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

Abstract iii

Acknowledgements v

List of Figures ix

List of Tables xi

List of Symbols xiii

List of Abbreviations xv

1 Introduction 1
   1.1 Motivation ................................................. 3
   1.2 Thesis structure ......................................... 5

2 Background 7
   2.1 Derivation of the incoherent scatter spectrum .......... 7
      2.1.1 Fourier transforms ................................. 8
      2.1.2 Ensemble average ................................. 8
      2.1.3 Scattering cross section ........................... 9
      2.1.4 Fluctuations ................................. 10
      2.1.5 Spectral distribution ............................ 14
   2.2 Suprathermal electrons ................................. 20
   2.3 Numerical description of suprathermal distributions ..... 22

3 Derivation of dielectric functions 23
   3.1 The kappa distribution function ........................ 23
   3.2 Dielectric function for the kappa distribution ........ 24
   3.3 Dielectric function for isotropic distributions ........ 27
   3.4 Alternative derivation for isotropic distributions ....... 30
   3.5 Alternative versions of the kappa distribution ........ 30

4 Implementation in computer code 33
4.1 Evaluating the Gordeyev integral using the Simpson’s rule . . 33
4.2 Implementation of calculated electron distributions . . . . . 36
4.3 Testing the numerical precision . . . . . . . . . . . . . . . . . . 37
4.4 Evaluating the Gordeyev integral using the chirp z-transform 42

5 Results from model calculations of IS spectra 45
5.1 Spectra from Maxwellian and kappa distributions . . . . . 45
5.2 The plasma lines . . . . . . . . . . . . . . . . . . . . . . . . . . . 48
5.3 Plasma line power structures at Arecibo Observatory . . . 52
  5.3.1 Measurements . . . . . . . . . . . . . . . . . . . . . . . . . . . 52
  5.3.2 Comparison with numerical model . . . . . . . . . . . . . 54
  5.3.3 Results compared to measurements by Djuth . . . . . . 61

6 Conclusion 63
  6.1 Future work . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 64

A Source code 67
A.1 main.py . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 68
A.2 config.py . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 71
A.3 reproduce.py . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 72
A.4 hello_kitty.py . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 84
A.5 spectrum_calculation.py . . . . . . . . . . . . . . . . . . . . . . . . . . . . 88
A.6 integrand_functions.py . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 93
A.7 vdfs.py . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 97
A.8 read.py . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 100
A.9 test_ISR.py . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 102
A.10 gordeyev_int_parallel.py . . . . . . . . . . . . . . . . . . . . . . . . . . . 105
A.11 v_int_parallel.py . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 106
A.12 plot_class.py . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 108

Bibliography 117
List of Figures

2.1 Coordinate system for solving eq. (2.35) .......................... 15

3.1 Velocity distribution functions ..................................... 25

4.1 Integrand of Gordeyev integral ................................. 34
4.2 Sampling method .................................................. 35
4.3 Superimposing of thermal and suprathermal distributions ... 36
4.4 Comparison: semi-analytic and numerical calculation ....... 37
4.5 Comparison: high precision in velocity integral ................. 39
4.6 Comparison: high precision in Gordeyev integral .............. 39
4.7 Comparison: high precision in both integrals .................. 40
4.8 Comparison: low \( v_{\text{max}} \) .................................... 41
4.9 Chirp z-transform peak frequencies ............................ 42

5.1 IS spectra for a Maxwellian and three kappa distributions ... 46
5.2 Ion line of the IS spectrum ....................................... 47
5.3 Plasma line of the IS spectrum .................................. 49
5.4 Plasma line with changing temperature ......................... 50
5.5 Plasma line peak frequency as a function of temperature .... 51
5.6 Plasma line power structure measurement ..................... 53
5.7 Input to electron transport code .................................. 55
5.8 Maxwellian, kappa and calculated distribution ................ 56
5.9 Plasma line power as a function of \( \theta \) and \( n_e \) ............... 57
5.10 Suprathermal distribution used in fig. 5.9 ....................... 58
5.11 Plasma line power as a function of \( \theta \) and \( n_e \) ............... 59
5.12 Suprathermal distribution used in fig. 5.11 ..................... 60
List of Tables

5.1 Plasma parameters for fig. 5.1 . . . . . . . . . . . . . . . . . . . . . . 46
5.2 Plasma parameters for fig. 5.3 . . . . . . . . . . . . . . . . . . . . . . 49
5.3 Plasma parameters for fig. 5.9 . . . . . . . . . . . . . . . . . . . . . . 57
5.4 Plasma parameters for fig. 5.11 . . . . . . . . . . . . . . . . . . . . . 59
List of Symbols

⟨·⟩ Ensemble average of ·

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

T Auto-correlation function

$\mathcal{F}\{\cdot\}$ Fourier transform of ·

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

$\Re\{\cdot\}$ and $\Im\{\cdot\}$ Real and imaginary part of ·

$\epsilon$ Dielectric function

$\chi$ Susceptibility of a dielectric

$f$ Phase-space density distribution function, frequency when specified

$f_0$ Velocity distribution function

$f_r$ Radar frequency

$f_R$ Resonance frequency of an ion or plasma wave

$g$ Gordeyev integral

$n_\alpha$ Number density of particle species $\alpha$

$\rho$ Charge density

$\alpha$ Electrons or ions ($\alpha = e, i$)

$q$ Elementary charge
\( \mathbf{r} \) Distance vector

\( \mathbf{k} \) Wave vector

\( E \) Electric field vector

\( \mathbf{B} \) Magnetic field vector

\( \theta \) Aspect angle, angle between the magnetic field line and the radar wave vector

\( \omega \) Angular frequency

\( \omega_{pa} \) Angular plasma frequency of particle species \( \alpha \)

\( \Omega_\alpha \) Angular gyro frequency of particle species \( \alpha \)

\( \nu_{\alpha} \) Collision frequency of particle species \( \alpha \)

\( v_{\text{th},\alpha} \) Thermal speed of particle species \( \alpha \)

\( C_s \) Ion sound speed

\( \lambda_D \) Debye length

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

\( m_\alpha \) Mass of particle species \( \alpha \)

\( T_\alpha \) Temperature of particle species \( \alpha \)

\( k_B \) Boltzmann’s constant

\( \varepsilon_0 \) 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
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. 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 $\omega_{pe}$. 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)

$$\omega_{R,e} := \Re\{\omega_e\} = (\omega_{pe}^2 + 3k^2v_{th,e}^2 + \Omega_e^2 \sin^2 \theta)^{1/2},$$

(1.1)

where $k$ is the wave number, $v_{th,e}$ is the thermal velocity, $\Omega_e$ is the electron gyro frequency and $\theta$ is the angle between the radar wave vector and the magnetic field line known as the aspect angle. $\omega_e$ 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)

$$k_\pm = \frac{1}{c} [\omega_r + (\omega_r \pm \omega_{R,e})]$$

(1.2)

where $\pm$ is for the up- and downshifted waves, $c$ is the speed of light and $\omega_r$ 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 $\omega_r \pm \omega_{R,e}$, where $\omega_{R,e} \approx \omega_{pe} \ll \omega_r$ (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

$$\omega_{R,i} := \Re\{\omega_i\} = kC_s$$

(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 $\omega_{Ki}$ (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 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, $\Delta f_R = f_{R+} - f_{R-}$. 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, $f_R(\theta) = A(\cos \theta)^{0.97}$, where $A$ is a normalization constant. This was an empirically derived formula using a value of $B$ 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.
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, \( \mathbf{k} \), 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 \( \Psi \) of space and time, may be defined as

\[
\mathcal{F}_T \{ \Psi(r, t) \} = \Psi(r, \omega) = \int_T \Psi(r, t) \exp(-i\omega t) dt \tag{2.1a}
\]

\[
\mathcal{F}_V \{ \Psi(r, t) \} = \Psi(k, t) = \int_V \Psi(r, t) \exp(i\mathbf{k} \cdot \mathbf{r}) d^3 r \tag{2.1b}
\]

where \( \mathbf{r} \) 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

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

\[
\mathcal{F}_V^{-1} \{ \Psi(k, t) \} = \Psi(r, t) = \frac{1}{(2\pi)^3} \int_K \Psi(k, t) \exp(-i\mathbf{k} \cdot \mathbf{r}) d^3 k \tag{2.2b}
\]

where \( \Omega \) is the span of frequencies, \( \omega \), that is integrated over and \( K \) is the span of wave vectors, \( \mathbf{k} \), that is integrated over. This gives a transformation for time and space as

\[
\mathcal{F}_{V,T} \{ \Psi(r, t) \} = \Psi(k, \omega) = \int_V \int_T \Psi(r, t) \exp(\mathbf{k} \cdot \mathbf{r} - \omega t) dtd^3 r. \tag{2.3}
\]

The subscripts on the Fourier transform symbol, \( \mathcal{F} \), 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 \( \langle |\Psi(k, t + \tau)|^2 \rangle \), i.e. the
notation $\langle \cdot \rangle$ define an ensemble average. Further, we write
\[
\langle \Psi(k, t + \tau) \Psi^*(k, t) \rangle = \int_V \int_V \langle \Psi(r_1, t + \tau) \Psi^*(r_2, t) \rangle \\
\times \exp[i k \cdot r_1] \exp[-i k \cdot r_2] d^3 r_1 d^3 r_2
\]
\[
= \int_V \int_V \langle \Psi(r + r', t + \tau) \Psi^*(r, t) \rangle \\
\times \exp[i k \cdot r'] d^3 r d^3 r'
\]
and let $r_1 \to r + r'$ and $r_2 \to r$. The expected value is assumed to be independent of $r$ and $t$, i.e. the assumptions of homogeneity and stationarity are applied. This makes the first integral over $r$ trivial, yielding
\[
\langle \Psi(k, t + \tau) \Psi^*(k, t) \rangle = V \langle |\Psi(r, t)|^2 \rangle \int_V \Upsilon(r', \tau) \exp[k \cdot r'] dr'
\]
where $\Upsilon(r', \tau)$ is the ACF of $\Psi$ 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:
\[
\langle |\Psi(k, \omega)|^2 \rangle = T \int_T \langle \Psi(k, \tau) \rangle \exp[-i \omega \tau] d\tau
\]
\[
= VT \langle |\Psi(k, t)|^2 \rangle \int_T \Upsilon(k, \tau) \exp[-i \omega \tau] d\tau
\]
which is defined as the power density spectrum of the function $\Psi$, and where we have the normalization such that $\Upsilon(k, 0) \equiv 1$.

### 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)
\[
\sigma = \sigma_e V \langle |n_e(k)|^2 \rangle
\]
where $\sigma_e$ is the single electron scattering cross section per unit solid angle and per unit incident power density, and where $k$ is the difference between the
wave vectors of the incident radar wave \((k_r)\) and the scattered wave, i.e.

\[
k = \left(-k_r + \frac{\pm \omega_R}{c}\right) - k_t, \tag{2.8}
\]

where \(\pm\) is for up- and downshifted waves, \(\omega_R\) is the angular resonance frequency and direction of the ionospheric wave and \(c\) is the speed of light. Due to the \(\pm\) on the resonance frequency, the wave vector will in general have the subscript \(\pm\) for up- and downshifted waves, but this is omitted. \(n_e(k)\) is the number density of electrons as a function of wave vector, defined as the Fourier transform of \(n_e(r)\) through eq. (2.1b) as

\[
n_e(k) = \frac{1}{V} \int_V n_e(r) \exp[ik \cdot r] \, dr \tag{2.9}
\]

where \(n_e(r)\) 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

\[
\sigma(\omega) = \sigma_0 V \langle |n_e(k, \omega)|^2 \rangle. \tag{2.10}
\]

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

\[
\sigma = \int_{-\infty}^{\infty} \sigma(\omega) \, d\omega. \tag{2.11}
\]

### 2.1.4 Fluctuations

We assume fluctuations in a neutral plasma and that the average number density of ions and electrons are \(n_{i,0}\) and \(n_{e,0}\). The number of charges on the ions (to make things neutral) is then \(Z := n_{e,0}/n_{i,0}\). 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

\[
n_e(r) = \sum_{j=1}^{n_{e,0}V} \delta(r - r_{e,j}) \tag{2.12a}
\]

\[
n_i(r) = \sum_{j=1}^{n_{i,0}V} \delta(r - r_{i,j}). \tag{2.12b}
\]

\(r_{e,j}\) and \(r_{i,j}\) are the positions of all the electrons and ions. Charge density becomes

\[
\rho(r) = q [Zn_i(r) - n_e(r)] \tag{2.13}
\]

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

\[
\rho(k) = q [Zn_i(k) - n_e(k)] \tag{2.14}
\]
where we let \( k = 2\pi(\ell_1, \ell_2, \ell_3)/L, \ell_j \in \mathbb{Z} \) 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, \( E. \) \( E \) is a function of \( r, \) 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 \( E \) can be derived from a scalar electrical potential (Hagfors, 1961). From Poisson’s equation:

\[
E(k) = \frac{ik}{\varepsilon_0 k^2} \rho(k)
\]

(2.15)

where \( \varepsilon_0 \) 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\( k_B T_e / \varepsilon c^2 \ll 1 \) (Hagfors, 1961), where \( k_B \) is the Boltzmann constant, \( T_e \) is the electron temperature, \( \varepsilon_c \) 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

\[
E = \frac{1}{2} \left[ \sum_{j=1}^{n_o V} m_i v_{i,j}^2 + \sum_{j=1}^{n_e V} m_e v_{e,j}^2 + \varepsilon_0 \int_V \|E(r)\|^2 d^3r \right].
\]

(2.16)

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

\[
E_{el} = \frac{1}{2} \varepsilon_0 \int_V \|E(r)\|^2 d^3r = \frac{V \varepsilon_0}{2} \sum_k \|E(k)\|^2
\]

\[
= \frac{V q^2}{2 \varepsilon_0} \sum_k -\frac{1}{k^2} \left[ |Z n_i(k) - n_e(k) | \right]^2
\]

(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

\[
E = \frac{1}{2} \left[ \sum_{j=1}^{n_o V} m_i v_{i,j}^2 + \sum_{j=1}^{n_e V} m_e v_{e,j}^2 - \frac{V q^2}{\varepsilon_0} \sum_k \frac{1}{k^2} \left[ |Z n_i(k) - n_e(k) | \right]^2 \right].
\]

(2.18)

