NTNU
Norwegian University of
Science and Technology

Developing an analysis technique for identification of wave structures in MATS data

Andreas Fjeldstad

Master of Science
Submission date: June 2018
Supervisor: Patrick Joseph Espy, IFY

Norwegian University of Science and Technology
Department of Physics

Abstract

This thesis presents a method for analyzing data gathered during the MATS mission. The MATS mission is a Swedish satellite mission which aims to improve our understanding of gravity waves in the mesosphere and their influence on the atmosphere. The MATS satellite will be launched in 2019, and observations will mainly be done by imaging the limb of the atmosphere.

The development of a technique for analyzing such data resulted in a model which returns wave characteristics of waves identified within data sets similar to what is expected from the MATS mission. The model was tested with both synthetic data and more realistic data from a forward model made to simulate MATS data. The synthetic data was made to simulate gravity waves in the mesosphere, and was mainly used during the development of the model.

The model developed treats and filters the data, and returns wave characteristics of waves present within the data set. The results presented in this thesis show that the model often mistakes noise and side lobes for waves, leading to an output not consistent with the expected results when synthetic data is used. However, the model was able to find the wave characteristics of data accepted as wave peaks, leading to the conclusion that the problems and inaccuracies in the output are from the identification of waves, and not from the calculations of the wave characteristics. Thus, further work with filtering and identifying waves in data fields resembling MATS data is needed.

Sammendrag

I denne oppgaven presenteres det en metode for å analysere data fra MATS-oppdraget. MATS-oppdraget er et svensk satellittoppdrag som har som mål å bedre vår forståelse av gravitasjonsbølger i mesosfæren og deres påvirkning på atmosfæren. MATS-satellitten skal skytes opp i løpet av 2019, og observasjonene vil hovedsaklig gjøres ved å ta bilder av deler av atmosfæren. Utviklingen av en teknikk for å analysere slike data resulterte i en modell som returnerer bølgeegenskapene til bølger som er identifisert i datasett med bølger lagd for å etterligne gravitasjonsbølger. Modellen ble testet med både syntetiske data og realistiske data fra en modell utviklet for å etterligne MATS-dataene. Syntetiske data ble laget slik at den best mulig skulle etterligne gravitasjonsbølger i mesosfæren, og det var hovedsaklig disse dataene som ble analysert under utvikling av modellen. Modellen behandler og filtrerer data, og returnerer bølgeegenskaper fra bølger som finnes i datasettene. Resultatene som presenteres i denne oppgaven viser at modellen ofte tar feil av støy og bølger, noe som fører til at resultatene fra modellen ikke er som forventet med dataene den behandler. Likevel klarte modellen å beregne bølgeegenskapene til dataene som ble registrert som bølger, noe som leder til konklusjonen at feilene og unøyaktighetene i dataene modellen returnerer er grunnet feil i identifiseringen av bølger, og ikke fra beregningene av bølgeegenskapene. Videre arbeid med modellen vil derfor bestå av å bedre modellens evne til å skille mellom støy og bølger.

Preface

This work is a continuation of the work done in my specialization project, which was carried out during the autumn of 2017. Most of the work included in this thesis was carried out during the spring of 2018.

I would like to express my gratitude to Professor Patrick Espy for great help and guidance in the writing of this thesis, and for the insightful discussions and explanations of the nature of gravity waves and concepts of signal processing.

Andreas Fjeldstad
June 2018
Trondheim

Contents

List of Figures

List of Tables

Abbreviations

MATS	=	Mesopheric Airglow/Aerosol Tomography and Spectroscopy
DALR	=	Dry Adiabatic Lapse Rate
DFT	=	Discrete Fourier Transform
FFT	=	Fast Fourier Transform
FWHM	=	Full Width Half Maximum
HWHM	=	Half Width Half Maximum
FIR	=	Finite Impulse Response

1 Introduction

The MATS mission is a Swedish satellite mission which aims to improve our understanding of gravity waves in the mesosphere and their influence on the atmosphere. MATS is an acronym for Mesospheric Airglow/Aerosol Tomography and Spectroscopy. Gravity waves are internal mechanical waves that exist throughout the atmosphere, and is a source of dynamic coupling between different layers of the atmosphere. The mesosphere is often viewed as a transition region between space and the atmosphere, and with the aim to understand the atmosphere as a whole, we need to obtain a greater understanding of the mechanics of this transition region and its coupling to atmospheric conditions below and above. This mission will be able to provide a global scale mapping of gravity waves, which can be used to further develop atmospheric global scale models and weather forecast models. The satellite will be launched in 2019, and the mission is planned to last for two years.

