version != 'a_vdf' or params['vdf'] != 'real_data':
180         sys_set['Z'] = None
181         sys_set['mat_file'] = None
182         sys_set['pitch_angle'] = None
183     return sys_set, params
184
185
186 def version_check(version, vdf, kappa):
187     """Check if the parameters given are complete.
188
189     Args:
190         version {str} -- which Gordeyev integrand to use
191         vdf {str} -- which distribution to use
192         kappa {int or float} -- kappa index
193
194     Returns:
195         object -- an integrand object from `integrand_functions.py`
196     """
197     versions = ['kappa', 'maxwell', 'a_vdf']
198     try:
199         if not version in versions:
200             raise SystemError
201         print(f'Using version "{version}"', flush=True)
202     except SystemError:
203         sys.exit(version_error(version, versions))
204     if version == 'maxwell':
205         func = intf.INT_MAXWELL()
206     elif version == 'kappa':
207         kappa_check(kappa)
208         func = intf.INT_KAPPA()
209     elif version == 'a_vdf':
210         vdfs = ['maxwell', 'kappa', 'kappa_vol2', 'gauss_shell',
            ↪  'real_data']
211         try:
212             if not vdf in vdfs:
213                 raise SystemError
214             print(f'Using VDF "{vdf}"', flush=True)
215         except Exception:
```

```
            sys.exit(version_error(vdf, vdfs, element='VDF'))
        if vdf in ['kappa', 'kappa_vol2']:
            kappa_check(kappa)
        func = intf.INT_LONG()
    return func


def version_error(version, versions, element='version'):
    exc_type, _, exc_tb = sys.exc_info()
    fname = os.path.split(exc_tb.tb_frame.f_code.co_filename)[1]
    print(f'{exc_type} error in file {fname}, line {exc_tb.tb_lineno}')
    print(f'The {element} is wrong: "{version}" not found in {versions}')


def kappa_check(kappa):
    try:
        kappa = int(kappa)
    except SystemError:
        sys.exit(print('You did not send in a valid kappa index.'))
```

A.6 integrand_functions.py

```
"""Script containing the integrands used in the Gordeyev integral.
"""

from abc import ABC, abstractmethod, abstractproperty

import numpy as np
import scipy.constants as const
import scipy.special as sps
import scipy.integrate as si

from inputs import config as cf
from utils import vdfs
from utils.parallel import v_int_parallel


class INTEGRAND(ABC):
    """Base class for an integrand object.

    Arguments:
        ABC {ABC} -- abstract base class
    """
    @abstractproperty
    def the_type(self) -> str:
        """The type of the intregrand implementation.
        """

    @abstractmethod
    def initialize(self, y, params):
```

```
29          """Needs an initialization method.
30
31          Arguments:
32              y {np.ndarray} -- array for integration variable
33              params {dict} -- dictionary holding all needed parameters
34          """
35
36      @abstractmethod
37      def integrand(self):
38          """Method that returns the np.ndarray that is used as the integrand.
39          """
40
41
42  class INT_KAPPA(INTEGRAND):
43      """Implementation of the integrand of the Gordeyev
44      integral for the kappa distribution from Mace (2003).
45
46      Arguments:
47          INTEGRAND {ABC} -- base class used to create integrand objects
48      """
49      the_type = 'kappa'
50
51      def __init__(self):
52          self.y = np.array([])
53          self.params = {}
54          self.Z = float
55          self.Kn = float
56
57      def initialize(self, y, params):
58          self.y = y
59          self.params = params
60          self.z_func()
61
62      def z_func(self):
63          theta_2 = 2 * ((self.params['kappa'] - 3 / 2) /
            ↪  self.params['kappa']) * self.params['T'] * const.k /
            ↪  self.params['m']
64          self.Z = (2 * self.params['kappa'])**(1 / 2) * \
65              (self.params['K_RADAR']**2 * np.sin(self.params['THETA'])**2 *
                ↪  theta_2 / self.params['w_c']**2 *
66               (1 - np.cos(self.params['w_c'] * self.y)) +
67               1 / 2 * self.params['K_RADAR']**2 *
                 ↪  np.cos(self.params['THETA'])**2 * theta_2 * self.y**2)**(1
                 ↪  / 2)
68          self.Kn = sps.kv(self.params['kappa'] + 1 / 2, self.Z)
69          self.Kn[self.Kn == np.inf] = 1
70
71      def integrand(self):
72          G = self.Z**(self.params['kappa'] + .5) * self.Kn * np.exp(- self.y
            ↪  * self.params['nu'])
73
74          return G
75
76
```

```
77  class INT_MAXWELL(INTEGRAND):
78      """Implementation of the intregrand in the Gordeyev
79      integral for the Maxwellian distribution from
80      e.g. Hagfors (1961) or Mace (2003).
81
82      Arguments:
83          INTEGRAND {ABC} -- base class used to create integrand objects
84      """
85      the_type = 'maxwell'
86
87      def __init__(self):
88          self.y = np.array([])
89          self.params = {}
90
91      def initialize(self, y, params):
92          self.y = y
93          self.params = params
94
95      def integrand(self):
96          G = np.exp(- self.y * self.params['nu'] -
97                     self.params['K_RADAR']**2 *
                       ↪  np.sin(self.params['THETA'])**2 * self.params['T'] *
                       ↪  const.k /
98                     (self.params['m'] * self.params['w_c']**2) * (1 -
                       ↪  np.cos(self.params['w_c'] * self.y)) -
99                     .5 * (self.params['K_RADAR'] *
                       ↪  np.cos(self.params['THETA']) * self.y)**2 *
                       ↪  self.params['T'] * const.k / self.params['m'])
100
101         return G
102
103
104 class INT_LONG(INTEGRAND):
105     """Implementation of the intregrand in the Gordeyev
106     integral for the isotropic distribution from Mace (2003).
107
108     Arguments:
109         INTEGRAND {ABC} -- base class used to create integrand objects
110     """
111     the_type = 'a_vdf'
112
113     def __init__(self):
114         self.y = np.array([])
115         self.params = {}
116         self.char_vel = float
117
118     def initialize(self, y, params):
119         self.y = y
120         self.params = params
121
122     def v_int(self):
123         v = np.linspace(0, cf.V_MAX**(1 / cf.ORDER),
            ↪  int(cf.V_N_POINTS))**cf.ORDER
124         if self.params['vdf'] == 'maxwell':
```