If the amount of particles is so high that \( \ell_1 \ell_2 \ell_3 \ll n_{e,0} V \) and \( n_{i,0} V \) \( (n_{e,0} \) and \( n_{i,0} \) being continuous functions), many particles contribute to each particle density sample. Individual samples are denoted \( \{ n_{1,1}, \ldots, n_{8\ell_1 \ell_2 \ell_3} \} \) 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 \( k = 2\pi(\ell_1, \ell_2, \ell_3)/L. \) The discontinuous functions \( n_i(r) \)
and \(n_e(r)\) are related to the sampled values, and to find this relation we consider wave numbers \(k_{\eta_1,\eta_2,\eta_3}\) 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 \(n_i(r)\) and \(n_e(r)\) by integration over the periodicity cube with the following weighting factor:” (coefficients from 3D Fourier transform with periodic boundary conditions).

\[
w(r - r_{m_1,m_2,m_3}) = \prod_{j=1}^{3} \frac{\sin \left[ \frac{2\ell_j + 1}{L} \pi \left( x_j - \frac{m_jL}{2\ell_j + 1} \right) \right]}{(2\ell_j + 1) \sin \left[ \frac{\pi}{L} \left( x_j - \frac{m_jL}{2\ell_j + 1} \right) \right]}.
\]

So, \(\eta_j\) is the position indices in the frequency/Fourier transformed domain. Let us define

\[
r_{m_1,m_2,m_3} = L \left[ \frac{m_1}{2\ell_1 + 1}, \frac{m_2}{2\ell_2 + 1}, \frac{m_3}{2\ell_3 + 1} \right]
\]

(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 \(\bar{n}_i(r_{m_1,m_2,m_3})\) (for ions, similar for electrons). By making use of the Fourier transform in its discrete form, we get

\[
\bar{n}_i(r_{m_1,m_2,m_3}) = \frac{V}{\prod_{j=1}^{3} (2\ell_j + 1)} \sum_{\eta_1 = -\ell_1}^{\ell_1} \sum_{\eta_2 = -\ell_2}^{\ell_2} \sum_{\eta_3 = -\ell_3}^{\ell_3} n_i(k_{\eta_1,\eta_2,\eta_3}) \exp\left\{-i k_{\eta_1,\eta_2,\eta_3} \cdot r_{m_1,m_2,m_3} \right\}
\]

(2.21)

from which we obtain

\[
\sum_{m_1} \sum_{m_2} \sum_{m_3} \bar{n}_i^2(r_{m_1,m_2,m_3}) = \frac{V^2}{8\ell_1\ell_2\ell_3} \sum_{\eta_1} \sum_{\eta_2} \sum_{\eta_3} \left| n_i(k_{\eta_1,\eta_2,\eta_3}) \right|^2
\]

(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

\[
\exp \left\{ \left\{ -E(\bar{n}_{e,\xi}, \bar{n}_{i,\xi}) \right\} / k_B T \right\}
\]

(2.23)
When going from sampled densities to their Fourier components we see in eq. (2.22), being the energy of a microstate. The number of permutations of these microstates are given by 

\[
(n_i,0) \cdot (n,0)! / \prod_{j=1}^{N} \Pi \bar{n}_j! \prod \bar{n}_j!
\]

This has sampled densities in the exponent on the form as seen in eq. (2.22).

where \( \zeta \) and \( \xi \) 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 \((n_i,0)\cdot(n,0)! / \prod_{j=1}^{N} \Pi \bar{n}_j! \prod \bar{n}_j!\)

\[(2.24)\]

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

\[
\varphi(\bar{n}_e, \bar{n}_i) \sim \exp[-E/k_B T] \exp \left[-\frac{8{\ell}_1{\ell}_2{\ell}_3}{2n_e0V} \sum (\bar{n}_e^2 + \bar{n}_i^2) \right].
\]

\[(2.25)\]

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 \( n_i(k) \) contains both real and imaginary terms. Therefore, when changing the variables, only the directions of the wave vector \( k \) pointing into one hemisphere are accounted for if we want to use \( n_{e,R}, n_{e,I}, n_{i,R} \) and \( n_{i,I} \) (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)

\[
\varphi(n_{i,R}, n_{e,R}, n_{i,I}, n_{e,I}) \\

where \( n_{e,R+3} := n_{e,R} + n_{e,I} \) and \( n_{e,R+3} := n_{e,R} + n_{e,I} \), and where \( n_i = n_i(k_{\eta_1,\eta_2,\eta_3}), n_e = n_e(k_{\eta_1,\eta_2,\eta_3}) \). Also, \((2X_p^2)^{-1} = (\lambda_D ||k_{\eta_1,\eta_2,\eta_3}||^2 \) with \( \lambda_D^2 = \varepsilon_0 k_B T_e / n_{e,0} q_e^2 \) being the Debye length squared and where we have defined \( X_p^2 := \frac{m_e \omega_p^2}{2k_B T_e} \). 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 \( n_i(k) \) and \( n_e(k) \) for one particular wave number is written down separately as (Hagfors,
\(\varphi(n_{1,\Re}, n_{e,\Re})\)  

\[
\sim \exp \left[ -\frac{V}{n_{e,0}} \left\{ n_{1,\Re}^2 Z (1 + 2X_p^2) + n_{e,\Re}^2 (1 + 2X_p^2) - 4ZX_p^2 n_{1,\Re} n_{e,\Re} \right\} \right].
\]  

(2.27)

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

\[
\langle n_{e,\Re}^2 \rangle = \langle n_{e,\Im}^2 \rangle = \frac{n_{e,0}}{2V} \frac{1 + 2X_p^2}{1 + 2X_p^2 (1 + Z)}
\]

(2.28a)

\[
\langle n_{1,\Re}^2 \rangle = \langle n_{1,\Im}^2 \rangle = \frac{n_{e,0}}{2VZ} \frac{1 + 2X_p^2}{1 + 2X_p^2 (1 + Z)}
\]

(2.28b)

\[
\langle n_{e,\Re} n_{1,\Re} \rangle = \langle n_{e,\Im} n_{1,\Im} \rangle = \frac{n_{e,0}}{2V} \frac{2X_p^2}{1 + 2X_p^2 (1 + Z)}
\]

(2.28c)

\[
\langle n_{e,\Re} n_{e,\Im} \rangle = \langle n_{e,\Im} n_{e,\Re} \rangle = 0.
\]

(2.28d)

Further, it can be shown that

\[
\langle |n_e(k)|^2 \rangle = \langle n_{e,\Re}^2 \rangle + \langle n_{e,\Im}^2 \rangle = \frac{n_{e,0}}{V} \frac{1 + 2X_p^2}{1 + 2X_p^2 (1 + Z)}
\]

(2.28a)

\[
\langle |n_1(k)|^2 \rangle = \langle n_{1,\Re}^2 \rangle + \langle n_{1,\Im}^2 \rangle = \frac{n_{e,0}}{VZ} \frac{1 + 2X_p^2}{1 + 2X_p^2 (1 + Z)}
\]

(2.28b)

and since \(\|k\|\) is related to \(X_p\) through \((2X_p^2)^{-1} = (\lambda_D \|k_{\eta,\eta,\eta}\|)^2\) it can be shown that, for \(Z = 1\),

\[
\lim_{\|k\|\to 0} \langle |n_e(k)|^2 \rangle = \frac{n_{e,0}}{2V}
\]

(2.30)

\[
\lim_{\|k\|\to \infty} \langle |n_e(k)|^2 \rangle = \frac{n_{e,0}}{V}.
\]

(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

\[
\partial_t f + v \cdot \partial_r f + \mu_a \left[ E + v \times B \right] \cdot \partial_\alpha f = \left( \frac{\delta f}{\delta \tau} \right)_{\text{coll}}
\]

(2.32)
where \( f = f(r, \nu, t) \) and \( E \) and \( B \) are functions of space and time. \( B \) is the magnetic field vector, \( \alpha = e, i \) meaning the variables with subscript \( \alpha \) are for electrons or ions, and where \( \mu_e := -q/m_e \) for electrons and \( \mu_i := Zq/m_i \) 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

\[
5 = 5_0(\nu)[1 + 5_1]
\]

where \( 5_1 \ll 1 \). Using the spatial Fourier transform yields

\[
f_1(r, \nu, t) = \sum_k f_1(k, \nu, t) \exp[-ik \cdot r]
\]

(2.33)

and from the Laplace transform we have

\[
f_1(k, \nu, s) = \int_0^{\infty} f_1(k, \nu, t) \exp[-st]dt
\]

(2.34)

which yields for the linearized Boltzmann equation

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

(2.35)

where \( s' = s + \nu \) and \( \nu \) is a collision frequency. In the succeeding the prime is omitted by letting \( s' \rightarrow s \). In eq. (2.35), \( f_1 = f_1(k, \nu, s), f^{(1)} = f^{(1)}(k, \nu) = f_1(k, \nu, t = 0), f_0 = f_0(\nu), E = E(k, s) \) and \( B = B(k, s) \). Figure 2.1 present a cylindrical coordinate system with \( B \) parallel to the third axis, along \( u, \theta \) give the angle away from parallel to \( B \) and \( \phi \) is the angle away from the first axis in the plane perpendicular to \( B \). Using these coordinates the homogenous part
of eq. (2.35) is given as (Hagfors, 1961)

\[ C_0(w, u) = \exp \left\{ \frac{1}{\mu_B} [(s - ik_u \cos \theta) \varphi - ik_w \sin \theta \sin \varphi] \right\}. \tag{2.36} \]

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

\[ C(w, u, \varphi) = \frac{1}{\mu_B} \int_{\text{fixed limit}}^{\varphi} \exp \left\{ \frac{1}{\mu_B} [(s - ik_u \cos \theta) \varphi - ik_w \sin \theta \sin \varphi'] \right\} \times \left[ \frac{\mu_B}{f_{\alpha,0}} \partial_\varphi (f_{\alpha,0}) E - f_{\alpha}^{(1)}(k, \nu') \right] d\varphi'. \tag{2.37} \]

The solution of eq. (2.35), where \( f_\alpha(k, \nu, s) = f_\alpha(w, u, \varphi, s) \), then become

\[ f_{\alpha,1}(k, \nu, s) = \frac{1}{\mu_B} \int_{-\infty}^{\varphi} G_\alpha(\varphi, \varphi') \left\{ f_{\alpha}^{(1)}(k, \nu') \mp \frac{i2X_p^2}{f_{\alpha}(0)} k \cdot \nu' [Zn_i(k, s) - n_e(k, s)] \right\} d\varphi' \tag{2.38} \]

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

\[ G_\alpha = \exp \left[ \mp \int_{\varphi'}^{\varphi} \frac{s + ik \cdot \nu}{\Omega_\alpha} \, d\varphi \right] = \exp \left[ \mp \frac{s + ik_u \cos \theta}{\Omega_\alpha} (\varphi - \varphi') \pm \frac{ik_w \sin \theta}{\Omega_\alpha} (\sin \varphi - \sin \varphi') \right]. \tag{2.39} \]

\( \Omega_\alpha = \mu_B B \) is the gyrofrequency, where \( \mu_B \) give the charge to mass ratio.

Integrating over velocity space yields the spatial densities:

\[ n_\alpha(k, s) = \int f_{\alpha,0}(\nu) f_{\alpha,1}(k, \nu, s) d\nu \tag{2.40} \]

which gives us (Hagfors, 1961)

\[ n(k, s) = Y_e(k, s) - \frac{i}{n_0} 2X_p^2 R_e(k, s) \{ ZN(k, s) - n(k, s) \} \tag{2.41a} \]
\[ N(k, s) = Y_i(k, s) + \frac{i}{n_0} 2X_p^2 R_i(k, s) \{ ZN(k, s) - n(k, s) \} \tag{2.41b} \]

where the expressions

\[ Y_\alpha(k, s) = -\int_{-\infty}^{\varphi} G_\alpha(\varphi, \varphi') f_{\alpha,0}(\nu) f_{\alpha,1}(k, \nu') d\nu d\varphi' \tag{2.42} \]
\[ R_\alpha(k, s) = - \int \int \phi \to \infty k \nu G_\alpha(\phi, \phi') f_{\nu(0)}(\nu) d\nu d\phi' \]  \hspace{0.5cm} (2.43)

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

\[ n_e(k, s) = \frac{Y_e(k, s) \left( 1 - \frac{i}{n_{e0}} 2Z^2X_p^2R_i(k, s) \right) - Y_i(k, s) \frac{i}{n_{e0}} 2Z^2X_p^2R_e(k, s)}{1 - \frac{i}{n_{e0}} 2Z^2X_p^2R_e(k, s) + Z^2R_i(k, s)} \] \hspace{0.5cm} (2.44a)

\[ n_i(k, s) = \frac{Y_i(k, s) \left( 1 - \frac{i}{n_{i0}} 2Z^2X_p^2R_e(k, s) \right) - Y_e(k, s) \frac{i}{n_{i0}} 2Z^2X_p^2R_e(k, s)}{1 - \frac{i}{n_{i0}} 2Z^2X_p^2R_e(k, s) + Z^2R_i(k, s)} \] \hspace{0.5cm} (2.44b)

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

\[ n_\alpha(k, t) = \frac{1}{i2\pi \to \infty} \int_{-it+y}^{it+y} n_\alpha(k, s) \exp[st] ds \] \hspace{0.5cm} (2.45)

where \( y := \Re \{s\} \) is greater than the real part of all singularities of \( n_\alpha(k, s) \). 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:

\[ \langle n_\alpha^*(k, 0)n_\alpha(k, t) \rangle = \frac{1}{i2\pi \to \infty} \int_{-it+y}^{it+y} \langle n_\alpha^*(k, 0)n_\alpha(k, s) \rangle \exp[st] ds \] \hspace{0.5cm} (2.46)

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

\[ \langle n_\alpha^*(k, 0)n_\alpha(k, t) \rangle = 2\Re \{ \langle n_\alpha^*(k, 0)n_\alpha(k, t) \rangle \} \] \hspace{0.5cm} (2.47)

This, along with the Wiener-Khinchine theorem, give

\[ \langle |n_\alpha(k, \omega)|^2 \rangle = \frac{1}{\pi} \lim_{\gamma \to 0} \Re \{ \langle n_\alpha^*(k, 0)n_\alpha(k, s) \rangle \} \] \hspace{0.5cm} (2.48)

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

\[ \langle n_e^*(k, 0)n_e(k, s) \rangle = \frac{\langle n_e^*(k, 0)Y_e(k, s) \rangle \left( 1 - \frac{i}{n_{e0}} 2Z^2X_p^2R_i(k, s) \right)}{1 - \frac{i}{n_{e0}} 2Z^2X_p^2R_e(k, s) + Z^2R_i(k, s)} \] \hspace{0.5cm} (2.49)
The expressions \( \langle n_e^*(k,0)Y_e \rangle \) and \( \langle n_e^*(k,0)Y_i \rangle \) contain terms \( \langle n_e^*(k,0)f_e^{(1)}(k,v) \rangle \) and \( \langle n_e^*(k,0)f_i^{(1)}(k,v) \rangle \). 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

\[
\langle n_e^*(k,0)f_e^{(1)}(k,v) \rangle = \frac{1}{n_{e,0}} \langle |n_e(k)|^2 \rangle = \frac{1}{V} \left( \frac{1 + 2ZX_p^2}{1 + 2X_p^2(1 + Z)} \right)
\]  