Prior to launch methods for analyzing and interpreting data gathered by the satellite must be developed. Members of the collaboration group from Chalmers University of Technology in Gothenburg and Stockholm University have developed a model for simulating realistic gravity waves in the mesosphere. The goal of this project is to develop a model able to analyze and interpret data gathered during the MATS mission. The model, whose input is data containing possible wave structures, aims to have an output with discovered wave characteristics of these waves. A method used during development of this model is to have synthetic data as input and develop means of analyzing this data, before testing the model with realistic data provided by the forward model mentioned above.

This thesis gives an introduction to gravity waves and the MATS mission. Theory used when developing the model is presented, then techniques and methods used to develop the model are presented. Results from the development of the model are shown, both with synthetic data and data from the forward model. Further, the resulting model is discussed, based on theory and aims for the project.

2 Theory

To develop a method or model for analyzing data gathered from the MATS mission, some theory to base such a model on is needed. The theoretical foundation of the MATS mission is gravity waves and how these waves affect the surrounding atmosphere. The theory includes a short introduction of the atmosphere, with focus on the mesosphere, which is the atmospheric part of interest in the MATS mission. Furthermore, gravity waves and the MATS mission is explained and elaborated upon, before theory used when developing the model is presented and explained.

2.1 The atmosphere

The atmosphere consists of several layers, or spheres, divided according to their temperature profile. The different layers and their temperature change in vertical direction are shown in figure 2.1, as a plot of temperature with respect to height. As can be seen from this figure, the layers are divided based on the sign of the temperature gradient, Γ, which is the rate of decrease of temperature with height, called the lapse rate. It can be represented as

$$\Gamma = -\frac{dT}{dz}. \tag{2.1}$$

The thermodynamic change in temperature of a mass of dry air as it undergoes adiabatic changes in pressure associated with changes in altitude is called the dry adiabatic lapse rate (DALR) and can be approximated to 9.8 K/km [1]. If a parcel of air is displaced upwards in an atmosphere where Γ is less than the DALR, the parcel will be denser than its surroundings. It will tend to fall back towards its original altitude, and we say that the atmosphere is stable. The atmosphere consists of a mixture of ideal gases, where nitrogen and oxygen are dominant by volume, but minor constituents such as water vapor, ozone and carbon dioxide have huge impact on the atmosphere and its dynamics.

The troposphere, stratosphere and mesosphere are together called the homosphere, and is composed of mostly nitrogen and oxygen. Of particular interest in this project is the mesosphere, as this is the layer where the satellite will gather its data. The lower part of the mesosphere is located at a height of approximately 50 km above Earth's surface. The upper boundary is called the mesopause, and is located at a height of approximately 100 km above Earth's surface. The altitude limits of the mesosphere depend on

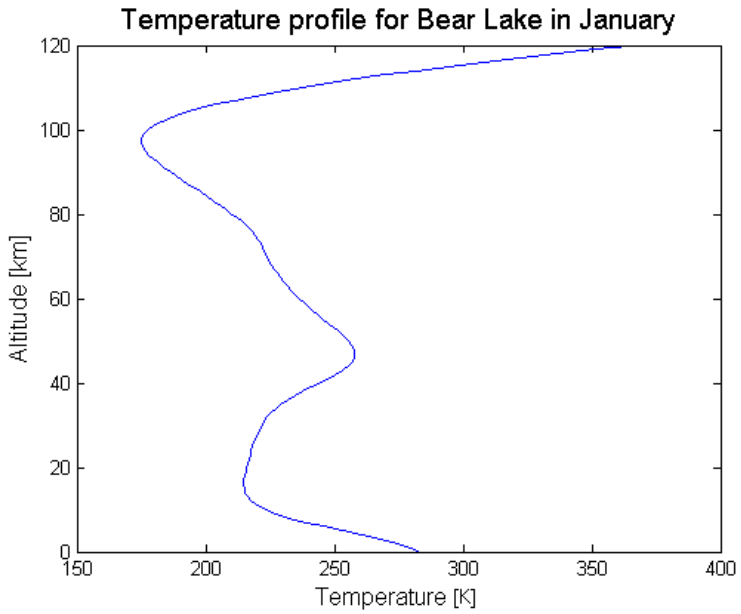

Figure 2.1: The temperature (in Kelvin) in the lowest 120 km of the atmosphere. The layers of the homosphere, based on their temperature profile in the vertical, are shown with their respective pause. The model used is NRLMSISE-00 [2]. The data obtained is from Bear Lake, US (44°N) in January.