```
125             f = vdfs.F_MAXWELL(v, self.params)
126         elif self.params['vdf'] == 'kappa':
127             f = vdfs.F_KAPPA(v, self.params)
128         elif self.params['vdf'] == 'kappa_vol2':
129             f = vdfs.F_KAPPA_2(v, self.params)
130         elif self.params['vdf'] == 'gauss_shell':
131             f = vdfs.F_GAUSS_SHELL(v, self.params)
132         elif self.params['vdf'] == 'real_data':
133             f = vdfs.F_REAL_DATA(v, self.params)
134 
135         # Compare the velocity integral to the Maxwellian case.
136         # This way we make up for the change in characteristic velocity
137         # and Debye length for different particle distributions.
138         res_maxwell = v_int_parallel.integrand(self.y, self.params, v,
            ↪ vdfs.F_MAXWELL(v, self.params).f_0())
139         int_maxwell = si.simps(res_maxwell, self.y)
140         res = v_int_parallel.integrand(self.y, self.params, v, f.f_0())
141         int_res = si.simps(res, self.y)
142         # The scaling of the factor describing the characteristic velocity
143         self.char_vel = int_maxwell / int_res
144         print(f'Debye length of the current distribution is {self.char_vel}'
            ↪ + \
145               'times the Maxwellian Debye length.')
146         return res
147 
148     def p_d(self):
149         # At y = 0 we get 0/0, so we use
150         # lim_{y→0+} dp/dy = |k||w_c|/√(w_c^2) (from above, opposite sign from
            ↪ below)
151         cos_t = np.cos(self.params['THETA'])
152         sin_t = np.sin(self.params['THETA'])
153         w_c = self.params['w_c']
154         num = abs(self.params['K_RADAR']) * abs(w_c) * \
155                   (cos_t**2 * w_c * self.y + sin_t**2 * np.sin(w_c *
                      ↪ self.y))
156         den = w_c * (cos_t**2 * w_c**2 * self.y**2 -
157                      2 * sin_t**2 * np.cos(w_c * self.y) +
158                      2 * sin_t**2)**.5
159         # np.sign(y[-1]) takes care of weather the limit should be
            ↪ considered taken from above or below.
160         # The last element of the np.ndarray is chosen since it is assumed y
            ↪ runs from 0 to some finite real number.
161         first = np.sign(self.y[-1]) * abs(self.params['K_RADAR']) * abs(w_c)
            ↪ / abs(w_c)
162         with np.errstate(divide='ignore', invalid='ignore'):
163             out = num / den
164         out[np.where(den == 0.)[0]] = first
165 
166         return out
167 
168     def integrand(self):
169         return self.p_d() * self.v_int()
```

A.7 vdfs.py

```
"""Velocity distribution function used in the version a_vdf,
one of the integrands available for use in the Gordeyev integral.

Any new VDF must be added as an option in
the a_vdf function in integrand_functions.py.
"""

from abc import ABC, abstractmethod

import numpy as np
import scipy.constants as const
import scipy.special as sps
import scipy.integrate as si

from utils import read


class VDF(ABC):
    """Base class for a VDF object.

    Arguments:
        ABC {class} -- abstract base class that all VDF objects inherit from
    """
    @abstractmethod
    def normalize(self):
        """Calculate the normalization for the VDF.
        """

    @abstractmethod
    def f_0(self):
        """Return the values along the velocity axis of a VDF.
        """


class F_MAXWELL(VDF):
    """Create an object that make Maxwellian distribution functions.

    Arguments:
        VDF {ABC} -- abstract base class to make VDF objects
    """
    def __init__(self, v, params):
        self.v = v
        self.params = params
        self.normalize()

    def normalize(self):
        self.A = (2 * np.pi * self.params['T'] * const.k /
        ↪  self.params['m'])**(- 3 / 2)

    def f_0(self):
```

```
50          func = self.A * np.exp(- self.v**2 / (2 * self.params['T'] * const.k
            ↪  / self.params['m']))
51
52          return func
53
54
55  class F_KAPPA(VDF):
56      """Create an object that make kappa distribution functions.
57
58      Arguments:
59          VDF {ABC} -- abstract base class to make VDF objects
60      """
61      def __init__(self, v, params):
62          """Initialize VDF parameters.
63
64          Arguments:
65              v {np.ndarray} -- 1D array with the sampled velocities
66              params {dict} -- a dictionary with all needed plasma parameters
67          """
68          self.v = v
69          self.params = params
70          self.normalize()
71
72      def normalize(self):
73          self.theta_2 = 2 * ((self.params['kappa'] - 3 / 2) /
            ↪  self.params['kappa']) * self.params['T'] * const.k /
            ↪  self.params['m']
74          self.A = (np.pi * self.params['kappa'] * self.theta_2)**(- 3 / 2) *
            ↪  \
75              sps.gamma(self.params['kappa'] + 1) /
                ↪  sps.gamma(self.params['kappa'] - 1 / 2)
76
77      def f_0(self):
78          """Return the values along velocity `v` of a kappa VDF.
79
80          Kappa VDF used in Gordeyev paper by Mace (2003).
81
82          Returns:
83              np.ndarray -- 1D array with the VDF values at the sampled points
84          """
85          func = self.A * (1 + self.v**2 / (self.params['kappa'] *
            ↪  self.theta_2))**(- self.params['kappa'] - 1)
86
87          return func
88
89
90  class F_KAPPA_2(VDF):
91      """Create an object that make kappa vol. 2 distribution functions.
92
93      Arguments:
94          VDF {ABC} -- abstract base class to make VDF objects
95      """
96      def __init__(self, v, params):
97          """Initialize VDF parameters.
```