(2.50a)

\[
\langle n_e^*(k,0)f_i^{(1)}(k,v) \rangle = \frac{Z}{n_{e,0}} \langle n_e^*(k)n_i(k) \rangle = \frac{1}{V} \left( \frac{2ZX_p^2}{1 + 2X_p^2(1 + Z)} \right).
\]  

(2.50b)

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

\[
R_e(k,s) = i n_0 \left[ 1 - \frac{s}{\Omega_e} g_e \left( k, \frac{s}{\Omega_e} \right) \right] = i n_0 f_e \left( k, \frac{s}{\Omega_e} \right)
\]  

(2.51a)

\[
R_i(k,s) = i \frac{n_0}{Z} \left[ 1 - \frac{s}{\Omega_i} g_i \left( k, \frac{s}{\Omega_i} \right) \right] = i \frac{n_0}{Z} f_i \left( k, \frac{s}{\Omega_i} \right).
\]  

(2.51b)

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

\[
\langle n_e^*(k)Y_e \rangle = \frac{1}{\Omega_e V} \left( \frac{1 + 2ZX_p^2}{V} \right) \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} G_e(\varphi,\varphi') f_e(\nu) d\nu d\varphi'
\]  

(2.52a)

\[
= \frac{n_{e,0}}{\Omega_e V} \left( \frac{1 + 2ZX_p^2}{1 + 2X_p^2(1 + Z)} \right) g_e \left( k, \frac{s}{\Omega_e} \right)
\]  

(2.52b)

where \( g_a(k,s/\Omega_a) \) is a Gordeyev integral, given as

\[
g_a \left( k, \frac{s}{\Omega_a} \right) = -\int_0^\infty \exp \left\{ -\left( \frac{s}{\Omega_a} \right)y \right\}
\]  

\[
- \left[ \sin^2 \theta (1 - \cos y) + \frac{1}{2} y^2 \cos^2 \theta \right] \frac{k_B T_a k^2}{m_a \Omega_a^2} \right\} dy,
\]  

(2.53)

where the general form of a Gordeyev integral is given as

\[
g(\omega) = \int_0^\infty I(y,\omega) \exp[\tau \omega y] dy,
\]  

(2.54)
where \( \tau \) 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

\[
\langle |n_e(k, \omega)|^2 \rangle = \frac{n_{e,0}}{\pi V_\omega} \frac{\Im\{-F_e\}|1 + 2X_p^2 F_e|^2 + 4ZX_e^4 \Im\{-F_i\}|F_i|^2}{|1 + 2X_e^2 (F_e + ZF_i)|^2} \tag{2.55a}
\]

\[
\langle |n_i(k, \omega)|^2 \rangle = \frac{n_{e,0}}{\pi ZV_\omega} \frac{\Im\{-F_i\}|1 + 2X^2 F_e|^2 + 4ZX^4 \Im\{-F_e\}|F_e|^2}{|1 + 2X^2 (F_e + ZF_i)|^2} \tag{2.55b}
\]

where for the functions \( F_e \) and \( F_i \) we have

\[
F_e(k, \omega) = 1 - \left( i \frac{X(\omega)}{X_e} + \Lambda_e \right) \int_0^\infty \exp \left\{ -iy \frac{X(\omega)}{X_e} - y\Lambda_e \right\} \frac{1}{2X^2_e} \left[ \sin^2 \theta (1 - \cos y) + \frac{1}{2} y^2 \cos^2 \theta \right] \, dy \tag{2.56a}
\]

\[
F_i(k, \omega) = 1 - \left( i \frac{\omega^2 X(\omega)}{ZX_e} + \Lambda_i \right) \int_0^\infty \exp \left\{ -iy \frac{\omega^2 X(\omega)}{ZX_e} - y\Lambda_i \right\} \frac{\omega^2}{2ZX^2_e} \left[ \sin^2 \theta (1 - \cos y) + \frac{1}{2} y^2 \cos^2 \theta \right] \, dy \tag{2.56b}
\]

where \( \omega := (m_i/m_e)^{1/2} \). The parameters \( X(\omega), X_e, X_p \) and \( \Lambda_\alpha \) are defined as

\[
X(\omega)^2 := \frac{m_e \omega^2}{2k_B T_e k^2} \tag{2.57a}
\]

\[
X_e^2 := \frac{m_e \Omega_e}{2k_B T_e k^2} \tag{2.57b}
\]

\[
X_p^2 := \frac{m_e \omega_p^2}{2k_B T_e k^2} = \frac{1}{2k^2 \lambda_D^2} \tag{2.57c}
\]

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

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

\[
\chi_\alpha(k, \omega) = 2X_p^2 F_\alpha(k, \omega) \tag{2.58}
\]

which in turn is related to the dielectric function through

\[
e(k, \omega) = 1 + \sum_\alpha \chi_\alpha(k, \omega) \tag{2.59}
\]
where $\alpha$ represents different particle species. Equation (2.55a) can then be written into the probably more familiar form

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

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

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

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)

\[
\begin{align*}
N_2 + h\nu(<796 \text{ Å}) &\rightarrow N_2^+ + e \quad \text{(R2.1)} \\
O_2 + h\nu(<1026 \text{ Å}) &\rightarrow O_2^+ + e \quad \text{(R2.2)} \\
O + h\nu(<911 \text{ Å}) &\rightarrow O^+ + e \quad \text{(R2.3)}
\end{align*}
\]

where $h$ is the Planck’s constant and $\nu$ 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,

\[
\begin{align*}
O_2 + h\nu(<662 \text{ Å}) &\rightarrow O^+ + O + e \quad \text{(R2.4)} \\
N_2 + h\nu(<510 \text{ Å}) &\rightarrow N^+ + N + e \quad \text{(R2.5)}
\end{align*}
\]

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

$$O_2 + h\nu(<1749 \text{ Å}) \rightarrow O(^1D) + O(^3P). \quad \text{(R2.6)}$$
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:

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

which states that at a point in the atmosphere, the intensity at wavelength \( \lambda \) is equal to \( I_{\infty}(\lambda) \) scaled with an exponential, where \( \tau \) is the optical depth. This is true for photons, but electrons do not annihilate in collisions with atoms and molecules. Instead, they scatter and lose 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 losing 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 split 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).
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 $\alpha, e$ 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

$$f_{0,M}(\nu) = \left(2\pi v_{\text{th}}^2\right)^{-3/2} \exp\left\{-\frac{\nu^2}{2v_{\text{th}}^2}\right\}$$

(3.1)

and

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

(3.2)

where $\Gamma$ is the gamma function, $\Theta^2 = v_{\text{th}}^2(2\kappa - 3)/\kappa$ is the characteristic speed (Hellberg et al., 2009), $v_{\text{th}}^2 = k_B T/m$ 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 $\kappa$ are indicative of a Maxwellian distribution and a kappa distribution, respectively. Both functions are normalized so that $\int f_{0}\nu^3 d\nu = 1$. 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 $\kappa$ 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 $\kappa$ 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.
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 $F(k, \omega)$ 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 \( \nu \) is omitted, which yields

\[
F(k, \omega) = 1 + i\omega \int_0^\infty \exp \left\{ i\omega y - \frac{k_B T k^2}{m\Omega} \left[ \sin^2 \theta (1 - \cos(y\Omega)) + \frac{1}{2} y^2 \Omega^2 \cos^2 \theta \right] \right\} dy
\]

where \( y \) and \( \omega \) have been substituted back in for \( y' \) and \( \omega' \). The above can then be written in short as

\[
F(k, \omega) = 1 + i\omega g(k, \omega),
\]

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

\[
e(k, \omega) = 1 + \sum_\alpha \frac{1}{k^2 \lambda_D^2} (1 + i\omega g(k, \omega)) = 1 + \sum_\alpha \chi(k, \omega).
\]

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, \( \omega \), 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

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

where

\[
z(k, y) = (2\kappa)^{1/2} \left[ \frac{k^2 \Theta^2 \sin^2 \theta}{\Omega^2} \left( 1 - \cos(y\Omega) \right) + \frac{1}{2} k^2 y^2 \cos^2 \theta \frac{k_B T}{m} \right]
\]

and

\[
\Theta^2 = 2 \frac{\kappa - 3/2}{\kappa} \frac{k_B T}{m}.
\]

\( K_\beta \) is the modified Bessel function of the second kind of real order \( \beta \).

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

\[
\lambda_D,\kappa = \lambda_D,\kappa \left( \frac{\kappa - 3/2}{\kappa - 1/2} \right) \left[ \frac{\varepsilon_0 k_B T}{n_0 q^2} \right]^{1/2}.
\]
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:

\[ \partial_t f_{\alpha 1} + \mathbf{v} \cdot \nabla f_{\alpha 1} + \mu_\alpha \mathbf{v} \times \mathbf{B}_0 \cdot \nabla_\mathbf{v} f_{\alpha 1} = -\mu_\alpha E_1 \cdot \nabla_\mathbf{v} f_{\alpha 0} \]  

(3.12)

where \( \mu \) is the charge to mass ratio and \( \alpha \) 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 \( \phi_1 \)

\[ \epsilon_0 k^2 \phi_1(k,s) = \sum_\alpha n_0 q \int f_1(k,v,s) d^3v = \sum_\alpha \rho_1(k,s). \]

(3.13)

The parameter \( k \) 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)

\[ f_1(k,v,s) = \frac{1}{\exp[-(2\pi/\Omega)(s + ik|v|)]} \int_{-\pi}^{\pi} \exp[P(\phi') - P(\phi)]Q(\phi') d\phi' \]

(3.14)

where

\[ P(\phi) = -\frac{1}{\Omega}[(s + ik|v|)\phi + ik \cdot v \sin \phi] \]

(3.15)

\[ Q(\phi) = -\frac{1}{\Omega} \left[f_1(k,v,t=0) + i \frac{q}{m} \phi_1(k,s)k \cdot \nabla_\mathbf{v} f_0 \right] \]

(3.16)
where \( \parallel \) and \( \perp \) refer to the parallel and perpendicular component of a vector relative to the magnetic field and \( \varphi \) is the gyro phase angle found in fig. 2.1, i.e., \( \mathbf{v} = v(\varphi) = (v_\perp \cos \varphi, v_\perp \sin \varphi, v_\parallel)^T \). Further, it is shown that \( f_1(\mathbf{k}, \mathbf{v}, s) \) can be written on the form

\[
f_1(\mathbf{k}, \mathbf{v}, s) = \int_{-\infty}^{\infty} \exp[P(\varphi') - P(\varphi)] Q(\varphi') d\varphi'
\]

(3.17)

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

\[
\rho_1(\mathbf{k}, s) = \psi(\mathbf{k}, s) + \chi(\mathbf{k}, s) \phi_1(\mathbf{k}, s)
\]

(3.18)

where

\[
\psi(\mathbf{k}, s) = n_0 q \int_{-\infty}^{0} \int f_1(\mathbf{k}, \mathbf{v}, t = 0) \exp[\mathbf{s} \cdot \mathbf{v}] d^3\mathbf{v} dy
\]

(3.19)

\[
\chi(\mathbf{k}, s) = i \frac{n_0 q^2}{m} \int_{-\infty}^{0} \int (\mathbf{\mathbf{p}}(y) \cdot \nabla f_0) \exp[\mathbf{s} \cdot \mathbf{v}] d^3\mathbf{v} dy,
\]

(3.20)

and where

\[
\mathbf{p}(y) = \left( \frac{k_\parallel}{\Omega} \sin(\Omega y), \frac{\mathbf{k}_\perp}{\Omega} [1 - \cos(\Omega y)], k_\parallel \right)^T
\]

(3.21)

\[
\mathbf{p}'(y) = \left( k_\perp \cos(\Omega y), k_\perp \sin(\Omega y), k_\parallel \right)^T.
\]

(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 \( \mathbf{v} \) yields for eq. (3.20) (Mace, 2003)

\[
\chi(\mathbf{k}, s) = \frac{n_0 q^2}{m} \int_{-\infty}^{0} \int \mathbf{p}(y) \cdot \mathbf{p}'(y) \exp[\mathbf{s} \cdot \mathbf{v}] f_0(\mathbf{v}) d^3\mathbf{v} dy.
\]

(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)

\[
\chi(\mathbf{k}, s) = 4\pi \frac{q^2}{m} \int_{-\infty}^{0} \int \exp[\mathbf{s} \cdot \mathbf{p}(y)] p'(y) \int_{0}^{\infty} \mathbf{v} \sin[\mathbf{s} \cdot \mathbf{v}] f_0(\mathbf{v}) d\mathbf{v} dy
\]

(3.24)

where \( p'(y) = dp(y)/dy = d[\mathbf{p}(y) \cdot \mathbf{p}(y)]^{1/2}/dy \) and \( n_0 q^2/m = v_0^2 e_0/\lambda_D^2 \) was used to rewrite the fraction in eq. (3.23). With the relations in eqs. (2.58) and (2.59) the dielectric function become

\[
\epsilon(\mathbf{k}, \omega) = 1 + \sum_{\alpha} \chi(\mathbf{k}, \omega),
\]

(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:

\[
I(y) = A(\kappa \theta^2)^{\kappa+1} \int_0^\infty \frac{v \sin[p(y)v]}{(\kappa \theta^2 + v^2)^{\kappa+1}} dv
\]  

(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

\[
\chi(k, s) = -\frac{\epsilon_0}{\lambda_{D,M}^2} \left( \frac{\kappa - \frac{3}{2}}{\kappa - \frac{1}{2}} \right) \left[ 1 - s \int_{-\infty}^0 \frac{\exp[\alpha y]z^{\kappa+1/2}K_{\kappa+1/2}(z)dy}{2^{\kappa-1/2}\Gamma(\kappa + \frac{1}{2})} \right].
\]  

(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

\[
\frac{\lambda_{D,K}^2}{\lambda_{D,M}^2} = \frac{\kappa - \frac{3}{2}}{\kappa - \frac{1}{2}}
\]  

(3.28)

any general Debye length can be found through

\[
\frac{\lambda_{D,S}^2}{\lambda_{D,M}^2} = \frac{\int_{-\infty}^0 \int_0^\infty v \sin[p(y)v] f_{0,M} dv dy}{\int_{-\infty}^\infty \int_0^\infty v \sin[p(y)v] f_{0,S} dv dy}
\]  

(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

\[ \chi(k, s) = -4\pi \frac{n_0 q^2}{m} \left[ \int_0^\infty f_0 dv - \int_0^\infty f_0 \int_{-\infty}^0 s \exp[sv] \cos(pv)dv \right], \quad (3.30) \]

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

\[
\begin{align*}
    \int_{-\infty}^{0} \exp[sv]p' \sin(pv)dv &= \int_{-\infty}^{0} \exp[sv] \left( -\frac{1}{v} \cos(pv) \right)' dv \\
    &= -\exp[sv] \frac{1}{v} \cos(pv) \bigg|_{-\infty}^{0} + \int_{-\infty}^{0} s \exp[sv] \frac{1}{v} \cos(pv)dv \\
    &= -\frac{1}{v} + \frac{1}{v} \int_{-\infty}^{0} \exp[sv] \cos(pv)dv
\end{align*}
\]

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

Mace (2003) argues that the form of eq. (3.30) is useful because one can factor out the term \( \int_0^\infty f_0(v)dv \). On this form you are more likely to find analytical solutions to expressions (e.g. the integral \( \int_0^\infty f_0(v)dv \)) 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

\[ f_s^{(\alpha)}(v_\parallel, v_\perp) = A_s^{(\sigma_s)} \left( 1 + \frac{v_\parallel^2}{\kappa_s w_\parallel^2} + \frac{v_\perp^2}{\kappa_s w_\perp^2} \right)^{-\sigma_s} \]  \hspace{1cm} (3.32)

where \( A_s \) is a normalization constant and \( w_\parallel \) and \( w_\perp \) are proportional to the parallel and perpendicular thermal speeds, \( v_\parallel \) and \( v_\perp \), but also functions of \( \kappa \). \( s \) is the particle species and \( \alpha_s, \sigma_s \) and \( \kappa_s \) 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.
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 $\langle |n(k, \omega)|^2 \rangle$, which in turn is a function of the suscep-
Figure 4.1: Shape of the integrand $I(k, y) \exp[\tau \omega y]$ 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.