season, and will thus vary through the year. As can be seen in figure 2.1 the temperature gradient is negative in the mesosphere. One of the reasons the MATS mission will be important is because the mesosphere has been found to be difficult to study. This is due to weather balloons not being able to fly high enough to provide in situ measurements, and satellites orbiting above this layer. This means that sounding rockets are the only means of performing direct observations, but such observational flights are both brief and infrequent. Waves originating from lower parts of the atmosphere are able to grow to large amplitudes and deposit energy and momentum, affecting among others global wind patterns. One of the main dynamic processes in the mesosphere is due to a type of atmospheric wave called internal atmospheric gravity wave, or gravity wave for short.

2.2 Gravity waves

Atmospheric gravity waves are mechanical waves that may occur internally in the atmosphere or in the transition between two media. These waves have been subject to intense research since the 1960s, due to their contributions to atmospheric structure, variability and circulation [3]. Gravity waves that occur within a medium are called internal gravity waves, and gravity waves that occur at an interface are called surface gravity waves. Gravity waves are common in Earth's atmosphere as they are lightly damped and can be produced by a variety of sources [4]. These waves can be generated from e.g. thunderstorms, mountains, convection and shear, to mention some. One of the most important sources of gravity waves is topography. Topographic waves are often called mountain waves or lee waves. When air blows towards a large topographic object (e.g. a mountain range), the air is pushed to higher altitudes. When a parcel of dense air is moved into a region of less dense air, gravity will pull the parcel of heavier air downwards. The parcel may then overshoot, and since buoyancy will push parcels lighter than their surroundings upwards, the parcel will be pushed back to higher altitudes. This movement is similar to a mass attached to a spring, oscillating in both x- and z-direction, as illustrated in figure 2.2.

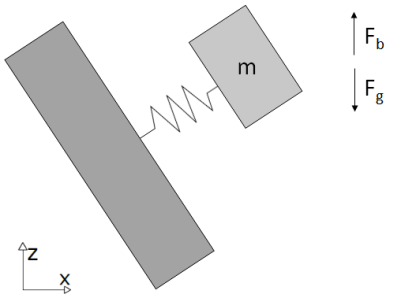

Figure 2.2: A mass attached to a spring. The mass oscillates in the same manner as a parcel of air may oscillate. The movement can move the perturbations to adjacent parcels of air. F_g represents gravity force, and is directed in negative z-direction. F_b represents buoyancy force, and is directed in positive z-direction.

A motion as described above can push layers of air surrounding the parcel, which in vertical direction results in the wave propagating to higher or lower altitude, as illustrated in figure 2.3. In time, the wave will break and

deposit momentum and energy into its surrounding atmosphere. Movement in horizontal direction moves the perturbations to parcels of air adjacent in horizontal direction, and allows the wave to propagate away from its source.

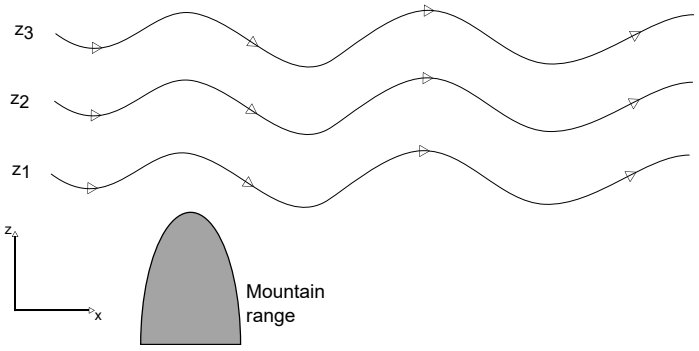

Figure 2.3: An illustration of a mountain wave, an internal gravity wave that may form when airflow blows over a mountain range. Each line represents a particular streamline of the airflow, at different altitudes z_1, z_2 and z_3. The arrows indicate the direction of the airflow.