```
98
99          Arguments:
100             v {np.ndarray} -- 1D array with the sampled velocities
101             params {dict} -- a dictionary with all needed plasma parameters
102         """
103         self.v = v
104         self.params = params
105         self.normalize()
106
107     def normalize(self):
108         self.v_th = np.sqrt(self.params['T'] * const.k / self.params['m'])
109         self.A = (np.pi * self.params['kappa'] * self.v_th**2)**(- 3 / 2) *
            ↪ \
110             sps.gamma(self.params['kappa']) / sps.gamma(self.params['kappa']
                ↪ - 3 / 2)
111
112     def f_0(self):
113         """Return the values along velocity `v` of a kappa VDF.
114
115         Kappa VDF used in dispersion relation paper by
116         Ziebell, Gaelzer and Simoes (2017). Defined by
117         Leubner (2002) (sec 3.2).
118
119         Returns:
120             np.ndarray -- 1D array with the VDF values at the sampled points
121         """
122         func = self.A * (1 + self.v**2 / (self.params['kappa'] *
            ↪ self.v_th**2))**(- self.params['kappa'])
123
124         return func
125
126
127 class F_GAUSS_SHELL(VDF):
128     """Create an object that make Gauss shell distribution functions.
129
130     Arguments:
131         VDF {ABC} -- abstract base class to make VDF objects
132     """
133     def __init__(self, v, params):
134         self.v = v
135         self.params = params
136         self.vth = np.sqrt(self.params['T'] * const.k / self.params['m'])
137         self.r = (self.params['T_ES'] * const.k / self.params['m'])**.5
138         self.steep = 5
139         self.f_M = F_MAXWELL(self.v, self.params)
140         self.normalize()
141
142     def normalize(self):
143         func = np.exp(- self.steep * (abs(self.v) - self.r)**2 / (2 *
            ↪ self.params['T'] * const.k / self.params['m']))
144         f = func * self.v**2 * 4 * np.pi
145         self.A = 1 / si.simps(f, self.v)
146         ev = .5 * const.m_e * self.r**2 / const.eV
147         print(f'Gauss shell at E = {round(ev, 2)} eV')
```

```
148
149     def f_0(self):
150         func = self.A * np.exp(- self.steep * (abs(self.v) - self.r)**2 / (2
            ↪  * self.params['T'] * const.k / self.params['m'])) + \
151                1e4 * self.f_M.f_0()
152
153         return func / (1e4 + 1)
154
155
156 class F_REAL_DATA(VDF):
157     """Create an object that make distribution functions from
158     a 1D array.
159
160     Arguments:
161         VDF {ABC} -- abstract base class to make VDF objects
162     """
163     def __init__(self, v, params):
164         self.v = v
165         self.params = params
166         self.normalize()
167
168     def normalize(self):
169         func = read.interpolate_data(self.v, self.params)
170         f = func * self.v**2 * 4 * np.pi
171         self.A = 1 / si.simps(f, self.v)
172
173     def f_0(self):
174         func = self.A * read.interpolate_data(self.v, self.params)
175
176         return func
```

A.8 read.py

```
1  """This script reads from folder `arecibo` and combines the
2  calculated electron distribution from file with a Maxwellian.
3  """
4
5  import os
6  import sys
7
8  import ast
9  import numpy as np
10 from scipy.io import loadmat
11 import scipy.constants as const
12
13
14 def f_0_maxwell(v, params):
15     # NOTE: Normalized to 1D
16     A = (2 * np.pi * params['T'] * const.k / params['m'])**(- 1 / 2)
17     func = A * np.exp(- v**2 / (2 * params['T'] * const.k / params['m']))
```

```
18      return func
19
20
21  def interpolate_data(v, params):
22      """Interpolate calculated distribution down to zero
23      energy and add to a 1D Maxwellian.
24
25      Args:
26          v {np.ndarray} -- 1D velocity array
27          params {dict} -- dictionary of all needed parameters
28
29      Returns:
30          np.ndarray -- 1D array of the distribution
31      """
32      if os.path.basename(os.path.realpath(sys.argv[0])) != 'main.py':
33          path = 'data/arecibo/'
34          if not os.path.exists(path):
35              path = 'program/data/arecibo/'
36          x = loadmat(path + params['mat_file'])
37          data = x['fe_zmuE']
38          if isinstance(params['pitch_angle'], list):
39              if all(isinstance(x, int) for x in params['pitch_angle']):
40                  sum_over_pitch = data[:, params['pitch_angle'], :]
41                  norm = len(params['pitch_angle'])
42          else:
43              norm = 18
44          sum_over_pitch = np.einsum('ijk->ik', data) / norm  # removes
            ↪  j-dimansion through dot-product
45          idx = int(np.argwhere(read_dat_file('z4fe.dat')==params['Z']))
46          f_1 = sum_over_pitch[idx, :]
47          energies = read_dat_file('E4fe.dat')
48      else:
49          path = 'data/arecibo/'
50          x = loadmat(path + params['mat_file'])
51          data = x['fe_zmuE']
52          if isinstance(params['pitch_angle'], list):
53              if all(isinstance(x, int) for x in params['pitch_angle']):
54                  sum_over_pitch = data[:, params['pitch_angle'], :]
55                  norm = len(params['pitch_angle'])
56          else:
57              norm = 18
58          sum_over_pitch = np.einsum('ijk->ik', data) / norm  # removes
            ↪  j-dimansion through dot-product
59          idx = int(np.argwhere(read_dat_file('z4fe.dat')==params['Z']))
60          f_1 = sum_over_pitch[idx, :]
61          energies = read_dat_file('E4fe.dat')
62
63      velocities = (2 * energies * const.eV / params['m'])**.5
64      new_f1 = np.interp(v, velocities, f_1)
65      f_0 = f_0_maxwell(v, params)
66      f0_f1 = f_0 + new_f1
67
68      return f0_f1
69
```

```
70
71  def read_dat_file(file):
72      """Return the contents of a `.dat` file as a single numpy row vector.
73
74      Arguments:
75          file {str} -- the file name of the .dat file
76
77      Returns:
78          np.ndarray -- contents of the .dat file
79      """
80      l = np.array([])
81      path = 'data/arecibo/'
82      if not os.path.exists(path):
83          path = 'program/data/arecibo/'
84      with open(path + file) as f:
85          ll = f.readlines()
86          ll = [x.strip() for x in ll]
87          l = np.r_[l, ll]
88      if len(l) == 1:
89          for p in l:
90              l = p.split()
91      e = []
92      for p in l:
93          k = ast.literal_eval(p)
94          e.append(k)
95      return np.array(e)
```

A.9 test_ISR.py

```
1   """This script implements tests for
2   functions used throughout the program.
3
4   Run from directory `program` with command
5   python -m unittest test.test_ISR -b
6   """
7
8   import multiprocessing as mp
9   mp.set_start_method('fork')
10
11  import unittest  # pylint: disable=C0413
12  import numpy as np  # pylint: disable=C0413
13  import scipy.integrate as si  # pylint: disable=C0413
14  import scipy.constants as const  # pylint: disable=C0413
15
16  from utils import spectrum_calculation as isr  # pylint: disable=C0413
17  from utils import vdfs  # pylint: disable=C0413
18
19
20  class TestISR(unittest.TestCase):
21      """Check if the output from isr_spectrum is as expected.
```