The susceptibility 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

$$g(k, \omega) = \int_{0}^{\infty} I(k, y) \exp[\tau \omega y] dy = \int_{0}^{y_{\text{max}}} I(k, y) \exp[\tau \omega y] dy \quad (4.1)$$

where $\tau$ is a complex number. A lot of computation can be omitted when realizing that the integrand $I(k, y) \exp[\tau \omega y]$ 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 $y_{\text{max}}$ was chosen. To further take advantage of the shape of the integrand the integral was sampled according to the formula

$$y = (y')^a \quad (4.2)$$

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
4.1 / evaluating the gordeyev integral using the simpson's rule

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

\[
\chi(k, s) = 4\pi \frac{n_0 q^2}{m} \int_{-\infty}^{0} \exp[s y] \rho'(k, y) \gamma(k, y) dy
\]  

(4.3)

where

\[
\gamma(k, y) = \int_{0}^{\infty} v \sin[p(k, y) v] f_0(v) dv
\]  

(4.4)

and it is clear that the velocity integral, \( \gamma(k, y) \), 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 \( v_{\text{max}} \). The value of \( v_{\text{max}} \) was chosen based on the available energies and subsequently velocities in the calculated electron fluxes. The maximum available energy from the calculated
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, ...
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-

Figure 4.4: Comparison between the semi-analytic implementation and the numerical implementation of the IS spectrum calculation. Here, $N_g = 8 \times 10^4$ and $N_\theta = 4 \times 10^3$. (a) show the spectra from a Maxwellian distribution. (Continues on the next page.)

personal communication). The suprathermal distribution covered the interval from $E = 1$ eV to $E = 110$ eV and was interpolated to cover energies down to $E = 0$ eV. Interpolation was carried out using the `interp` algorithm provided by `numpy` with the default setting, which give the value at $E = 1$ eV to all samples in the region $E = [0, 1)$ 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 $f_{0,M} + f_{0,S}$, where the Maxwellian distribution for the thermal electrons is shown by the orange “dash, dot” line labelled $f_{0,M}$, and the suprathermal distribution is shown by the green “dash, dot, dot” line labelled $f_{0,S}$.
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 \( N_y = 8 \times 10^4 \) and \( N_v = 4 \times 10^4 \) samples in the velocity integral and \( N_y = 4 \times 10^5 \) samples in the Gordeyev 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 \( N_v = 4 \times 10^5 \)
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 \( N_o = 4 \times 10^5 \).

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 \( N_o = 4 \times 10^4 \).

did not do much of a difference. Figure 4.5 show the same comparison as fig. 4.4, but with \( N_o = 4 \times 10^5 \) samples in the velocity integral instead of \( N_o = 4 \times 10^4 \) 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 \( N_o = 4 \times 10^4 \) samples and that the reason for the poor numerical precision in fig. 4.4 was not caused by the value of \( N_o \).

In fig. 4.6, the sampling of the velocity was reset down to \( N_o = 4 \times 10^4 \), 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 \( N_o = 4 \times 10^5 \), shown in
we have seen, increasing the sampling above with the Maxwellian distribution lies in the decimal precision. Figure 3.1 show was, nevertheless, still poor at frequencies larger than $\times 10$.

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 $N_y > 4 \times 10^4$, 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 $E = 10$ eV, the magnitude of the Maxwellian is about $1 \times 10^{-40}$ times any of the kappa distributions presented. Also, the kappa distribution for kappa index $\kappa < 8$ never reach a magnitude of less than $1 \times 10^{-34}$ on the whole energy range up to $E = 110$ eV.

To see if the decimal precision is the issue, the upper limit, $v_{\text{max}}$, was lowered to $v_{\text{max}} = 2 \times 10^6$ ms$^{-1}$ from $v_{\text{max}} = 6 \times 10^6$ ms$^{-1}$, where $v = 2 \times 10^6$ ms$^{-1}$ give $E \approx 11.4$ eV. This was done to force the magnitude of the distribution functions at velocities higher than $v_{\text{max}}$ 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 $N_y = 4 \times 10^4$ do not provide a significant improvement.
Figure 4.8: Comparison between the semi-analytical implementation and the numerical implementation of the $I_S$ spectrum calculation for low upper boundary in the velocity integral. Here, $N_y = 8 \times 10^4$, $N_v = 4 \times 10^4$ and $v_{\text{max}} = 2 \times 10^6 \text{ ms}^{-1}$. 

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 $v_{\text{max}} = 6 \times 10^5 \text{ ms}^{-1}$ 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 $v_{\text{max}} = 2 \times 10^6 \text{ ms}^{-1}$, 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
velocity space should be equal to one:

\[
\int f_0 d^3v = 1. \tag{4.6}
\]

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, $N_u$, was equal in all numerical tests, i.e., $N = N_f = N_u$. 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 $1 \times 10^5$ Hz. Because of these numerical errors, the chirp $z$-transform was put aside.
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
results from model calculations of IS spectra

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 $\pm 0.6$ MHz referred

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Unit</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_r$</td>
<td>Hz</td>
<td>$430 \times 10^6$</td>
</tr>
<tr>
<td>$n_0$</td>
<td>$m^{-3}$</td>
<td>$2.0 \times 10^{10}$</td>
</tr>
<tr>
<td>$B$</td>
<td>[T]</td>
<td>$3.5 \times 10^{-5}$</td>
</tr>
<tr>
<td>$m_i$</td>
<td>[amu]</td>
<td>29</td>
</tr>
<tr>
<td>$v_e$</td>
<td>[Hz]</td>
<td>0</td>
</tr>
<tr>
<td>$v_i$</td>
<td>[Hz]</td>
<td>0</td>
</tr>
<tr>
<td>$T_e$</td>
<td>[K]</td>
<td>200</td>
</tr>
<tr>
<td>$T_i$</td>
<td>[K]</td>
<td>200</td>
</tr>
<tr>
<td>$\theta$</td>
<td>[$^\circ$]</td>
<td>135.0</td>
</tr>
</tbody>
</table>
spectra from maxwellian and kappa distributions

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 $\pm 1.5$ 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 $\chi_s$ approach zero. Similarly, the ion line appear where $\chi_i$ 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, $T_e = T_i = 200$ 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

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

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
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, $\nu$ is collision frequency, $T$ is temperature and $\theta$ is the angle between the radar beam and the magnetic field line.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Unit</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_r$</td>
<td>[Hz]</td>
<td>$933 \times 10^6$</td>
</tr>
<tr>
<td>$n_0$</td>
<td>[m$^{-3}$]</td>
<td>$2.0 \times 10^{11}$</td>
</tr>
<tr>
<td>$B$</td>
<td>[T]</td>
<td>$5 \times 10^{-5}$</td>
</tr>
<tr>
<td>$m_i$</td>
<td>[amu]</td>
<td>16</td>
</tr>
<tr>
<td>$\nu_e$</td>
<td>[Hz]</td>
<td>0</td>
</tr>
<tr>
<td>$\nu_i$</td>
<td>[Hz]</td>
<td>0</td>
</tr>
<tr>
<td>$T_e$</td>
<td>[K]</td>
<td>5000</td>
</tr>
<tr>
<td>$T_i$</td>
<td>[K]</td>
<td>2000</td>
</tr>
<tr>
<td>$\theta$</td>
<td>[$^\circ$]</td>
<td>180.0</td>
</tr>
</tbody>
</table>
results from model calculations of is spectra

\( e = 10000 \text{K} \)

\( e = 9000 \text{K} \)

\( e = 8000 \text{K} \)

\( e = 7000 \text{K} \)

\( e = 6000 \text{K} \)

\( e = 5000 \text{K} \)

\( e = 4000 \text{K} \)

\( e = 3000 \text{K} \)

\( e = 2000 \text{K} \)

**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
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

$$\Delta f_R = f_{R+} - f_{R-},$$

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 1S 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.

\[ k_\pm = \frac{2\pi}{c} [f_i + (f_i \pm f_R)] \]  

(5.3)

assuming \( f_R/f_i \ll 1 \) and where \( \pm \) refer to the up- and downshifted frequencies. For the Arecibo radar the asymmetry parameter \( (f_{R+} + f_{R-}) \) 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, \( E = 14 \text{ eV} \) to \( E = 27 \text{ eV} \). 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:

\[ E(\theta) = D\cos^{1.94} \]  

(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:

\[ f_{R}(\theta) = A\cos^{0.97} \]  

(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 \( \mu \)s with bits of 2 \( \mu \)s.
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 \(18^\circ20'39''\ N, 66^\circ45'10''\ W\), 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 \(15^\circ\), and the antenna was rotated \(720^\circ\) in azimuth during the experiment, hence the aspect angle variation between \(\theta = 120^\circ\) and \(\theta = 150^\circ\) 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

\[
E = \frac{1}{2} m_e \left( \frac{f_R \lambda}{\cos \theta/2} \right)^2.
\]

(5.6)

Plasma wave phase velocity is defined as \(v_\phi = f_R \lambda/2\) (Yngvesson and Perkins,
1968; Djuth et al., 2018), where $f_R$ is the resonance frequency of the plasma wave measured by the radar, $\lambda$ 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 $E = m_e \sigma_n^2 / 2$. 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)

$$T_p(\nu_0) = T_e - \frac{f_M(\nu_0) + f_S(\nu_0) + \chi_{\text{coll}}}{f_M(\nu_0) - k_B T_e \frac{d}{dE} f_S(\nu_0) + \chi_{\text{coll}}}$$

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 $\chi_{\text{coll}}$ 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 $-k_B T_e \frac{d}{dE} f_S(\nu_0)$ 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 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 $1S$ 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 $1S$ spectrum from the program yields reasonable results. The level of precision was the same as used in fig. 4.4, i.e., $Nv = 4 \times 10^4$ and $Ny = 8 \times 10^4$, since the spectrum was calculated for frequency $f < 9.5$ 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, $\theta$, 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, $\theta$, and electron number density, $n_e$. Table 5.3 give the plasma parameters used in the computation of the 1S spectrum, while the green shaded regions represent plasma peak frequencies that map to $E = (15.58, 18.42) \text{ eV}$ or $E = (22.47, 23.75) \text{ eV}$, 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 $\theta = 135^\circ$ 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, $\nu$ 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.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Unit</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_r$</td>
<td>[Hz]</td>
<td>$430 \times 10^6$</td>
</tr>
<tr>
<td>$B$</td>
<td>[T]</td>
<td>$35 , 000 \times 10^{-9}$</td>
</tr>
<tr>
<td>$m_i$</td>
<td>[amu]</td>
<td>16</td>
</tr>
<tr>
<td>$\nu_e$</td>
<td>[Hz]</td>
<td>100</td>
</tr>
<tr>
<td>$\nu_i$</td>
<td>[Hz]</td>
<td>100</td>
</tr>
<tr>
<td>$T_e$</td>
<td>[K]</td>
<td>2000</td>
</tr>
<tr>
<td>$T_i$</td>
<td>[K]</td>
<td>1500</td>
</tr>
<tr>
<td>Height</td>
<td>[km]</td>
<td>599</td>
</tr>
<tr>
<td>ToD</td>
<td>[UT]</td>
<td>12:00</td>
</tr>
</tbody>
</table>
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).

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.

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, \( \nu_e \) 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.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Unit</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f_r )</td>
<td>[Hz]</td>
<td>( 430 \times 10^6 )</td>
</tr>
<tr>
<td>( B )</td>
<td>[T]</td>
<td>( 35000 \times 10^{-9} )</td>
</tr>
<tr>
<td>( m_i )</td>
<td>[amu]</td>
<td>16</td>
</tr>
<tr>
<td>( \nu_e )</td>
<td>[Hz]</td>
<td>100</td>
</tr>
<tr>
<td>( \nu_i )</td>
<td>[Hz]</td>
<td>100</td>
</tr>
<tr>
<td>( T_e )</td>
<td>[K]</td>
<td>2000</td>
</tr>
<tr>
<td>( T_i )</td>
<td>[K]</td>
<td>1500</td>
</tr>
<tr>
<td>Height</td>
<td>[km]</td>
<td>300</td>
</tr>
<tr>
<td>ToD</td>
<td>[UT]</td>
<td>12:00</td>
</tr>
</tbody>
</table>
results from model calculations of IS spectra

Electron number density, \( n_e \), Table 5.4 gives 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
5.3 / plasma line power structures at Arecibo Observatory

![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.](image)

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

\[
E = \frac{1}{2} m_e \left( \frac{f_R \lambda}{\cos \theta} \right)^2 = \frac{1}{2} m_e \left( \frac{A \cos (\theta)^{0.97} \lambda}{\cos \theta} \right)^2 \tag{5.8}
\]
or, without the scaling for the angle to the magnetic field:

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

(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.
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 $N_g = 8 \times 10^4$ was found to be sufficient up to about 9.5 MHz in the IS spectrum. A sample size in the Gordeyev integral of $N_g = 8 \times 10^5$ 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.
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.
"""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)
        # self.r = reproduce.PlotTemperature(self.plot)
def create_data(self):
    """Create IS spectra.