2.3 Effects of gravity waves in the atmosphere

Gravity waves are recognized for their role in transporting energy and momentum to different parts of the atmosphere. In addition, the waves influence the mean circulation, temperature and density profiles of the atmosphere and contribute to turbulence and mixing. Especially in the mesosphere, waves may propagate to such large amplitudes they will affect the mean atmospheric state. The spatial resolution needed to model effects of gravity waves on larger scale circulation is overwhelmingly fine, requiring the waves to be parametrized. The waves influencing the circulation in the atmosphere may have horizontal wavelengths ranging from tens to thousands of kilometers. Vertically, wavelengths may range substantially due to wind shear among other things. In the stratosphere, waves with vertical wavelengths as small as 1-2 km may be important, and waves with periods as short as 10 minutes are able to carry significant momentum flux vertically. Sources of these waves may be processes that are poorly resolved and parametrized [3].

An acknowledgment regarding global model studies is that in order to describe gravity waves through parameterization, efforts to understand and observe climatology of such waves are needed, and gravity wave climatology

is thus an area of active research [3]. In this context, climatology refers to variations in characteristics of gravity waves such as phase speed and wavelength, and temporal and geographical variations in activity. Guidelines to input parameters describing sources and estimation of wave dissipation as a function of height are important for parameterization of these waves. Currently, model studies of gravity waves are often idealized and poorly constrained by observations. The evidence for global variations in sources of gravity wave activity is lacking, but the importance of sources such as topography and convection are evident, and thus sources must vary seasonally and geographically. An important goal of future observational gravity wave studies is to detect variations between gravity wave activity and climatological patterns. A means of separating variations due to background atmosphere and variations due to gravity wave sources can then be found by combining observations and modeling tools. This can help determine whether a monthly mean of gravity wave activity represents constant, small amplitude wave activity or large outbursts of such activities. Knowledge of generation mechanics and their resulting wave characteristics is needed to provide an accurate description of gravity wave effects in global models [3].

The observational method which holds the most promise for observing geographical and temporal coverage needed to understand gravity wave variability is satellite observations [3].

2.4 The MATS mission

MATS is short for Mesospheric Airglow/Aerosol Tomography and Spectroscopy, and is a Swedish satellite mission which aims to improve our understanding of gravity waves in the mesosphere. The satellite's orbit enables it to provide global scale mapping of gravity waves in the region of interest, namely the mesosphere. The mission will thus provide contributions to understanding the climatology of gravity waves in the mesosphere and mapping of sources of such waves in more detail than previous observations. As a result, the mission may contribute to making global climate models and weather forecast models more precise in the future. The satellite will be launched in 2019, and will orbit in a sun-synchronous orbit 600 km above Earth's surface using optical remote sensing to observe gravity waves. The mission will last for two years. The satellite has a mass of about 50 kg, and its dimensions is $60 \times 70 \times 85$ cm. The spacecraft is based on a spacecraft concept called InnoSat. InnoSat is a small, low cost satellite platform and is intended for scientific research missions in Low Earth Orbit, meaning orbits in an altitude of 160 - 2000 km.

On board the satellite are two scientific instruments, a limb imager and a nadir imager. The field of view for the limb imager is 250 km in the horizontal and 60 km in the vertical, but may be less depending on operational mode. The satellite, with it's scientific equipment, is shown in figure 2.4. The mission aims to do three dimensional reconstruction of gravity waves by performing a tomographic analysis of the images. Observations will be done mainly by imaging the limb of the atmosphere, and the viewing direction is in the track of the satellite orbit. The imaging will be done at six different wavelengths; two in UV (between 270-300 nm) and four in IR (between 760-780 nm) [5]. To investigate atmospheric waves, the satellite will be imaging structures in noctilucent clouds, which are clouds that form at an altitude of around 80 km, in addition to variation in light emitted by oxygen molecules at an altitude of around 100 km, so called airglow.

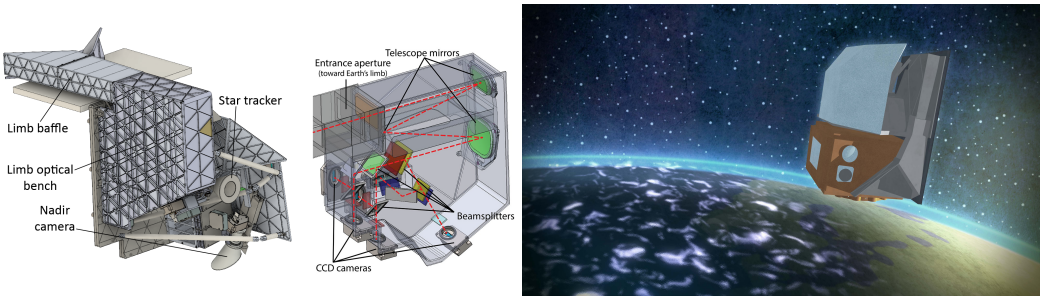

Figure 2.4: Two figures showing different drawings of the MATS satellite. The image to the left shows the scientific instruments on board the satellite, while the image to the right is an early illustration of the satellite.