```
22
23      Should return two numpy.ndarrays of equal shape.
24
25      Arguments:
26          unittest.TestCase {class} -- inherits from unittest
27          to make it a TestCase
28      """
29
30      @classmethod
31      def setUpClass(cls):
32          cls.a, cls.b = None, None
33
34      def setUp(self):
35          F0 = 430e6
36          K_RADAR = - 2 * F0 * 2 * np.pi / const.c
37          self.sys_set = {'K_RADAR': K_RADAR, 'B': 5e-4, 'MI': 16,
38                          'NE': 2e11, 'NU_E': 0, 'NU_I': 0,
39                          'T_E': 5000, 'T_I': 2000, 'T_ES': 90000,
40                          'THETA': 40 * np.pi / 180, 'Z': 599,
41                          'mat_file': 'fe_zmuE-07.mat', 'pitch_angle': 'all'}
42          self.params = {'kappa': 3, 'vdf': 'gauss_shell', 'area': False}
43
44      def tearDown(self):
45          self.assertIsInstance(self.a, np.ndarray)
46          self.assertIsInstance(self.b, np.ndarray)
47          self.assertEqual(self.a.shape, self.b.shape,
            ↪  msg='a.shape != b.shape')
48
49      def test_isr_maxwell(self):
50          self.a, self.b, meta_data = isr.isr_spectrum('maxwell',
            ↪  self.sys_set, **self.params)
51          self.assertEqual(meta_data['kappa'], None)
52          self.assertEqual(meta_data['vdf'], None)
53          self.assertEqual(meta_data['T_ES'], None)
54          self.assertEqual(meta_data['Z'], None)
55          self.assertEqual(meta_data['mat_file'], None)
56
57      def test_isr_kappa(self):
58          self.a, self.b, meta_data = isr.isr_spectrum('kappa', self.sys_set,
            ↪  **self.params)
59          self.assertEqual(meta_data['kappa'], 3)
60          self.assertEqual(meta_data['vdf'], None)
61          self.assertEqual(meta_data['T_ES'], None)
62          self.assertEqual(meta_data['Z'], None)
63          self.assertEqual(meta_data['mat_file'], None)
64
65      def test_isr_long_calc_gauss(self):
66          self.a, self.b, meta_data = isr.isr_spectrum('a_vdf', self.sys_set,
            ↪  **self.params)
67          self.assertEqual(meta_data['kappa'], None)
68          self.assertEqual(meta_data['vdf'], 'gauss_shell')
69          self.assertEqual(meta_data['T_ES'], 90000)
70          self.assertEqual(meta_data['Z'], None)
71          self.assertEqual(meta_data['mat_file'], None)
```

```
72
73      def test_isr_long_calc_real(self):
74          self.params['vdf'] = 'real_data'
75          self.a, self.b, meta_data = isr.isr_spectrum('a_vdf', self.sys_set,
            ↪  **self.params)
76          self.assertEqual(meta_data['kappa'], None)
77          self.assertEqual(meta_data['vdf'], 'real_data')
78          self.assertEqual(meta_data['T_ES'], None)
79          self.assertEqual(meta_data['Z'], 599)
80          self.assertEqual(meta_data['mat_file'], 'fe_zmuE-07.mat')
81
82
83  # Reference to TestVDF
84  class TestVDF(unittest.TestCase):
85      """Class which test if the VDFs are normalized.
86
87      Arguments:
88          unittest.TestCase {class} -- inherits from unittest
89          to make it a TestCase
90      """
91
92      @classmethod
93      def setUpClass(cls):
94          cls.v = np.linspace(0, (6e6)**(1 / 3), int(4e4))**3
95          cls.params = {'m': 9.1093837015e-31, 'T': 1000,
96                        'kappa': 3, 'T_ES': 90000, 'Z': 300,
97                        'mat_file': 'fe_zmuE-07.mat', 'pitch_angle': 'all'}
98          cls.f = None
99          # cls.fs = []
100
101     # @classmethod
102     # def tearDownClass(cls):
103     #     np.savez('f', v=cls.v, m=cls.fs[0], k=cls.fs[1], r=cls.fs[2])
104
105     def tearDown(self):
106         # The function f is scaled with the Jacobian of cartesian to
            ↪  spherical
107         f = self.f.f_0() * self.v**2 * 4 * np.pi
108         res = si.simps(f, self.v)
109         self.assertAlmostEqual(res, 1, places=6)
110
111     def test_vdf_maxwell(self):
112         self.f = vdfs.F_MAXWELL(self.v, self.params)
113         # self.fs.insert(0, self.f.f_0())
114
115     def test_vdf_kappa(self):
116         self.f = vdfs.F_KAPPA(self.v, self.params)
117         # self.fs.insert(1, self.f.f_0())
118
119     # def test_vdf_kappa_vol2(self):
120     #     self.f = vdfs.F_KAPPA_2(self.v, self.params)
121
122     # def test_vdf_gauss_shell(self):
123     #     self.f = vdfs.F_GAUSS_SHELL(self.v, self.params)
```

```
124
125     def test_vdf_real_data(self):
126         self.f = vdfs.F_REAL_DATA(self.v, self.params)
127         # self.fs.insert(2, self.f.f_0())
128
129
130 if __name__ == '__main__':
131     unittest.main()
```

A.10 gordeyev_int_parallel.py

```
1  """Implementation of parallel computation of
2  the Gordeyev integral as a function of frequency.
3  """
4
5  import ctypes
6  import multiprocessing as mp
7  from functools import partial
8
9  import numpy as np
10 import scipy.special as sps
11 import scipy.constants as const
12 import scipy.integrate as si
13
14 from inputs import config as cf
15
16
17 def integrate(m, T, nu, y, function, kappa=None):
18     """Integrate from `0` to `Y_MAX` with an integrand on the form
19     `e^{-iwy}f(y)`, for every value in the np.ndarray `w`.
20
21     Arguments:
22         m {float} -- mass [kg]
23         T {float} -- temperature [K]
24         nu {float} -- collision frequency [Hz]
25         y {np.ndarray} -- integration sample points
26         function {class object} -- object from an integrand class
27
28     Keyword Arguments:
29         kappa {int or float} -- index determining the order of the
30             kappa VDFs (default: {None})
31
32     Returns:
33         np.ndarray -- a scaled version of the result from the
34             integration based on Hagfors [1968]
35     """
36     idx = set(enumerate(cf.w))
37     f = function.integrand()
38     func = partial(parallel, y, f)
39     pool = mp.Pool()
```