    The spectra should be appended to the `self.data` list, giving a list of spectra that are themselves `np.ndarrays`, or into a list of such lists as the aforementioned.

    A list of spectra can be plotted in `plot_normal`, while a list of lists can be plotted by `plot_ridge`. When using `plot_ridge`, it is assumed that all the lists in the outer list is of equal length.

    The list `self.ridge_txt` should be the same length as the length of the outer list when plotting with `plot_ridge`, since this text will go on the left of every ridge. The list `self.legend_txt` should be the same length as the length of the inner lists, and will give the legend for the spectra given in the inner lists.

    Notes:
    ::
        Possible items in the sys_set dictionary include:
        K_RADAR -- Radar wavenumber
            (= -4pi(radar frequency)/(speed of light)) [m^(-1)]
        B -- Magnetic field strength [T]
        MI -- Ion mass in atomic mass units [u]
        NE -- Electron number density [m^(-3)]
        NU_E -- Electron collision frequency [Hz]
        NU_I -- Ion collision frequency [Hz]
        T_E -- Electron temperature [K]
        T_I -- Ion temperature [K]
        T_ES -- Temperature of suprathermal electrons in the gauss_shell VDF [K]
        THETA -- Aspect angle [1]
        Z -- Height of real data [100, 599] [km]
        mat_file -- Important when using real data and decides the time of day
        pitch_angle -- list of integers that determine which slices of the pitch angles are used. 'all' uses all

    Examples:
    ::
        TEMPS = [2000, 5000]
        methods = ['maxwell', 'kappa']
        params = {'kappa': 3, 'vdf': 'kappa', 'area': False}
        for T in TEMPS:
            ridge = []
            sys_set['T_E'] = T
            self.ridge_txt.append(f'T_e = {T} K')
            for m in methods:
                self.f, s, meta_data = isr.isr_spectrum(m, sys_set, **params)
```python
self.meta_data.append(meta_data)
ridge.append(s)
self.data.append(ridge)

# For a nicer legend, this is added manually
self.legend_txt.append('Maxwellian')
self.legend_txt.append('Kappa')
```

```python
def plot_data(self):
    """Plot the created data from `self.data`.

    If you want to only plot the plasma line, set
    `self.plot.plasma = True`

    `self.plot.plot_normal()` accepts a list of `np.ndarray`'s and
    `self.plot.plot_ridge()` accepts a list of lists of `np.ndarray`'s,
    i.e. a list of the type you send to `self.plot.plot_normal()`.

    Examples:
    ::
        """
        # Given the example in self.create_data()
        # self.plot.plasma = True
        self.plot.plot_normal(self.f, self.data[0], 'plot',
                              self.legend_txt)
        self.plot.plot_normal(self.f, self.data[0], 'semilogy',
                              self.legend_txt)
        self.plot.plot_ridge(self.f, self.data, 'plot', self.legend_txt,
                              self.ridge_txt)
        self.plot.plot_ridge(self.f, self.data, 'semilogy',
                              self.legend_txt, self.ridge_txt)
        """
    
    self.r.plot_it()
```

```python
def save_handle(self, mode):
    if mode == 'setUp':
        if self.plot.save in ['y', 'yes']:
            self.plot.save_it(self.f, self.data, self.legend_txt,
                              self.ridge_txt, self.meta_data)
    elif mode == 'tearDown':
        if self.plot.save in ['y', 'yes']:
```
```python
self.plot.pdffig.close()
plt.show()

def run(self):
    self.create_data()
    self.save_handle('setUp')
    self.plot_data()
    self.save_handle('tearDown')

if __name__ == '__main__':
    Simulation().run()
    # hk.HelloKitty1().run()

---

### A.2 config.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

# 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
def create_it(self, *args, from_file=False):
    if not from_file:
        self.create_from_code()
    else:
        self.create_from_file(*args)

@abstractmethod
def create_from_code(self):
    """Method that create needed data."
    """

def create_from_file(self, *args):
    """Accepts zero, one or two arguments.

    If zero arguments are given, a default path is used to look for files.
    ::
    If one argument is given, it should include
    the full path (with or without file ending).
    ::
    If two arguments are given, the first should be the path to
    the directory where the file is located, and the second
    argument must be the name of the file.
    """
    if len(args) != 0:
        if len(args) == 1:
            parts = args.split('/
            path = '/'.join(parts[:-1]) + '/
            name = parts[-1]
        elif len(args) == 2:
            path = args[0]
            name = args[1]
        else:
            path = '../figures/
            name = 'hello_kitty_2020_6_9_2--28--4.npz'
            name = name.split('.')[0]
            try:
                f = np.load(path + name + '.npz', allow_pickle=True)
            except Exception:
                sys.exit(print(f'Could not open file {path + name}.npz'))
            sorted(f)
            self.f, self.data, self.meta_data = f['frequency'],
            list(f['spectra']), list(f['meta'])
            self.legend_txt, self.ridge_txt = list(f['legend_txt']),
            list(f['ridge_txt'])
    if self.p.save in ['y', 'yes']:
        self.p.save_path = name

@abstractmethod
def plot_it(self):
    """Method that plot relevant plots."
    """
class PlotNumerical(Reproduce):
    '''Reproduce figure with a comparison between the semi-analytic
    and numerical implementation.

    In config, set
    ```
    'F_MIN': - 2e6, 'F_MAX': 9e6
    ```
    Also, using
    ```
    F_N_POINTS = 1e3
    ```
    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': 36000e-9, 'MI': 16,
               'NE': 1e12, 'NU_E': 100, 'NU_I': 100, 'T_E': 2000,
               'T_I': 1500, 'T_ES': 90000,
               'THETA': 30 * np.pi / 180, 'Z': 300,
               'mat_file': 'fe_zmuE-07.mat',
               'pitch_angle': 'all'}
    params = {'kappa': 3, 'vdf': 'maxwell', 'area': False}

    ridge = []
    self.f, s1, meta_data = isr.isr_spectrum('maxwell', sys_set, **params)
    ridge.append(s1)
    self.meta_data.append(meta_data)
    _, s2, _ = isr.isr_spectrum('a_vdf', sys_set, **params)
    ridge.append(s2)
    self.data.append(ridge)

    ridge = []
    params['vdf'] = 'kappa'
    self.f, s1, meta_data = isr.isr_spectrum('kappa', sys_set, **params)
    ridge.append(s1)
    self.meta_data.append(meta_data)
    _, s2, _ = isr.isr_spectrum('a_vdf', sys_set, **params)
    ridge.append(s2)
    self.data.append(ridge)

def plot_it(self):
    for maxwell, data in enumerate(self.data):
        self.plotter(maxwell, data)

def plotter(self, maxwell, data):
    s1 = data[0]
    s2 = data[1]
    plot = plt.semilogy
    xlim = [1e3, self.f[-1]]
```python

d = s1 - s2
rd = d / s1
plt.figure(figsize=(8, 5))
plt.subplot(3, 1, 1)
if maxwell == 0:
    plt.title('Maxwell')
else:
    plt.title('Kappa')
plot(self.f, s1, 'k', label='Semi-analytic (SA)')
plot(self.f, s2, 'r--', label='Numerical (N)')
plt.legend()
# plt.xlim(xlim)
plt.minorticks_on()
plt.grid(True, which='both', ls='-', alpha=0.4)
plt.subplot(3, 1, 2)
plt.title('Difference (SA - N)')
plot(self.f, d, 'k', label='Positive')
plot(self.f, -d, 'r', label='Negative')
plt.legend()
# plt.xlim(xlim)
plt.minorticks_on()
plt.grid(True, which='both', ls='-', alpha=0.4)
plt.subplot(3, 1, 3)
plt.title('Difference relative to semi-analytic [(SA - N) / SA]')
plot(self.f, rd, 'k', label='Positive')
plot(self.f, -rd, 'r', label='Negative')
plt.legend()
# plt.xlim(xlim)
plt.minorticks_on()
plt.grid(True, which='both', ls='-', alpha=0.4)
plt.yticks([1e-9, 1e-6, 1e-3, 1e0])
plt.tight_layout()

if self.p.save in ['y', 'yes']:
    self.p.pdffig.attach_note('numerical precision test')
plt.savefig(self.p.pdffig, bbox_inches='tight', format='pdf',
            dpi=600)
plt.savefig(str(self.p.save_path) + f'_page_{self.p.page}.pgf',
            bbox_inches='tight')
self.p.page += 1

class PlotTestDebye(Reproduce):
    
    """Reproduce figure of IS spectra using two kappa dist with and without Debye length correction."
    """  In config, set
    """  "'F_MIN': -2e6, 'F_MAX': 2e6"
    """  Also, using
    """
    F_N_POINTS = 5e5
```
```

is sufficient.

---

```python

def create_from_code(self):
    F0 = 430e6
    K_RADAR = - 2 * F0 * 2 * np.pi / const.c # Radar wavenumber
    self.legend_txt = 
    ↔ ['r'$\lambda_{\text{maxwell}}$' = \lambda_{\text{maxwell}(D)} \kappa',
    ↔ r'$\lambda_{\text{maxwell}(D)}$ = \lambda_{\text{maxwell}(D,N)}$']
    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)
    self.legend_txt = ['f_zmuE-07.mat']

    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
```
for k in kappa:
    params['kappa'] = k
    self.f, s, meta_data = isr.isr_spectrum('kappa', sys_set, **params)
    self.data.append(s)
    meta_data['version'] = 'both'
    self.meta_data.append(meta_data)

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

class PlotIonLine(Reproduce):
    # Reproduce figure with ridge plot over different temperatures.
    # In config, set
    # "F_MIN": -3e3, "F_MAX": 3e3
    # Also, using
    # "F_N_POINTS = 1e3" is sufficient.
    
    def create_from_code(self):
        F0 = 430e6
        K_RADAR = -2 * F0 * 2 * np.pi / const.c
        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, 'HI': 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)
        for k in kappa:
            params['kappa'] = k
            self.f, s, meta_data = isr.isr_spectrum('kappa', sys_set, **params)
            self.data.append(s)
            meta_data['version'] = 'both'
            self.meta_data.append(meta_data)

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

class PlotPlasmaLine(Reproduce):
    # Reproduce figure with ridge plot over different temperatures.
In config, set
```
'F_MIN': 3.5e6, 'F_MAX': 7e6
```
Also, using
```
F_N_POINTS = 1e3
```
is sufficient.

```python
def create_from_code(self):
    F0 = 933e6
    K_RADAR = -2 * F0 * 2 * np.pi / const.c
    self.legend_txt = ['Maxwellian', r'$r\$\kappa = 20$', r'$r\$\kappa = 8$']
    kappa = [20, 8, 3]
    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, 
               'THETA': 0 * 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)
    for k in kappa:
        params['kappa'] = k
        self.f, s, meta_data = isr.isr_spectrum('kappa', sys_set, **params)
        self.data.append(s)
    meta_data['version'] = 'both'
    self.meta_data.append(meta_data)

def plot_it(self):
    self.p.plot_normal(self.f, self.data, plot, self.legend_txt)
```

```python
class PlotTemperature(Reproduce):
    ""
    Reproduce figure with ridge plot over different temperatures.
    ""
    def __init__(self, p):
        super(PlotTemperature, self).__init__(p)
        self.f_list = [[]]

    def create_from_file(self, *args):
```

```
In config, set
```
```
'F_MIN': 3.5e6, 'F_MAX': 7.5e6
```
Also, using
```
F_N_POINTS = 5e3
```
is sufficient.

```python
def __init__(self, p):
    super(PlotTemperature, self).__init__(p)
    self.f_list = [[]]

def create_from_file(self, *args):
```
"""Accepts zero, one or two arguments.

If zero arguments are given, a default path is used to look for files.

::

If one argument is given, it should include the full path (with or without file ending).

::

If two arguments are given, the first should be the path to the directory where the file is located, and the second argument must be the name of the file.

"""

if len(args) != 0:
    if len(args) == 1:
        parts = args.split('/

        path = '/

        name = parts[-1]

    elif len(args) == 2:
        path = args[0]
        name = args[1]

else:
    path = './../figures/

name = 'hello_kitty_2020_6_9_2--28--4.npz'

try:
    f = np.load(path + name + '.npz', allow_pickle=True)
except Exception:
    sys.exit(print(f'Could not open file {path + name}.npz'))

sorted(f)

self.f, self.data, self.meta_data = f[f'frequency'],
                        list(f['spectra']), list(f['meta'])

self.legend_txt, self.ridge_txt = list(f['legend_txt']),
                        list(f['ridge_txt'])

for r in self.data:

    peak = int(np.argmax(r[0] == np.max(r[0])))

    self.f_list[0].append(self.f[peak])

    peak = int(np.argmax(r[1] == np.max(r[1])))

    self.f_list[1].append(self.f[peak])

    peak = int(np.argmax(r[2] == np.max(r[2])))

    self.f_list[2].append(self.f[peak])

if self.p.save in ['y', 'yes']:

    self.p.save_path = name

def create_from_code(self):
    F0 = 933e6

    K_RADAR = -2 * F0 * 2 * np.pi / const.c

    T = [2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000, 10000]

    self.ridge_txt = ['$T_{\mathrm{e}} = %d \ \mathrm{K}$' % j for j in T]

    self.legend_txt = ['Maxwellian', r'$\kappa = 20$', r'$\kappa = 3$']
```python
sys_set = {'K_RADAR': K_RADAR, 'B': 500000e-9, 'MI': 16, 'NE': 2e11,
           'NU_E': 0, 'NU_I': 0, 'T_E': 2000, 'T_I': 2000, 'T_ES': 90000,
           'THETA': 0 * np.pi / 180, 'Z': 599, 'mat_file':
           'fe_zmuE-07.mat'}
params = {'kappa': 8, 'vdf': 'real_data', 'area': False}
kappa = [20, 3]
for t in T:
    ridge = []
    sys_set['T_E'] = t
    self.f, s, meta_data = isr.isr_spectrum('maxwell', sys_set,
                                           **params)
    ridge.append(s)
    for k in kappa:
        params['kappa'] = k
        self.f, s, meta_data = isr.isr_spectrum('kappa', sys_set,
                                           **params)
        ridge.append(s)
    self.data.append(ridge)
    self.meta_data.append(meta_data)

for r in self.data:
    peak = int(np.argmax(r[0] == np.max(r[0])))
    self.f_list[0].append(self.f[peak])
    peak = int(np.argmax(r[1] == np.max(r[1])))
    self.f_list[1].append(self.f[peak])
    peak = int(np.argmax(r[2] == np.max(r[2])))
    self.f_list[2].append(self.f[peak])

def plot_it(self):
    self.p.plot_ridge(self.f, self.data, 'plot', self.legend_txt,
                      self.ridge_txt)

    T = [2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000, 10000]
    plt.figure(figsize=(6, 3))
    plt.plot(T, self.f_list[0], 'k', label='Maxwellian')
    plt.plot(T, self.f_list[1], 'k--', label=r'$\kappa = 20$')
    plt.plot(T, self.f_list[2], 'k:', label=r'$\kappa = 3$')
    plt.legend()

    if self.p.save in ['y', 'yes']:
        self.p.pdf.attach_note('freq change')
        pl.savefig(self.p.pdf, bbox_inches='tight', format='pdf',
                   dpi=600)
        pl.savefig(str(self.p.save_path) + f'_page_{self.p.page}.pgf',
                   bbox_inches='tight')
        self.p.page += 1

class PlotHKExtremes(Reproduce):
    """Reproduce figure with ridge plot over the extremes from
the Hello Kitty plot."
    In config, set
    """
```


```python
# F_MIN': 2.5e6, 'F_MAX': 9.5e6

Also, using