Airglow is a result of sunlight interacting with molecules in the atmosphere, and can be divided into dayglow and nightglow, depending on whether it occurs during daytime or nighttime. Since the emission from airglow is weak, it is easiest to detect at night, when the far more dominant sunlight is absent. Such nightglow can be seen as a thin, green shell in figure 2.5. This optical radiation is emitted by excited atoms, molecules or ions which are generated in a variety of production and excitation processes during the day when sunlight deposits energy into the atmosphere [6]. The largest portion of the chemical energy stored in the upper part of the atmosphere is carried by oxygen atoms. These atoms eventually participate in chemical reactions leading to emission of radiation, called chemiluminescence. These processes continue throughout the night, and produce nightglow in a 10-20 km thick layer at an altitude of approximately 90 km [6]. Excess energy is carried away by the products of the reaction

$$O + O + M \rightarrow O_2^* + M, \tag{2.2}$$

where O is atomic oxygen, O_2^* is a newly formed O_2 molecule with internal energy, and M represents a third body, which in the mesopause may be any of the major atmospheric components N_2, O_2 or O. Energy may be released, either through radiative relaxation after the above reaction or by other reactions, and can be observed as nightglow. The system of interest in this mission is the O_2 A-band, which is one of the lowest excited states [6].

Figure 2.5: An image of Earth at night, taken from ISS [7]. The red light on the ground is from wildfire in Australia, while airglow is shown as a thin, green layer in the atmosphere.

2.5 Wave properties

The data from the satellite will provide the first three dimensional reconstruction of gravity waves in the upper mesosphere [5]. Methods for analyzing data generated by the satellite are necessary, and a mean of automatically identifying and extracting wave properties from this data is thus needed.

This will be done by developing a model whose input is MATS data and output is prospective wave properties. A method of analyzing this data is with the Fourier transform in two dimensions. The input in the analysis will be points sampled from data gathered by the satellite. The Fourier transform of a complex function $\mathbf{g}$ of two independent variables, x and y, will be represented here by $\mathcal{F}\{\mathbf{g}\}$ and is defined as

$$\mathcal{F}\{\mathbf{g}\} = \iint_{-\infty}^{+\infty} \mathbf{g}(x,y) \exp[-j2\pi(f_X x + f_Y y)]\, dx\, dy. \tag{2.3}$$

The transform is itself a complex-valued function of two independent variables, f_X and f_Y [8].

When in Fourier domain, the amplitude, phase and wavelength of the waves can be extracted from the data and used to identify prospective gravity waves within the data. The gathered data points will be run through a model consisting of several steps to determine whether the sampled data contains wave characteristics.

The amplitude of the Fourier transform of each data point can be defined by

$$A = \sqrt{\mathrm{Re}^2 + \mathrm{Im}^2}, \tag{2.4}$$

where "Re" and "Im" are the real and imaginary part, respectively, of a complex number. Such a complex number can be represented by

$$z = a + b \cdot j,$$

where a and b are real numbers and j is a solution to the equation $x^2 = -1$.

The phase is given by

$$\phi = \arctan\left(\frac{\mathrm{Im}}{\mathrm{Re}}\right). \tag{2.5}$$

A one-dimensional wave as the one in figure 2.6 shows how the real and imaginary part of a wave in Fourier space can describe the phase of a wave. Note in the figure the change in amplitude in Fourier space as the wave is