```
    pool.map(func, idx)
    pool.close()
    if function.the_type == 'kappa':  #
        a = array / (2**(kappa - 1 / 2) * sps.gamma(kappa + 1 / 2))
    elif function.the_type == 'a_vdf':
        # Characteristic velocity scaling
        a = 4 * np.pi * T * const.k * array / m * function.char_vel
    else:
        a = array
    if function.the_type == 'a_vdf':
        F = a
    else:
        F = 1 - (1j * cf.w + nu) * a
    return F


def parallel(y, f, index):
    array[index[0]] = simpson(index[1], y, f)


def simpson(w, y, f):
    val = np.exp(- 1j * w * y) * f

    sint = si.simps(val, y)
    return sint


def shared_array(shape):
    """
    Form a shared memory numpy array.

    https://tinyurl.com/c9m75k2
    """

    shared_array_base = mp.Array(ctypes.c_double, 2 * shape[0])
    shared_arr = np.ctypeslib.as_array(shared_array_base.get_obj())
    shared_arr = shared_arr.view(np.complex128).reshape(*shape)
    return shared_arr


# F_N_POINTS = N_f
array = shared_array((int(cf.F_N_POINTS),))
```

A.11 v_int_parallel.py

```
"""Implementation of parallel computation of the
velocity integrals as a function of the integral
variable y from the Gordeyev integral.
"""

```

```python
import ctypes
import multiprocessing as mp
from functools import partial

import numpy as np
import scipy.integrate as si

from inputs import config as cf


def integrand(y, params, v, f):
    """Integrate from `0` to `V_MAX` with an integrand on
    the form `e^{-iwt}f(t)`, for every value in the np.ndarray `y`.

    Arguments:
        y {np.ndarray} -- sample points of integration variable
            from Gordeyev integral
        params {dict} -- plasma parameters
        v {np.ndarray} -- sample points of VDF
        f {np.ndarray} -- value of VDF at sample points

    Returns:
        np.ndarray -- the value of the velocity integral at every
        sample of the integration variable
    """
    idx = set(enumerate(y))
    func = partial(parallel, params, v, f)
    pool = mp.Pool()
    pool.map(func, idx)
    pool.close()
    return array


def parallel(params, v, f, index):
    array[index[0]] = v_int_integrand(index[1], params, v, f)


# Velocity integral
def v_int_integrand(y, params, v, f):
    sin = np.sin(p(y, params) * v)
    val = v * sin * f
    res = si.simps(val, v)
    return res


def p(y, params):
    """From Mace [2003].

    Args:
        y {np.ndarray} -- parameter from Gordeyev integral
        params {dict} -- plasma parameters

    Returns:
        np.ndarray -- value of the `p` function
```

```
    """
    k_perp = params['K_RADAR'] * np.sin(params['THETA'])
    k_par = params['K_RADAR'] * np.cos(params['THETA'])
    return (2 * k_perp**2 / params['w_c']**2 * (1 - np.cos(y *
    ↪  params['w_c'])) + k_par**2 * y**2)**.5


def shared_array(shape):
    """
    Form a shared memory numpy array.

    https://tinyurl.com/c9m75k2
    """

    shared_array_base = mp.Array(ctypes.c_double, shape[0])
    shared_arr = np.ctypeslib.as_array(shared_array_base.get_obj())
    shared_arr = shared_arr.view(np.double).reshape(*shape)
    return shared_arr


# Y_N_POINTS = N_y
array = shared_array((int(cf.Y_N_POINTS),))
```

A.12 plot_class.py

```
"""Class containing two plotting styles used in `reproduce.py`.
"""

import os
import time
import datetime
import itertools

import matplotlib.gridspec as grid_spec
import matplotlib.pyplot as plt
from matplotlib.backends.backend_pdf import PdfPages
import numpy as np
import scipy.signal as signal
import si_prefix as sip

from inputs import config as cf

class PlotClass:
    """Create a plot object to show the data created.
    """

    def __init__(self):
        """Make plots of an IS spectrum based on a variety of VDFs.

        Keyword Arguments:
```

```
            plasma {bool} -- choose to plot only the part of the
                spectrum where the plasma line is found (default: {False})
        """
        self.save = input('Press "y/yes" to save plot, ' + \
                          'any other key to dismiss.\t').lower()
        self.page = 1
        self.plasma = False
        self.pdffig = None
        self.save_path = None
        self.correct_inputs()
        self.colors = ['k', 'magenta', 'royalblue', 'yellow',
                       'chartreuse', 'firebrick', 'red', 'darkorange']
        self.line_styles = ['-', '--', '-.', ':',
                            (0, (3, 5, 1, 5, 1, 5)),
                            (0, (3, 1, 1, 1, 1, 1))]

    def __setattr__(self, name, value):
        self.__dict__[name] = value
        self.correct_inputs()

    # TODO: probably not needed anymore
    def correct_inputs(self):
        """Extra check suppressing the parameters
        that was given but is not necessary.
        """
        try:
            if not isinstance(self.plasma, bool):
                self.plasma = False
        except Exception:
            pass

    def save_it(self, f, data, l_txt, r_txt, params):
        """Save the figure as a multi page pdf with all
        parameters saved in the meta data, and as one
        pgf file for each page.

        The date and time is used in the figure name, in addition
        to it ending with which method was used. The settings that
        was used in config as inputs to the plot object is saved
        in the metadata of the figure.

        If a figure is created from file, the same file name is used.
        """
        version = ''
        for d in params:
            if 'version' in d:
                if any(c.isalpha() for c in version):
                    version += f'_{d["version"][0]}'
                else:
                    version += f'{d["version"][0]}'
        if self.save_path is None:
            params.insert(0, {'F_MIN': cf.I_P['F_MIN'], 'F_MAX':
            ↪  cf.I_P['F_MAX'],
```