```

F_N_POINTS = 1e4

is sufficient.
```
class PlotHK:
    """Reproduce the Hello Kitty figures from saved data."""

def __init__(self, *args):
    """Accepts zero, one or two arguments.
    
    If zero arguments are given, a default path is used to look for files.
    ::
    If one argument is given, it should include
    the full path (with or without file ending).
    ::
    If two arguments are given, the first should be the path to
    the directory where the file is located, and the second
    argument must be the name of the file.
    """
    if len(args) != 0:
        if len(args) == 1:
            args = args[0]
            parts = args.split('/
            path = '/'.join(parts[:-1]) + '/
            self.name = parts[-1]
        elif len(args) == 2:
            path = args[0]
            self.name = args[1]
    else:
        path = '.../figures/
        # Old
        # self.name = 'hello_kitty_2020_6_9_2--28--4.npz'
        self.name = 'hello_kitty_2020_6_8_22--1--51.npz'
        # New
        # self.name = 'hello_kitty_2020_6_15_22--27--16.npz'
        # self.name = 'hello_kitty_2020_6_15_15--50--18.npz'
        self.name = self.name.split('..')
        try:
            self.file = np.load(path + self.name + '.npz')
        except Exception:
            sys.exit(print(f'Could not open file {path + self.name}'))
        self.g = self.file['power']

def shade(self):
    dots_x = []
    dots_y = []
    for i, d in enumerate(self.file['dots'][1]):
        arg = np.argmax(self.file['angle'] ==
                         self.file['angle'][int(d)])
        dots_x = np.r_[dots_x, arg[1, 0]]
        dots_y = np.r_[dots_y, np.ones(len(arg[1, 0]))]
    s = set(self.file['dots'][:, 0])
    for i in s:
        mask = np.argmax(self.file['dots'][:, 0] == i)
        xs = []
        y_min = []
        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'$\text{int}(i)$', 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 = 0.5 * const.m_e * (self.file['fr'] + 1 / (2 * np.cos(deg * np.pi / 180))**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, 0, len(self.file['density']) - 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, $n_\text{e}$')
    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.ylim([150, 135, 120])
plt.ylabel('Aspect angle')
axs = []
axs += [ax0]
axs += [ax1]
gs.update(hspace=0.05)
f.colorbar(im, ax=axs[0]).ax.set_ylabel('Echo power')
plt.tick_params(axis='x', which='both', bottom=False, top=False, labelbottom=False)
plt.savefig(f'{self.name}.pgf', bbox_inches='tight', transparent=True)
plt.show()

if __name__ == '__main__':
    PlotHK().plot_it() #

A.4 hello_kitty.py

"""Script for calculating the peak power of the plasma line at different aspect angles, height and time of day.
Already implemented are two versions, vol.1 and vol.2. Run from 'main.py'."

import os
import sys
import time
import datetime

import numpy as np
import matplotlib
import matplotlib.pyplot as plt
from matplotlib.backends.backend_pdf import PdfPages
from matplotlib import gridspec
import scipy.integrate as si
import scipy.constants as const
from lmfit.models import LorentzianModel
from tqdm import tqdm

from utils import spectrum_calculation as isr
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'
})
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.
        """
        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
        wave_number
        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': 350000e-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:
params = { 'kappa': 8, 'vdf': 'real_data', 'area': False }
with tqdm(total=len(self.Z) * len(self.A)) as pbar:
    for i, z in enumerate(self.Z):
        sys_set['NE'] = z
        plasma_freq = (sys_set['NE'] * const.elementary_charge**2 /
                       (const.m_e * const.epsilon_0))**.5 / (2 * np.pi)
        cf.I_P['F_MIN'] = plasma_freq
        cf.I_P['F_MAX'] = plasma_freq + 4e5
        cf.f = np.linspace(cf.I_P['F_MIN'], cf.I_P['F_MAX'],
                           int(cf.F_N_POINTS))
        cf.w = 2 * np.pi * cf.f  # Angular frequency
        for j, a in enumerate(self.A):
            sys_set['THETA'] = a * np.pi / 180
            old_stdout = sys.stdout
            f = open(os.devnull, 'w')
            sys.stdout = f
            f, s, meta_data = isr.isr_spectrum
            plasmask, energy_interval, fr = self.check_energy(f, s, a)
            sys.stdout = old_stdout
            self.dots[0].append(energy_interval)
            self.dots[1].append(j)
            self.dots[2].append(z)
            self.fr[i, j] = fr
            self.g[i, j] = plasma_power
            pbar.update(1)
            self.meta.append(meta_data)

def check_energy(self, f, s, deg):
    p = int(np.argmax(s == np.max(s)))
    freq = f[p]
    f_mask = (freq - 5e2 < f) & (f < freq + 5e2)
    x = f[f_mask]
    y = s[f_mask]
    mod = LorentzianModel()
    pars = mod.guess(y, x=x)
    out = mod.fit(y, pars, x=x)
    power = si.simps(out.best_fit, x)
    l = const.c / self.F0
    # Calculate corresponding energy with formula:
    E_plasma = .5 * const.m_e * (freq * l / (2 * np.cos(deg * np.pi /
                           np.sqrt(180)))**2 / const.eV)
    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]]
    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[:, 0]]
        dots_y = np.r_[dots_y, np.ones(len(arg[:, 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'$Electron$ $number$ $density, n_e$)
    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)
tt = time.localtime()
the_time = strftime('%Y-%m-%d %H:%M:%S', t)
save_path = 'hello_kitty_%s.pdf' % the_time
self.meta.insert(0, {'F_MAX': cf.I_P['F_MAX'], 'V_MAX':

pdffig = PdfPages(str(save_path) + '.pdf')
metadata = pdffig.infodict()
metadata['Title'] = 'Hello Kitty plot'
metadata['Author'] = 'Eirik R. Enger'
metadata['Subject'] = 'Plasma line power as a function of ' + 
    'electron number density and aspect angle.'

metadata['Keywords'] = str(self.meta)
metadata['ModDate'] = datetime.datetime.today()
plt.savefig(pdffig, bbox_inches='tight', format='pdf', dpi=600)
pdffig.close()
plt.savefig(f'{save_path}.pgf', bbox_inches='tight', 
    metadata=self.meta)
np.savez(f'{save_path}', angle=self.A, density=self.Z, 
    power=self.g, dots=self.dots, fr=self.fr)

plt.show()

def run(self):
    self.create_data()
    self.plot_data()
def isr_spectrum(version, system_set, kappa=None, vdf=None, area=False, debye=None):
    """Calculate an ISR spectrum using the theory presented by Hagfors [1961] and Mace [2003]."""

    Arguments:
    version {str} -- decide which integral to use when calculating ISR spectrum
    system_set {dict} -- all plasma parameters and other parameters needed in the different calculation methods

    Keyword Arguments:
    kappa {int} -- kappa index used in any kappa distribution (default: {None})
    vdf {str} -- gives the VDF used in the a_vdf calculation (default: {None})
    area {bool} -- if True, calculates the area under the ion line (default: {False})
    debye {str} -- if set to 'maxwell', the Maxwellian Debye length is used (default: {None})

    Returns:
    f {np.ndarray} -- 1D array giving the frequency axis
    Is {np.ndarray} -- 1D array giving the spectrum at the sampled frequencies
    meta_data {dict} -- all parameters used to calculate the returned spectrum

    sys_set, p = correct_inputs(version, system_set.copy(), {'kappa': kappa, 'vdf': vdf})
    kappa, vdf = p['kappa'], p['vdf']
    func = version_check(version, vdf, kappa)
    w_c = w_e_gyro(np.linalg.norm([sys_set['B']]), 2)
    M_i = sys_set['MI'] * (const.m_p + const.m_n) / 2
    W_c = w_ion_gyro(np.linalg.norm([sys_set['B']]), M_i)

    # Ions
    params = {'K_RADAR': sys_set['K_RADAR'], 'THETA': sys_set['THETA'],
              'nu': sys_set['NU_I'], 'm': M_i, 'T': sys_set['T_I'], 'w_c':
              W_c}
    y = np.linspace(0, cf.Y_MAX_i**(1 / cf.ORDER), int(cf.Y_N_POINTS),
                    dtype=np.double)**cf.ORDER
    f_ion = intf.INT_MAXWELL()
    f_ion.initialize(y, params)
    F_i = gordeyev_int_parallel.integrate(M_i, sys_set['T_I'],
                                            sys_set['NU_I'], y, function=f_ion, kappa=kappa)

    # Electrons
    params = {'K_RADAR': sys_set['K_RADAR'], 'THETA': sys_set['THETA'],
              'nu': sys_set['NU_E'], 'm': const.m_e, 'T': sys_set['T_E'],
              'T_ES': sys_set['T_ES'], 'w_c': w_c, 'kappa': kappa, 'vdf':
              vdf, 'Z': sys_set['Z'], 'mat_file': sys_set['mat_file'],
              'pitch_angle': sys_set['pitch_angle']}

    sys_set, p = correct_inputs(version, system_set.copy(), {'kappa': kappa, 'vdf': vdf})
    kappa, vdf = p['kappa'], p['vdf']
    func = version_check(version, vdf, kappa)
    w_c = w_e_gyro(np.linalg.norm([sys_set['B']]), 2)
    M_i = sys_set['MI'] * (const.m_p + const.m_n) / 2
    W_c = w_ion_gyro(np.linalg.norm([sys_set['B']]), M_i)

    # Ions
    params = {'K_RADAR': sys_set['K_RADAR'], 'THETA': sys_set['THETA'],
              'nu': sys_set['NU_I'], 'm': M_i, 'T': sys_set['T_I'], 'w_c':
              W_c}
    y = np.linspace(0, cf.Y_MAX_i**(1 / cf.ORDER), int(cf.Y_N_POINTS),
                    dtype=np.double)**cf.ORDER
    f_ion = intf.INT_MAXWELL()
    f_ion.initialize(y, params)
    F_i = gordeyev_int_parallel.integrate(M_i, sys_set['T_I'],
                                            sys_set['NU_I'], y, function=f_ion, kappa=kappa)
```python
y = np.linspace(0, cf.Y_MAX_e**(1 / cf.ORDER), int(cf.Y_N_POINTS),
                 dtype=np.double)**cf.ORDER

func.initialize(y, params)

Fe = gordeyev_int_parallel.integrate(const.m_e, sys_set['T_E'],
                                      sys_set['NU_E'], y, function=func, kappa=kappa)

Xp_i = np.sqrt(1 / (2 * L_Debye(sys_set['NE'], sys_set['T_E'],
                          kappa=None)**2 * \
                       sys_set['K_RADAR']**2))

if func.the_type == 'maxwell' or debye == 'maxwell':
    Xp_e = np.sqrt(1 / (2 * L_Debye(sys_set['NE'], sys_set['T_E'],
                                   kappa=None)**2 * \
                       sys_set['K_RADAR']**2))

if func.the_type == 'kappa':
    Xp_e = np.sqrt(1 / (2 * L_Debye(sys_set['NE'], sys_set['T_E'],
                                      kappa=kappa)**2 * \
                       sys_set['K_RADAR']**2))

elif func.the_type == 'a_vdf':
    Xp_e = np.sqrt(1 / (2 * L_Debye(sys_set['NE'], sys_set['T_E'],
                                     char_vel=func.char_vel)**2 * \
                       sys_set['K_RADAR']**2))

# In case we have ω = 0 in our frequency array, we just ignore this
# warning message

with np.errstate(divide='ignore', invalid='ignore'):
    Is = sys_set['NE'] / (np.pi * cf.w) * (np.imag(- Fe) * abs(1 + 2 * \
                                          Xp_i**2 * Fi)**2 + (4 * Xp_e**4 * np.imag(- Fi) * abs(Fe)**2)) / abs(1 + 2 * Xp_e**2
                                      -*Fe + 2 * Xp_i**2 * Fi)**2

    if area:
        if cf.I_P['F_MAX'] < 1e4:
            area = si.simps(Is, cf.f)
            print(f'The area under the ion line is ¥1.6e.' % area)
        else:
            print('F_MAX is set too high. The area was not calculated.')

        sys_set['THETA'] = round(params['THETA'] * 180 / np.pi, 1)
        sys_set['version'] = version
        return cf.f, Is, dict(sys_set, **p)

def L_Debye(*args, kappa=None, char_vel=None):
    """Calculate the Debye length."