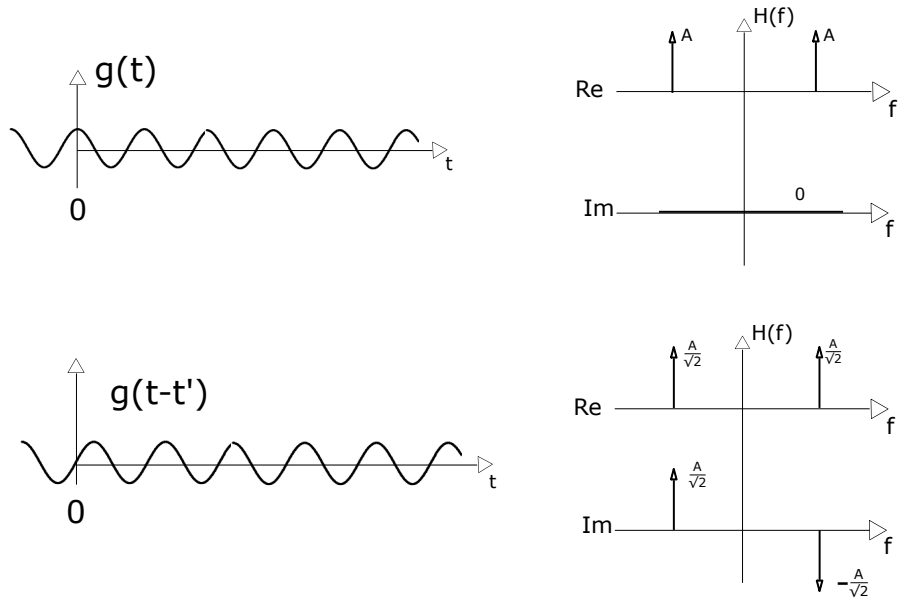

Figure 2.6: A one-dimensional wave shown in both time domain (left) and frequency domain (right). The second wave is equal to the first, but time shifted by a constant t'. The figure shows how time shifting results in a change in the phase angle, as can be seen from the relationship in equation (2.5).

time shifted. However, time shifting does not change the total magnitude of the Fourier transform [9].

When a one-dimensional wave $\mathbf{g}(t)$ is shifted by a constant t', the Fourier transform becomes

$$\int_{-\infty}^{+\infty} \mathbf{g}(t - t') \exp[-j2\pi f_T t]\, dt = H(f) \exp[-j2\pi f_T t']. \tag{2.6}$$

By Euler's formula [10], this can be written in the form

$$H(f) \exp[-j2\pi f_T t'] = H(f)[\cos(2\pi f_T t') - j \sin(2\pi f_T t')],$$

and when combining this with equation (2.4), the magnitude, or amplitude,

is given as

$$\begin{aligned} &|H(f)[\cos(2\pi f_T t') - j\sin(2\pi f_T t')]| \\ &= \sqrt{H^2(f)[\cos^2(2\pi f_T t') + \sin^2(2\pi f_T t')]} \\ &= \sqrt{H^2(f)}, \end{aligned} \tag{2.7}$$

where the relationship $\cos^2 x + \sin^2 x = 1$ and the definition of a complex number j as the solution of the equation $x^2 = -1$, is used.

The fact that the amplitude of the wave does not change because of time shifting makes the amplitude a reasonable property to use for filtering wave structures when analyzing the data.

The phase can be used to identify waves propagating across several altitudes. The phase in a wave will change in an orderly fashion, meaning it will change in a linear, quadratic or other mathematically easily described manner, while noise will have completely random phases. This is illustrated in figure 2.7, where the phase is represented by the blue points. The phase shifts towards right as the wave propagates in z-direction, showing how phase can progress with altitude.

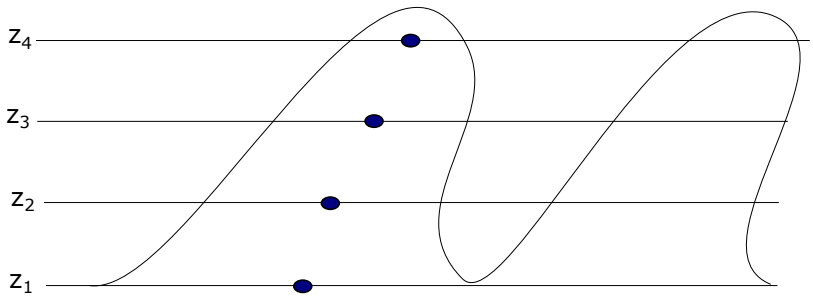

Figure 2.7: A one-dimensional wave propagating across multiple altitudes z_n. The blue points represent the phase, and show how it shifts towards right.