```
78                                  'V_MAX': cf.V_MAX, 'F_N_POINTS':
                                    ↪  cf.F_N_POINTS,
79                                  'Y_N_POINTS': cf.Y_N_POINTS, 'V_N_POINTS':
                                    ↪  cf.V_N_POINTS})
80          tt = time.localtime()
81          the_time = f'{tt[0]}_{tt[1]}_{tt[2]}_{tt[3]}--{tt[4]}--{tt[5]}'
82          save_path = '../../../report/master-thesis/figures/in_use'
83          if not os.path.exists(save_path):
84              save_path = '../figures'
85              os.makedirs(save_path, exist_ok=True)
86          if self.save_path is None:
87              self.save_path = f'{save_path}/{the_time}_{version}'
88          else:
89              self.save_path = save_path + '/' + self.save_path
90          np.savez(f'{self.save_path}', frequency=f, spectra=data,
            ↪  legend_txt=l_txt, ridge_txt=r_txt, meta=params)
91          self.pdffig = PdfPages(str(self.save_path) + '.pdf')
92          metadata = self.pdffig.infodict()
93          metadata['Title'] = f'ISR Spectrum w/ {version}'
94          metadata['Author'] = 'Eirik R. Enger'
95          metadata['Subject'] =
            ↪  f"IS spectrum made using a {version} distribution ' + \
96                                  'and Simpson's integration rule."
97          metadata['Keywords'] = f'{params}'
98          metadata['ModDate'] = datetime.datetime.today()
99
100     def plot_normal(self, f, Is, func_type, l_txt):
101         """Make a plot using `f` as `x` axis and `Is` as `y` axis.
102
103         Arguments:
104             f {np.ndarray} -- variable along x axis
105             Is {list} -- list of np.ndarrays that give the y axis
106                 values along x axis
107             func_type {str} -- attribute of the matplotlib.pyplot object
108             l_txt {list} -- a list of strings that give the legend
109                 of the spectra. Same length as the inner lists
110         """
111         try:
112             getattr(plt, func_type)
113         except Exception:
114             print(f'{func_type} is not an attribute of the ' + \
115                   'matplotlib.pyplot object. Using "plot".')
116             func_type = 'plot'
117         if len(Is) != len(l_txt):
118             print('Warning: The number of spectra does ' + \
119                   'not match the number of labels.')
120         self.colors = np.linspace(0, 1, len(Is))
121         Is = Is.copy()
122         # TODO: should probably remove this
123         # Linear plot show only ion line (kHz range).
124         if func_type == 'plot' and not self.plasma:
125             f, Is = self.only_ionline(f, Is)
126         p, freq, exp = self.scale_f(f)
127         plt.figure(figsize=(6, 3))
```

```
128         if self.plasma:
129             # Clip the frequency axis around the plasma frequency.
130             mask = self.find_p_line(freq * 10**exp, Is)
131             freq = freq[mask]
132         if func_type == 'semilogy':
133             plt.xlabel(f'Frequency [{p}Hz]')
134             plt.ylabel('Echo power [dB]')
135             for i, _ in enumerate(Is):
136                 Is[i] = 10 * np.log10(Is[i])
137         else:
138             plt.xlabel(f'Frequency [{p}Hz]')
139             plt.ylabel('Echo power')
140         for clr, st, s, lab in zip(itertools.cycle(self.colors),
        ↪  itertools.cycle(self.line_styles), Is, l_txt):
141             if self.plasma:
142                 s = s[mask]
143             if func_type == 'semilogy':
144                 plt.plot(freq, s, linestyle=st, alpha=.7, color=(clr, 0.,
                    ↪  0.),  # color=clr,
145                          linewidth=.8, label=lab)
146             else:
147                 plot_object = getattr(plt, func_type)
148                 plot_object(freq, s, linestyle=st, alpha=.7, color=(clr, 0.,
                    ↪  0.),  # color=clr,
149                             linewidth=.8, label=lab)
150
151         plt.legend()
152         plt.minorticks_on()
153         plt.grid(True, which="major", ls="-", alpha=0.4)
154         plt.tight_layout()
155
156         if self.save in ['y', 'yes']:
157             self.pdffig.attach_note(func_type)
158             plt.savefig(self.pdffig, bbox_inches='tight', format='pdf',
                ↪  dpi=600)
159             plt.savefig(str(self.save_path) + f'_page_{self.page}.pgf',
                ↪  bbox_inches='tight')
160             self.page += 1
161
162     def plot_ridge(self, frequency, multi_parameters, func_type, l_txt,
        ↪  ridge_txt=None):
163         """Make a ridge plot of several spectra.
164
165         Arguments:
166             frequency {np.ndarray} -- frequency axis
167             multi_parameters {list} -- list (outer) containing
168                 lists (inner) of np.ndarrays. The arrays
169                 contain the spectrum values at the frequencies
170                 given by "frequency"
171             func_type {str} -- attribute of the matplotlib.pyplot class
172             l_txt {list} -- a list of strings that give the legend of the
173                 spectra. Same length as the inner lists
174
175         Keyword Arguments:
```

```
            ridge_txt {list} -- list of strings that give the text to the left
                of all ridges. Same length as outer list or None (default: {None})
        """
        # Inspired by https://tinyurl.com/y9p5gewr
        try:
            getattr(plt, func_type)
        except Exception:
            print(f'{func_type} is not an attribute of the ' + \
                  'matplotlib.pyplot object. Using "plot".')
            func_type = 'plot'
        if len(multi_parameters) != len(ridge_txt):
            print('Warning: The list of spectra lists is not of the same ' +
            ↪  \
                  'length as the length of "ridge_txt"')
            if len(multi_parameters) > len(ridge_txt):
                for _ in range(len(multi_parameters) - len(ridge_txt)):
                    ridge_txt.append('')
        f_original = frequency.copy()
        multi_params = multi_parameters.copy()
        # Reverse the order to put the first elements at the bottom of the
        ↪  figure
        multi_params.reverse()
        ridge_txt = ridge_txt.copy()
        if ridge_txt is None:
            ridge_txt = ['' for _ in multi_params]
        else:
            ridge_txt.reverse()
        gs = grid_spec.GridSpec(len(multi_params), 1)
        fig = plt.figure(figsize=(7, 9))
        ax_objs = []
        Rgb = np.linspace(0, 1, len(multi_params))
        for j, params in enumerate(multi_params):
            if len(params) != len(l_txt):
                print('Warning: The number of spectra ' + \
                      'does not match the number of labels.')
            # f is reset due to the scaling of 'plot' below
            f = f_original
            # Linear plot show only ion line (kHz range).
            if func_type == 'plot' and not self.plasma:
                f, params = self.only_ionline(f, params)
            p, freq, exp = self.scale_f(f)
            if self.plasma:
                mask = self.find_p_line(freq * 10**exp, params)
                freq = freq[mask]
            # Make a new subplot / ridge
            ax_objs.append(fig.add_subplot(gs[j:j + 1, 0:]))
            first = 0
            for st, s, lab in zip(itertools.cycle(self.line_styles), params,
            ↪  l_txt):
                if self.plasma:
                    s = s[mask]
                plot_object = getattr(ax_objs[-1], func_type)
                plot_object(freq, s, color=(Rgb[j], 0., 1 - Rgb[j]),
                ↪  linewidth=1, label=lab, linestyle=st)
```