    Input args may be
    - n_e -- electron number density
    - T_e -- electron temperature
    - T_i -- ion temperature

    Returns:
    - float -- the Debye length

    >>> n_margin = len(args)
```
if nargin == 1:
    n_e = args[0]
elif nargin == 2:
    n_e = args[0]
    T_e = args[1]
elif nargin == 3:
    n_e = args[0]
    T_e = args[1]
    T_i = args[2]

Ep0 = 1e-09 / 36 / np.pi

if nargin < 3:
    if kappa is not None:
        LD = np.sqrt(Ep0 * const.k * T_e / (max(0, n_e) * const.e**2)
                   ) * np.sqrt(((kappa - 3 / 2) / (kappa - 1 / 2))
    elif char_vel is not None:
        LD = np.sqrt(Ep0 * const.k * T_e / (max(0, n_e) * const.e**2)
                   ) * np.sqrt(char_vel)
    else:
        LD = np.sqrt(Ep0 * const.k * T_e /
                      (max(0, n_e) * const.e**2))
    else:
        LD = np.sqrt(Ep0 * const.k /
                      ((max(0, n_e) / T_e + max(0, n_e) / T_i) / const.e**2))

return LD

def w_ion_gyro(B, m_ion):
    """Ion gyro frequency as a function of magnetic field strength and ion mass."

    Arguments:
    B {float} -- magnetic field strength
    m_ion {float} -- ion mass

    Returns:
    float -- ion gyro frequency
    """

    w_e = const.e * B / m_ion

    return w_e

def w_e_gyro(B):
    """Electron gyro frequency as a function of magnetic field strength."

    Arguments:
    B {float} -- magnetic field strength

    Returns:
    float -- electron gyro frequency
    """
w_e = const.e * B / const.m_e

return w_e

def correct_inputs(version, sys_set, params):
    """Extra check suppressing the parameters that was given but is not necessary."
    if version != 'kappa' and not (version == 'a_vdf' and params['vdf'] in ['kappa', 'kappa_vol2']):
        params['kappa'] = None
    if version != 'a_vdf':
        params['vdf'] = None
    if version != 'a_vdf' or params['vdf'] != 'gauss_shell':
        sys_set['T_ES'] = None
    if version != 'a_vdf' or params['vdf'] != 'real_data':
        sys_set['Z'] = None
        sys_set['mat_file'] = None
        sys_set['pitch_angle'] = None
    return sys_set, params

def version_check(version, vdf, kappa):
    """Check if the parameters given are complete.
    Args:
    version {str} -- which Gordeyev integrand to use
    vdf {str} -- which distribution to use
    kappa {int or float} -- kappa index
    Returns:
    object -- an integrand object from 'integrand_functions.py'
    """
    versions = ['kappa', 'maxwell', 'a_vdf']
    try:
        if not version in versions:
            raise SystemError
            print(f'Using version "{version}"', flush=True)
    except SystemError:
        sys.exit(version_error(version, versions))
    if version == 'maxwell':
        func = intf.INT_MAXWELL()
    elif version == 'kappa':
        kappa_check(kappa)
        func = intf.INT_KAPPA()
    elif version == 'a_vdf':
        vdfs = ['maxwell', 'kappa', 'kappa_vol2', 'gauss_shell', 'real_data']
        try:
            if not vdf in vdfs:
                raise SystemError
                print(f'Using VDF "{vdf}"', flush=True)
        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."
    """

    @abstractmethod
    def the_type(self) -> str:
        """The type of the integrand implementation."
        """

    @abstractproperty
    def initialize(self, y, params):
        ..

---

    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."
    """

    @abstractmethod
    def the_type(self) -> str:
        """The type of the integrand implementation."
        """

    @abstractproperty
    def initialize(self, y, params):
        ..
"""Needs an initialization method.

Arguments:
y (np.ndarray) -- array for integration variable
   params (dict) -- dictionary holding all needed parameters
"""

@abstractmethod
def integrand(self):
   """Method that returns the np.ndarray that is used as the integrand.
   """

class INT_KAPPA(INTEGRAND):
   """Implementation of the integrand of the Gordeyev
   integral for the kappa distribution from Mace (2003).
   """
   the_type = 'kappa'

def __init__(self):
   self.y = np.array([])
   self.params = {}
   self.Z = float
   self.Kn = float

def initialize(self, y, params):
   self.y = y
   self.params = params
   self.z_func()

def z_func(self):
   theta_2 = 2 * ((self.params['kappa'] - 3 / 2) /
      self.params['kappa'] * self.params['T'] * const.k /
      self.params['m'])
   self.Z = (2 * self.params['kappa'])**(1 / 2) * \n      (self.params['K_RADAR']**2 * np.sin(self.params['THETA'])**2 *
      theta_2 / self.params['w_c']**2 *
      (1 - np.cos(self.params['w_c'] * self.y)) +
      1 / 2 * self.params['K_RADAR']**2 *
      np.cos(self.params['THETA'])**2 * theta_2 * self.y**2)**(1 / 2)
   self.Kn = sps.kv(self.params['kappa'] + 1 / 2, self.Z)

def integrand(self):
      * self.params['nu'])
   return G
class INT_MAXWELL(INTEGRAND):
    """Implementation of the integrand in the Gordeyev integral for the Maxwellian distribution from e.g. Hagfors (1961) or Mace (2003).
    """
    the_type = 'maxwell'

    def __init__(self):
        self.y = np.array([])
        self.params = {}

    def initialize(self, y, params):
        self.y = y
        self.params = params

    def integrand(self):
        G = np.exp(- self.y * self.params['nu'] -
                   self.params['K_RADAR']**2 *
                   - np.sin(self.params['THETA'])**2 * self.params['T'] *
                   - const.k /
                   (self.params['m'] * self.params['w_c']**2) * (1 -
                   - np.cos(self.params['w_c'] * self.y)) -
                   .5 * (self.params['K_RADAR'] *
                   - np.cos(self.params['THETA']) * self.y)**2 *
                   - self.params['T'] * const.k / self.params['m'])

        return G

class INT_LONG(INTEGRAND):
    """Implementation of the integrand in the Gordeyev integral for the isotropic distribution from Mace (2003).
    """
    the_type = 'a_vdf'

    def __init__(self):
        self.y = np.array([])
        self.params = {}
        self.char_vel = float

    def initialize(self, y, params):
        self.y = y
        self.params = params

    def v_int(self):
        v = np.linspace(0, cf.V_MAX**(1 / cf.ORDER),
                        int(cf.V_N_POINTS))**cf.ORDER
        if self.params['vdf'] == 'maxwell':
f = vdfs.F_MAXWELL(v, self.params)
elif self.params['vdf'] == 'kappa':
    f = vdfs.F_KAPPA(v, self.params)
elif self.params['vdf'] == 'kappa_vol2':
    f = vdfs.F_KAPPA_2(v, self.params)
elif self.params['vdf'] == 'gauss_shell':
    f = vdfs.F_GAUSS_SHELL(v, self.params)
elif self.params['vdf'] == 'real_data':
    f = vdfs.F_REAL_DATA(v, self.params)

# Compare the velocity integral to the Maxwellian case.
# This way we make up for the change in characteristic velocity
# and Debye length for different particle distributions.
res_maxwell = v_int_parallel.integrand(self.y, self.params, v,
                                       vdfs.F_MAXWELL(v, self.params).f_0())
int_maxwell = si.simps(res_maxwell, self.y)
res = v_int_parallel.integrand(self.y, self.params, v, f.f_0())
int_res = si.simps(res, self.y)
# The scaling of the factor describing the characteristic velocity
self.char_vel = int_maxwell / int_res
print(f'Debye length of the current distribution is {self.char_vel} times the Maxwellian Debye length.')
return

def p_d(self):
    # At $y = 0$ we get 0/0, so we use
    # $\lim_{y \to 0} dp/dy = |k|w_c/\sqrt{(w_c^2)}$ (from above, opposite sign from below)
    cos_t = np.cos(self.params['THETA'])
    sin_t = np.sin(self.params['THETA'])
    w_c = self.params['w_c']
    num = abs(self.params['K_RADAR']) * abs(w_c) * 
          (cos_t**2 + w_c * self.y + sin_t**2 + np.sin(w_c * self.y))
    den = w_c * (cos_t**2 + w_c**2 + self.y**2 - 
                    2 * sin_t**2 + np.cos(w_c * self.y) + 
                    2 * sin_t**2)**.5
    # np.sign(y[-1]) takes care of weather the limit should be
    # considered taken from above or below.
    # The last element of the np.ndarray is chosen since it is assumed y
    # runs from 0 to some finite real number.
    first = np.sign(self.y[-1]) * abs(self.params['K_RADAR']) * abs(w_c)
    with np.errstate(divide='ignore', invalid='ignore'):
        out = num / den
    out[np.where(den == 0.)[0]] = first
    return out

def integrand(self):
    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."

```python
from abc import ABC, abstractmethod

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

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):
```

---

```
from abc import ABC, abstractmethod

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

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):
```
func = self.A * np.exp(- self.v**2 / (2 * self.params['T'] * const.k / self.params['m']))

return func

class F_KAPPA(VDF):
    """Create an object that make kappa distribution functions."
    Arguments:
    VDF {ABC} -- abstract base class to make VDF objects
    """
    def __init__(self, v, params):
        """Initialize VDF parameters."
        self.v = v
        self.params = params
        self.normalize()

    def normalize(self):
        self.theta_2 = 2 * ((self.params['kappa'] - 3 / 2) / self.params['kappa']) * self.params['m']
        self.A = (np.pi * self.params['kappa'] * self.theta_2)**(- 3 / 2) * \
                  sps.gamma(self.params['kappa'] + 1) / sps.gamma(self.params['kappa'] - 1 / 2)

    def f_0(self):
        """Return the values along velocity \(v\) of a kappa VDF."
        Kappa VDF used in Gordeyev paper by Mace (2003).

Returns:
    np.ndarray -- 1D array with the VDF values at the sampled points
    """
    func = self.A * (1 + self.v**2 / (self.params['kappa'] * self.theta_2))**(- self.params['kappa'] - 1)

    return func

class F_KAPPA_2(VDF):
    """Create an object that make kappa vol. 2 distribution functions."
    Arguments:
    VDF {ABC} -- abstract base class to make VDF objects
    """
    def __init__(self, v, params):
        """Initialize VDF parameters."

Arguments:

v (np.ndarray) -- 1D array with the sampled velocities
params (dict) -- a dictionary with all needed plasma parameters

```
self.v = v
self.params = params
self.normalize()
```

```
def normalize(self):
    self.v_th = np.sqrt(self.params['T'] * const.k / self.params['m'])
    self.A = (np.pi * self.params['kappa'] * self.v_th**2)**(- 3 / 2) *
              sps.gamma(self.params['kappa']) / sps.gamma(self.params['kappa']


    return f_0
```

```
def f_0(self):
    """Return the values along velocity v of a kappa VDF."

    Kappa VDF used in dispersion relation paper by
    Ziebell, Gaelzer and Simoes (2017). Defined by

    Returns:
    np.ndarray -- 1D array with the VDF values at the sampled points
    
    func = self.A * (1 + self.v**2 / (self.params['kappa'] *
                     self.v_th**2))**( - self.params['kappa']

    return func
```

class F_GAUSS_SHELL(VDF):
    """Create an object that make Gauss shell distribution functions.

    Arguments:
    VDF (ABC) -- abstract base class to make VDF objects
    
    def __init__(self, v, params):
        self.v = v
        self.params = params
        self.vth = np.sqrt(self.params['T'] * const.k / self.params['m'])
        self.r = (self.params['T_ES'] * const.k / self.params['m'])**.5
        self.steep = 5
        self.f_M = F_MAXWELL(self.v, self.params)
        self.normalize()

    def normalize(self):
        func = np.exp(- self.steep * (abs(self.v) - self.r)**2 / (2 *
            self.params['T'] * const.k / self.params['m'])))
        f = func * self.v**2 * 4 * np.pi
        self.A = 1 / si.simpson(f, self.v)
        ev = .5 * const.m_e * self.r**2 / const.eV
        print(f'Gauss shell at E = {round(ev, 2)} eV')
def f_0(self):
    func = self.A * np.exp(- self.steep * (abs(self.v) - self.r)**2 / (2
    * self.params['T'] * const.k / self.params['m'])) + \n    1e4 * self.f_M.f_0()
    return func / (1e4 + 1)

class F_REAL_DATA(VDF):
    """Create an object that make distribution functions from
    a 1D array.
    """
    def __init__(self, v, params):
        self.v = v
        self.params = params
        self.normalize()

    def normalize(self):
        func = read.interpolate_data(self.v, self.params)
        f = func + self.v**2 * 4 * np.pi
        self.A = 1 / si.simps(f, self.v)

    def f_0(self):
        func = self.A * read.interpolate_data(self.v, self.params)
        return func

A.8  read.py

"""This script reads from folder `arecibo` and combines the
calculated electron distribution from file with a Maxwellian.
"""
import os
import sys
import ast
import numpy as np
from scipy.io import loadmat
import scipy.constants as const

def f_0_maxwell(v, params):
    # NOTE: Normalized to 1D
    A = (2 * np.pi * params['T'] * const.k / params['m'])**(-1 / 2)
    func = A * np.exp(-v**2 / (2 * params['T'] * const.k / params['m']))
return func

def interpolate_data(v, params):
    """Interpolate calculated distribution down to zero
    energy and add to a 1D Maxwellian.
    """
    Args:
    v {np.ndarray} -- 1D velocity array
    params {dict} -- dictionary of all needed parameters
    Returns:
    np.ndarray -- 1D array of the distribution
    """
    if os.path.basename(os.path.realpath(sys.argv[0])) != 'main.py':
        path = 'data/arecibo/
        if not os.path.exists(path):
            path = 'program/data/arecibo/
        x = loadmat(path + params['mat_file'])
        data = x['fe_zmuE']
        if isinstance(params['pitch_angle'], list):
            if all(isinstance(x, int) for x in params['pitch_angle']):
                sum_over_pitch = data[:, params['pitch_angle'], :]
                norm = len(params['pitch_angle'])
            else:
                norm = 18
                sum_over_pitch = np.einsum('ijk->ik', data) / norm  # removes
                idx = int(np.argwhere(read_dat_file('z4fe.dat')==params['Z']))
                f_1 = sum_over_pitch[idx, :]
                energies = read_dat_file('E4fe.dat')
        else:
            path = 'data/arecibo/
            x = loadmat(path + params['mat_file'])
            data = x['fe_zmuE']
            if isinstance(params['pitch_angle'], list):
                if all(isinstance(x, int) for x in params['pitch_angle']):
                    sum_over_pitch = data[:, params['pitch_angle'], :]
                    norm = len(params['pitch_angle'])
                else:
                    norm = 18
                    sum_over_pitch = np.einsum('ijk->ik', data) / norm  # removes
                    idx = int(np.argwhere(read_dat_file('z4fe.dat')==params['Z']))
                    f_1 = sum_over_pitch[idx, :]
                    energies = read_dat_file('E4fe.dat')
            velocities = (2 * energies * const.eV / params['m'])**.5
            new_f1 = np.interp(v, velocities, f_1)
            f_0 = f_0_maxwell(v, params)
            f0_f1 = f_0 + new_f1
        return f0_f1
def read_dat_file(file):
    """Return the contents of a `.dat` file as a single numpy row vector.

    Arguments:
    file {str} -- the file name of the .dat file

    Returns:
    np.ndarray -- contents of the .dat file
    """
    l = np.array([])
    path = 'data/arecibo/'
    if not os.path.exists(path):
        path = 'program/data/arecibo/
    with open(path + file) as f:
        ll = f.readlines()
        ll = [x.strip() for x in ll]
        l = np.r_[l, ll]
    if len(l) == 1:
        for p in l:
            l = p.split()
    e = []
    for p in l:
        k = ast.literal_eval(p)
        e.append(k)
    return np.array(e)

A.9  test_ISR.py

"""This script implements tests for functions used throughout the program.

Run from directory `program` with command python -m unittest test.test_ISR -b
"""
import multiprocessing as mp
mp.set_start_method('fork')
import unittest  # pylint: disable=C0413
import numpy as np  # pylint: disable=C0413
import scipy.integrate as si  # pylint: disable=C0413
import scipy.constants as const  # pylint: disable=C0413
from utils import spectrum_calculation as isr  # pylint: disable=C0413
from utils import vdfs  # pylint: disable=C0413

class TestISR(unittest.TestCase):
    """Check if the output from isr_spectrum is as expected."""
should return two numpy.ndarrays of equal shape.

Arguments:

```
unittest.TestCase (class) -- inherits from unittest
to make it a TestCase
```
def test_isr_long_calc_real(self):
    self.params['vdf'] = 'real_data'
    self.a, self.b, meta_data = isr_isr_spectrum('a_vdf', self.sys_set,
                                                **self.params)
    self.assertEqual(meta_data['kappa'], None)
    self.assertEqual(meta_data['vdf'], 'real_data')
    self.assertEqual(meta_data['T_ES'], None)
    self.assertEqual(meta_data['Z'], 599)
    self.assertEqual(meta_data['mat_file'], 'fe_zmuE-07.mat')

# Reference to TestVDF

class TestVDF(unittest.TestCase):
    """Class which test if the VDFs are normalized."
    Arguments:
        unittest.TestCase {class} -- inherits from unittest
to make it a TestCase
    """

    @classmethod
    def setUpClass(cls):
        cls.v = np.linspace(0, (6e6)**(1 / 3), int(4e4))**3
        cls.params = {'m': 9.1093837015e-31, 'T': 1000,
                      'kappa': 3, 'T_ES': 90000, 'Z': 300,
                      'mat_file': 'fe_zmuE-07.mat', 'pitch_angle': 'all'}
        cls.f = None
    # cls.fs = []

    @classmethod
    def tearDownClass(cls):
        np.savez('f', v=cls.v, m=cls.fs[0], k=cls.fs[1], r=cls.fs[2])

    def tearDown(self):
        # The function f is scaled with the Jacobian of cartesian to
        # spherical
        f = self.f.f_0() * self.v**2 * 4 * np.pi
        res = si.simps(f, self.v)
        self.assertAlmostEqual(res, 1, places=6)

    def test_vdf_maxwell(self):
        self.f = vdfs.F_MAXWELL(self.v, self.params)
        # self.fs.insert(0, self.f.f_0())

    def test_vdf_kappa(self):
        self.f = vdfs.F_KAPPA(self.v, self.params)
        # self.fs.insert(1, self.f.f_0())

        # def test_vdf_kappa_vol2(self):
        #    self.f = vdfs.F_KAPPA_2(self.v, self.params)

        # def test_vdf_gauss_shell(self):
        #    self.f = vdfs.F_GAUSS_SHELL(self.v, self.params)
A.10 gordeyev_int_parallel.py

```python
"""Implementation of parallel computation of the Gordeyev integral as a function of frequency."
"""

import ctypes
import multiprocessing as mp
from functools import partial
import numpy as np
import scipy.special as sps
import scipy.constants as const
import scipy.integrate as si

from inputs import config as cf

def integrate(m, T, nu, y, function, kappa=None):
    """Integrate from `0` to `Y_MAX` with an integrand on the form
    `e^{-iwy}f(y)`, for every value in the np.ndarray `w`.
    """
    idx = set(enumerate(cf.w))
    f = function.integrand()
    func = partial(parallel, y, f)
    pool = mp.Pool()
    pool.map(func, idx)
    pool.close()
    pool.join()
    return f

def test_vdf_real_data(self):
    self.f = vdfs.F_REAL_DATA(self.v, self.params)
    # self.fs.insert(2, self.f.f_0())
    if __name__ == '__main__':
        unittest.main()
```

```
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 = Nf
array = shared_array((int(cf.F_N_POINTS),))
```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.""
    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
```
```python
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
```

```python
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
```

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

---

**A.12  plot_class.py**

```python
"""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'],
                        'version': version})
\texttt{V\_MAX} : cf. \texttt{V\_MAX}, \texttt{F\_N\_POINTS}:
\texttt{→ cf.} \texttt{F\_N\_POINTS},
\texttt{Y\_N\_POINTS} : cf. \texttt{Y\_N\_POINTS}, \texttt{V\_N\_POINTS}:
\texttt{→ cf.} \texttt{V\_N\_POINTS})

\texttt{tt = time.localtime()}
\texttt{the\_time = f\{tt[0]\}._{tt[1]}._{tt[2]}._{tt[3]}._{tt[4]}._{tt[5]}\}}
\texttt{save\_path = '.../.../report/master-thesis/figures/in\_use'
\texttt{if not os.path.exists(save\_path):
\texttt{save\_path = '../figures'
\texttt{os.makedirs(save\_path, exist\_ok=True)
\texttt{if self.save\_path is None:
\texttt{self.save\_path = f\{save\_path\}/(the\_time)\}(version)'
\texttt{else:
\texttt{self.save\_path = save\_path + '/'} + self.save\_path
\texttt{np.savez(f\{self.save\_path\}, frequency=f, spectra=data,
\texttt{→ legend\_txt=l\_txt, ridge\_txt=r\_txt, meta=\texttt{params}
\texttt{self.pdf\_fig = PdfPages(str(self.save\_path) + '.pdf')
\texttt{metadata = self.pdf\_fig\texttt{.infodict()}
\texttt{metadata['Title'] = f'ISR Spectrum w/ \{version}\'
\texttt{metadata['Author'] = 'Eirik R. Enger'
\texttt{metadata['Subject'] = 'IS spectrum made using a \{version\} distribution \ + \
\texttt{→ and Simpson's integration rule.'
\texttt{metadata['Keywords'] = f'\texttt{\{params\}'
\texttt{metadata['ModDate'] = datetime.datetime.today()}
\texttt{def plot\_normal(self, f, Is, func\_type, l\_txt):
\texttt{'''Make a plot using \`f` as \`x\` axis and \`Is\` as \`y\` axis.
\texttt{Arguments:
\texttt{f \{np.ndarray\} -- variable along x axis
\texttt{Is \{list\} -- list of np.ndarray\s that give the y axis
\texttt{→ values along x axis
\texttt{func\_type \{str\} -- attribute of the matplotlib.pyplot object
\texttt{l\_txt \{list\} -- a list of strings that give the legend
\texttt{→ of the spectra. Same length as the inner lists
\texttt{'''}
\texttt{try:
\texttt{getattr(plt, func\_type)
\texttt{except Exception:
\texttt{print(f'\{func\_type\} is not an attribute of \texttt{the} \ + \
\texttt{→ matplotlib.pyplot object. Using "plot".')}
\texttt{func\_type = 'plot'
\texttt{if len(Is) \!= len(l\_txt):
\texttt{print('Warning: The number of spectra does \ + \
\texttt{→ not match the number of labels.')}
\texttt{self.colors = np.linspace(0, 1, len(Is))
\texttt{Is = Is.copy()}
\texttt{# TODO: should probably remove this
\texttt{# Linear plot show only ion line (kHz range).
\texttt{if func\_type \== 'plot' and not self.plasma:
\texttt{f, Is = self.only\_ion\_line(f, Is)
\texttt{p, freq, exp = self.scale\_f(f)
\texttt{plt.figure(figsize=(6, 3))}}
if self.plasma:
    # Clip the frequency axis around the plasma frequency.
    mask = self.find_p_line(freq * 10**exp, Is)
    freq = freq[mask]
if func_type == 'semilogy':
    plt.xlabel(f'Frequency [{p}Hz]')
    plt.ylabel('Echo power [dB]')
    for i, _ in enumerate(Is):
        Is[i] = 10 * np.log10(Is[i])
else:
    plt.xlabel(f'Frequency [{p}Hz]')
    plt.ylabel('Echo power')
    for clr, st, s, lab in zip(itertools.cycle(self.colors),
                               itertools.cycle(self.line_styles), Is, l_txt):
        if self.plasma:
            s = s[mask]
            if func_type == 'semilogy':
                plt.plot(freq, s, linestyle=st, alpha=.7, color=(clr, 0., 0.),
                          linewidth=.8, label=lab)
        else:
            plot_object = getattr(plt, func_type)
            plot_object(freq, s, linestyle=st, alpha=.7, color=(clr, 0., 0.),
                          linewidth=.8, label=lab)
plt.legend()
plt.minorticks_on()
plt.grid(True, which="major", ls="--", alpha=0.4)
plt.tight_layout()

if self.save in ['y', 'yes']:
    self.pdffig.attach_note(func_type)
    plt.savefig(self.pdffig, bbox_inches='tight', format='pdf',
                dpi=600)
    plt.savefig(str(self.save_path) + f'\_page_{self.page}.pgf',
                bbox_inches='tight')
    self.page += 1

def plot_ridge(self, frequency, multi_parameters, func_type, l_txt,
               ridge_txt=None):
    """Make a ridge plot of several spectra."
    Arguments:
    frequency (np.ndarray) -- frequency axis
    multi_parameters (list) -- list (outer) containing
      lists (inner) of np.ndarrays. The arrays
      contain the spectrum values at the frequencies
      given by "frequency"
    func_type (str) -- attribute of the matplotlib.pyplot class
    l_txt (list) -- a list of strings that give the legend of the
      spectra. Same length as the inner lists
    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:
        s = s[mask]
    plot_object = getattr(ax_objs[-1], func_type)
    plot_object(freq, p, color=(Rgb[j], 0., 1 - Rgb[j]),
            linewidth=1, label=lab, linestyle=st)
```python
if first == 0:
    idx = np.argwhere(freq > ax_objs[-1].viewLim.x0)[0]
    legend_pos = (ax_objs[-1].viewLim.x1, np.max(s))
    y0 = s[idx]
    ax_objs[-1].text(freq[idx], s[idx], ridge_txt[j],
                     fontsize=14, ha="right", va="bottom")

    first += 1
if j == 0:
    plt.legend(loc='upper right', bbox_to_anchor=legend_pos,
               bbox_transform=ax_objs[-1].transData)

    if func_type == 'plot':
        # Make a vertical line of comparable size in all plots.
        self.match_box(f_original, freq, multi_params, [y0, j])

    self.remove_background(ax_objs[-1], multi_params, j, p)

gs.update(hspace=-0.6)
if self.save in ['y', 'yes']:
    self.pdffig.attach_note(func_type)
    plt.savefig(self.pdffig, bbox_inches='tight', format='pdf',
                dpi=600)
    plt.savefig(str(self.save_path) + f'_page_{self.page}.pgf',
                bbox_inches='tight')
    self.page += 1

@staticmethod
def remove_background(plt_obj, multi_params, j, p):
    # Make the background transparent
    rect = plt_obj.patch
    rect.set_alpha(0)
    # Remove borders, axis ticks and labels
    plt_obj.set_yticklabels([])
    plt.tick_params(axis='y', which='both', bottom=False,
                    labelbottom=False)
    if j == len(multi_params) - 1:
        plt.xlabel(f'Frequency [{p[0]}Hz]')
    else:
        plt.tick_params(axis='x', which='both', bottom=False,
                        top=False, labelbottom=False)

    spines = ['top', 'right', 'left', 'bottom']
    for sp in spines:
        plt_obj.spines[sp].set_visible(False)

@staticmethod
def scale_f(frequency):
    """Scale the axis and add the corresponding SI prefix."

    Arguments:
        frequency {np.ndarray} -- the variable along an axis

    Returns:
        str, np.ndarray, int -- the prefix, the scaled variables, the
```
exponent corresponding to the prefix

```python
freq = np.copy(frequency)
exp = sip.split(np.max(freq))[1]
freq /= 10**exp
pre = sip.prefix(exp)
return pre, freq, exp
```

```python
@staticmethod
def find_p_line(freq, spectrum):
    """Find the frequency that is most likely the peak
    of the plasma line and return the lower and upper
    bounds for an interval around the peak.
    Arguments:
    freq {np.ndarray} -- sample points of frequency parameter
    spectrum {list} -- list of np.ndarray, values of spectrum
    at the sampled frequencies
    Keyword Arguments:
    check {bool} -- used in correct_inputs to check if plasma
    plots are possible (default: {False})
    Returns:
    np.ndarray -- array with boolean elements
    ""
    spec = spectrum[0]
    try:
        # Assumes that the rightmost peak (highest frequency) is the
        # plasma line
        p = signal.find_peaks(spec, height=10)[0][-1]
    except Exception:
        print('Warning: did not find any plasma line')
    return freq < np.inf
    f = freq[p]
    lower, upper = f - 1e6, f + 1e6
    # Don't want the ion line to ruin the scaling of the y axis
    if lower < 1e5:
        lower = 1e5
    return (freq > lower) & (freq < upper)
```

```python
@staticmethod
def only_ionline(f, Is):
    Is = Is.copy()
    idx = np.argwhere(abs(f) < 4e4)
    if len(idx) < 3:
        return f, Is
    f = f[idx].reshape((-1,))
    for i, _ in enumerate(Is):
        Is[i] = Is[i][idx].reshape((-1,))
    return f, Is
```
```python
def match_box(self, freq_original, freq, multi_parameters, args):
    """Create a scaling box for easier comparison of the ridges.
    Should cover as much as possible in the ridge that span the
    smallest range along the 'y' axis.
    """
    multi_params = multi_parameters.copy()
    v_line_x = np.linspace(.04, .2, len(multi_params))
    if self.plasma:
        f = freq_original.copy()
        spec = multi_params[0]
        mask = self.find_p_line(f, spec)
        diff = np.inf
    for params in multi_params:
        plot_diff = 0
        for s in params:
            if self.plasma:
                s = s[mask]
                difference = np.max(s) - np.min(s)
            if plot_diff < difference:
                plot_diff = difference
        if plot_diff < diff:
            diff = plot_diff
            x0 = np.min(freq) + (np.max(freq) - np.min(freq)) * v_line_x[0]
            plt.vlines(x=x0, ymin=args[0], ymax=args[0] + int(np.ceil(diff / 10) * 5), color='k', linewidth=3)
            plt.text(x0, args[0] + int(np.ceil(diff / 10) * 5) / 2, r'{}$\text{format}(\int\text{np.ceil(diff / 10) * 5})$, rotation=90, ha='right', va='center')
```

Bibliography