This change, or development, in phase is calculated with polynomial regression. Regression analysis is a statistical tool used to compute the relationship between values in a set of data. In a second degree polynomial regression, a function describing this relationship is modeled using multiple linear regression, and can be described by the equation

$$Y = \beta_0 + \beta_1 x + \beta_2 x^2, \tag{2.8}$$

where β_n are called the regression coefficients and are constants. Estimation of the coefficients is carried out by the method of least squares. Lastly, the model checks if peaks at different altitudes have roughly equal wavelengths, meaning

$$\lambda_{z_1} \approx \lambda_{z_2},$$

where z_1 and z_2 are two different altitudes. Equal wavelengths at different altitudes, within a margin of error, indicate that the same wave exists across different altitudes. The wavenumber

$$\vec{k_h} = \vec{k_x} + \vec{k_y}, \tag{2.9}$$

is a vector in the horizontal direction, where $\vec{k_x}$ and $\vec{k_y}$ describe vectors $\in \{0,...,2\pi\}$ of length N, where N is the number of sample points, and the vectors represent the directions $\hat{\mathbf{i}}$ and $\hat{\mathbf{j}}$ respectively. This means that

$$|k_h| = \sqrt{k_x(u)^2 + k_y(v)^2}, \tag{2.10}$$

where u and v is the first and second coordinate, respectively, of a point in Fourier space, and $|k_h|$ is the distance from this point to the origin. The wavelength is then found by

$$\lambda_h = \frac{2\pi}{|k_h|} \cdot L, \tag{2.11}$$

where L is the scale length between each point in the Fourier space and real space.

3 Methods

This section describes how input data is processed by the analysis model, in addition to describing the two different modes available in the model.

3.1 Sampling and readying data

The relationship

$$n = \frac{\text{view}}{\text{grid size}} = \text{number of samples} \tag{3.1}$$

gives the number of sample points in each direction, where view is the field of view of the MATS satellite and grid size is the resolution. The view depends on operational mode of the satellite, but is chosen to be $250 \times 250 \times 60$ km during the development of the model. The grid size during development is $5 \times 5 \times 0.2$ km. The three dimensional data gathered will be divided into two dimensional horizontal "slices" at each altitude, and an analysis will be performed on each slice. A discrete Fourier transform (DFT) is performed on the sampled, sliced MATS data. The DFT in two dimensions implemented in the model is of type fast Fourier transform (FFT) and can be defined as

$$A_{kl} = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} a_{mn} \exp\Big\{ -2\pi i \Big(\frac{mk}{M} + \frac{nl}{N} \Big) \Big\}, \tag{3.2}$$
$$k = 0, ..., M-1;\ l = 0, ..., N-1.$$

Here M and N describe a two-dimensional matrix of size $M \times N$ [11]. FFT is an algorithm which calculates the DFT and is optimized with respect to computational speed. FFT is fastest when the length of the transform is a power of two, so the model ensures that this condition is met by finding the nearest power of two larger than the signal length, and using this as the length of the FFT. The transform will be zero padded, meaning that the lengths M and N will be larger than the length of the input data in the model by a factor of 10. Zero padding is a computationally efficient method of interpolating a signal, as zero padding makes the spectrum look smoother. The FFT can also be described in terms of frequency bins, where the size of the frequency bin is inversely proportional to the number of points, including zeroes, transformed. The resolution of the FFT will be affected by using an apodization function, or window function, with a size equal to the original signal's length. Signal resolution is understood to be the ability to distinguish

between peaks. However, zero padding will not change the resolution of the signal, as the resolution is related only to the number of samples, i.e. the length of the signal [9].

Applying a window function is often called windowing, and is done by multiplying the time signal with a finite-length window with an amplitude that changes gradually and smoothly towards zero at the edges. Windowing has several advantages and areas of application to it. The FFT algorithm assumes the signal has infinite extension and is periodic. Of course, a real signal is not infinite, and there is no guarantee that the signal is periodic outside of the sampled part of the time signal. For the FFT, the two endpoints of the signal are interpreted as though they were connected together. If, for example, the measured signal is a sine wave that is not an integer number of periods, the signal will become truncated, since the endpoints are discontinuous. Truncated signals can lead to spectral leakage, but can be avoided by windowing the signal, as can be seen from figure 3.1.

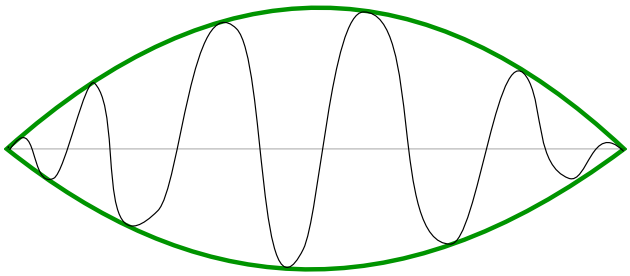

Figure 3.1: An illustration of how applying a window function to a signal affects the signal. The green lines are the apodization function, while the black line illustrates a signal.