```
226                 if first == 0:
227                     idx = np.argwhere(freq > ax_objs[-1].viewLim.x0)[0]
228                     legend_pos = (ax_objs[-1].viewLim.x1, np.max(s))
229                     y0 = s[idx]
230                     ax_objs[-1].text(freq[idx], s[idx], ridge_txt[j],
231                                      fontsize=14, ha="right", va='bottom')
232                 first += 1
233                 if j == 0:
234                     plt.legend(loc='upper right', bbox_to_anchor=legend_pos,
                    ↪  bbox_transform=ax_objs[-1].transData)
235
236             if func_type == 'plot':
237                 # Make a vertical line of comparable size in all plots.
238                 self.match_box(f_original, freq, multi_params, [y0, j])
239
240             self.remove_background(ax_objs[-1], multi_params, j, p)
241
242         gs.update(hspace=-0.6)
243         if self.save in ['y', 'yes']:
244             self.pdffig.attach_note(func_type)
245             plt.savefig(self.pdffig, bbox_inches='tight', format='pdf',
            ↪  dpi=600)
246             plt.savefig(str(self.save_path) + f'_page_{self.page}.pgf',
            ↪  bbox_inches='tight')
247             self.page += 1
248
249     @staticmethod
250     def remove_background(plt_obj, multi_params, j, p):
251         # Make the background transparent
252         rect = plt_obj.patch
253         rect.set_alpha(0)
254         # Remove borders, axis ticks and labels
255         plt_obj.set_yticklabels([])
256         plt.tick_params(axis='y', which='both', left=False,
257                         right=False, labelleft=False)
258         if j == len(multi_params) - 1:
259             plt.xlabel(f'Frequency [{p}Hz]')
260         else:
261             plt.tick_params(axis='x', which='both', bottom=False,
262                             top=False, labelbottom=False)
263
264         spines = ["top", "right", "left", "bottom"]
265         for sp in spines:
266             plt_obj.spines[sp].set_visible(False)
267
268     @staticmethod
269     def scale_f(frequency):
270         """Scale the axis and add the corresponding SI prefix.
271
272         Arguments:
273             frequency {np.ndarray} -- the variable along an axis
274
275         Returns:
276             str, np.ndarray, int -- the prefix, the scaled variables, the
```

```
277                                     exponent corresponding to the prefix
278         """
279         freq = np.copy(frequency)
280         exp = sip.split(np.max(freq))[1]
281         freq /= 10**exp
282         pre = sip.prefix(exp)
283         return pre, freq, exp
284
285     @staticmethod
286     def find_p_line(freq, spectrum):
287         """Find the frequency that is most likely the peak
288         of the plasma line and return the lower and upper
289         bounds for an interval around the peak.
290
291         Arguments:
292             freq {np.ndarray} -- sample points of frequency parameter
293             spectrum {list} -- list of np.ndarray, values of spectrum
294                                at the sampled frequencies
295
296         Keyword Arguments:
297             check {bool} -- used in correct_inputs to check if plasma
298                             plots are possible (default: {False})
299
300         Returns:
301             np.ndarray -- array with boolean elements
302         """
303         spec = spectrum[0]
304         try:
305             # Assumes that the rightmost peak (highest frequency) is the
                ↪  plasma line
306             p = signal.find_peaks(spec, height=10)[0][-1]
307         except Exception:
308             print('Warning: did not find any plasma line')
309             return freq < np.inf
310         f = freq[p]
311
312         lower, upper = f - 1e6, f + 1e6
313
314         # Don't want the ion line to ruin the scaling of the y axis
315         if lower < 1e5:
316             lower = 1e5
317         return (freq > lower) & (freq < upper)
318
319     @staticmethod
320     def only_ionline(f, Is):
321         Is = Is.copy()
322         idx = np.argwhere(abs(f) < 4e4)
323         if len(idx) < 3:
324             return f, Is
325         f = f[idx].reshape((-1,))
326         for i, _ in enumerate(Is):
327             Is[i] = Is[i][idx].reshape((-1,))
328         return f, Is
329
```

```
330     def match_box(self, freq_original, freq, multi_parameters, args):
331         """Create a scaling box for easier comparison of the ridges.
332 
333         Should cover as much as possible in the ridge that span the
334         smallest range along the `y` axis.
335 
336         Args:
337             freq_original {np.ndarray} -- frequency axis
338             freq {np.ndarray} -- copy of the frequency axis
339             multi_parameters {list} -- list of the spectra
340             args {list} -- zeroth element is y_min and
341                 first is the index for the ridge
342         """
343         multi_params = multi_parameters.copy()
344         v_line_x = np.linspace(.04, .2, len(multi_params))
345         if self.plasma:
346             f = freq_original.copy()
347             spec = multi_params[0]
348             mask = self.find_p_line(f, spec)
349         diff = np.inf
350         for params in multi_params:
351             plot_diff = 0
352             for s in params:
353                 if self.plasma:
354                     s = s[mask]
355                 difference = np.max(s) - np.min(s)
356                 if plot_diff < difference:
357                     plot_diff = difference
358             if plot_diff < diff:
359                 diff = plot_diff
360 
361         x0 = np.min(freq) + (np.max(freq) - np.min(freq)) *
            ↪  v_line_x[args[1]]
362         plt.vlines(x=x0, ymin=args[0], ymax=args[0] + int(np.ceil(diff / 10)
            ↪  * 5), color='k', linewidth=3)
363         plt.text(x0, args[0] + int(np.ceil(diff / 10) * 5) / 2,
364                  r'${}$'.format(int(np.ceil(diff / 10) * 5)), rotation=90,
                     ↪  ha='right', va='center')
```

Bibliography

Bernstein, I. B. (1958). Waves in a Plasma in a Magnetic Field. *Phys. Rev.*, 109(1):10–21.

Beynon, W. J. G. and Williams, P. J. S. (1978). Incoherent scatter of radio waves from the ionosphere. *Reports on progress in physics*, 41(6):909–955.

Bittencourt, J. A. (2004). *Fundamentals of plasma physics*. Springer, New York, 3rd edition.

Bjørnå, N., Esjeholm, B.-T., and Hansen, T. (1990). Gyro line observations with the EISCAT VHF radar. *Journal of Atmospheric and Terrestrial Physics*, 52(6-8):473–482.

Chen, F. F. (1984). *Introduction to plasma physics and controlled fusion: Volume 1: Plasma physics*, volume 1. Plenum Press, New York, 2nd edition.

Djuth, F. T., Carlson, H. C., and Zhang, L. D. (2018). Incoherent Scatter Radar Studies of Daytime Plasma Lines. *Earth, Moon, and Planets*, 121(1):13–43.

Djuth, F. T., Sulzer, M. P., and Elder, J. H. (1994). Application of the coded long-pulse technique to plasma line studies of the ionosphere. *Geophysical Research Letters*, 21(24):2725–2728.

Dougherty, J. P. and Farley, D. T. (1960). A Theory of Incoherent Scattering of Radio Waves by a Plasma. *Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences (1934-1990)*, 259(1296):79–99.

Fredriksen, Å., Bjørnå, N., and Lilensten, J. (1992). Incoherent scatter plasma lines at angles with the magnetic field. *Journal of Geophysical Research*, 97(A11):16921.

Gaelzer, R., Ziebell, L. F., and Meneses, A. R. (2016). The general dielectric tensor for bi-kappa magnetized plasmas. *Physics of Plasmas*, 23(6):062108.

Gordon, W. (1958). Incoherent Scattering of Radio Waves by Free Electrons with Applications to Space Exploration by Radar. *Proceedings of the IRE*, 46(11):1824–1829.

Guio, P. (1998). *Studies of ionospheric parameters by means of electron plasma lines observed by EISCAT*. Doctorat thesis, University of Tromsø.

Guio, P., Lilensten, J., Kofman, W., and Bjørnå, N. (1998). Electron velocity distribution function in a plasma with temperature gradient and in the presence of suprathermal electrons: application to incoherent-scatter plasma lines. *Annales geophysicae*, 16(10):1226–1240.

Hagfors, T. (1961). Density Fluctuations in a Plasma in a Magnetic Field, with Applications to the Ionosphere. *Journal of Geophysical Research*, 66(6):1699–1712.

Hellberg, M. A., Mace, R. L., Baluku, T. K., Kourakis, I., and Saini, N. S. (2009). Comment on "Mathematical and physical aspects of Kappa velocity distribution" [Phys. Plasmas 14, 110702 (2007)]. *Physics of Plasmas*, 16(9):094701.

Kudeki, E. and Milla, M. A. (2011). Incoherent Scatter Spectral Theories—Part I: A General Framework and Results for Small Magnetic Aspect Angles. *IEEE Transactions on Geoscience and Remote Sensing*, 49(1):315–328.

LaLonde, L. M. (1974). The Upgraded Arecibo Observatory. *Science*, 186(4160):213–218.

Li, Y. L., Liu, C. H., and Franke, S. J. (1991). Adaptive evaluation of the Sommerfeld-type integral using the chirp z-transform. *IEEE Transactions on Antennas and Propagation*, 39(12):1788–1791.

Livadiotis, G. and McComas, D. J. (2010). EXPLORING TRANSITIONS OF SPACE PLASMAS OUT OF EQUILIBRIUM. *The Astrophysical Journal*, 714(1):971–987.

Livadiotis, G. and McComas, D. J. (2011). INVARIANT KAPPA DISTRIBUTION IN SPACE PLASMAS OUT OF EQUILIBRIUM. *The Astrophysical Journal*, 741(2):88.

Mace, R. L. (2003). A Gordeyev integral for electrostatic waves in a magnetized plasma with a kappa velocity distribution. *Physics of Plasmas*, 10(6):2181–2193.

Mace, R. L. and Hellberg, M. A. (1995). A dispersion function for plasmas containing superthermal particles. *Physics of Plasmas*, 2(6):2098–2109.

Nicolls, M. J., Sulzer, M. P., Aponte, N., Seal, R., Nikoukar, R., and González, S. A. (2006). High-resolution electron temperature measurements using the plasma line asymmetry. *Geophysical Research Letters*, 33(18):L18107.

Olbert, S. (1968). Summary of Experimental Results from M.I.T. Detector on IMP-1. In Carovillano, R. L., McClay, J. F., and Radoski, H. R., editors, *Physics of the Magnetosphere*, pages 641–659, Dordrecht. Springer Netherlands.

Perkins, F. and Salpeter, E. E. (1965). Enhancement of Plasma Density Fluctuations by Nonthermal Electrons. *Physical review.*, 139(1A):A55–A62.

Rees, M. H. (1989). *Physics and chemistry of the upper atmosphere*, volume 1 of *Cambridge atmosperic and space science series*. Cambridge University Press, Cambridge Cambridgeshire.

Saito, S., Forme, F. R. E., Buchert, S. C., Nozawa, S., and Fujii, R. (2000). Effects of a kappa distribution function of electrons on incoherent scatter spectra. *Annales Geophysicae*, 18(9):1216–1223.

Salpeter, E. E. (1960a). Electron Density Fluctuations in a Plasma. *Physical review*, 120(5):1528–1535.

Salpeter, E. E. (1960b). Scattering of radio waves by electrons above the ionosphere. *Journal of Geophysical Research*, 65(6):1851–1852.

Salpeter, E. E. (1961). Plasma Density Fluctuations in a Magnetic Field. *Physical Review*, 122(6):1663–1674.

Showen, R. L. (1979). The spectral measurement of plasma lines. *Radio Science*, 14(3):503–508.

Thorne, R. M. and Summers, D. (1991). Landau damping in space plasmas. *Physics of Fluids B: Plasma Physics*, 3(8):2117–2123.

Vierinen, J., Gustavsson, B., Hysell, D. L., Sulzer, M. P., Perillat, P., and Kudeki, E. (2017). Radar observations of thermal plasma oscillations in the ionosphere. *Geophysical Research Letters*, 44(11):5301–5307.

Yngvesson, K. O. and Perkins, F. W. (1968). Radar Thomson scatter studies of photoelectrons in the ionosphere and Landau damping. *Journal of geophysical research*, 73(1):97–110.

Ziebell, L. F., Gaelzer, R., and Simões, F. J. R. (2017). Dispersion relation for electrostatic waves in plasmas with isotropic and anisotropic Kappa distributions

for electrons and ions. *Journal of Plasma Physics*, 83(5):905830503.