The apodization function applied to the input data in the model is the Hamming function. The Hamming function with length N in one dimension is given by

$$w(n) = 0.54 + 0.46 \cos \frac{\pi n}{N-1}, \quad n \, \epsilon \, [0, \; N-1]. \tag{3.3}$$

This function is optimized to minimize the height of the nearest side lobe, but results in a wider main peak. The Hamming window does not reach zero, as can be seen from the above equation, leading to a small discontinuity in the signal. A box-windowed transform is a sinc-function, which has side lobes

that are -13 dB that of the main peak, and which fall off at -6 dB/octave, creating a ripple throughout the transform. While the Hamming window increases the FWHM by a factor of 1.3, it reduces the size of the first side lobe to -43 dB that of the main peak, while the fall off is still -6 dB/octave [12].

The two dimensional Hamming window is produced by finding the square root of the outer product of two vectors containing the window function in x- and y-direction, which can be written as

$$W(x, y) = \sqrt{\vec{u} \otimes \vec{v}}, \tag{3.4}$$

where $\vec{u}$ and $\vec{v}$ is a vector describing the Hamming window in x- and y-direction respectively, and W is the Hamming window in two dimensions.

Since the transform is symmetric, as exemplified in figure 3.2, one can analyze only parts of the data field without loss of information, thus greatly reducing computational cost and time needed. A removal of the second and third quadrant of the transformed data field is performed, leading to only the first and fourth quadrant being analyzed, as these quadrants contain all information needed to perform an analysis of the data. This reduces the number of calculations needed significantly. For example, a data set of size 256×256 contains 65536 data points, while the shrunken data field of 128×256 consists of half the number of data points, namely 32768. A simplified illustration of how a two dimensional data field may be represented in Fourier space, without noise and with two different waves present in the data, is shown in figure 3.2. This illustration is idealized, but highlights the symmetry present when FFT is performed, and shows how removing two neighbouring quadrants still preserves all data needed to identify waves present in the field.

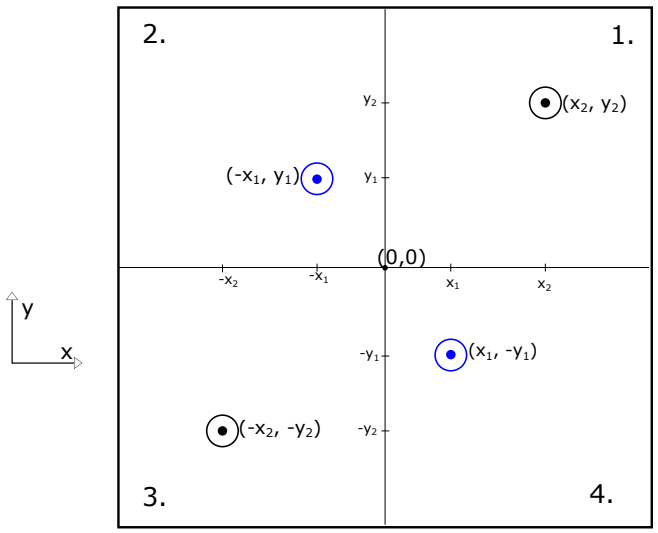

Figure 3.2: Two-dimensional Fourier space, with circles depicting two-dimensional waves. Notice how the blue and black waves are symmetric with respect to the origin. The number in each corner represents the quadrant.

Full width at half maximum (FWHM) is the width of a spectrum curve measured between the points in which the amplitude value is half of the maximum amplitude of the curve. The distance, in number of bins, to the FWHM in Fourier space can be found by the relationship

$$\frac{\Delta f_{FWHM}}{\Delta f_s}, \tag{3.5}$$

where Δf_{FWHM} is the distance to FWHM in units [bins/m] and Δf_s is the sampling distance of the FFT with units [1/m]. Δf_{FWHM} is given by the equation

$$\Delta f_{FWHM} = \frac{1.30}{w_{data}}, \tag{3.6}$$

with w_{data} as the length of the data field of the data sampled, which is also called view. Δf_s is given by the equation

$$\Delta f_s = \frac{\frac{1}{2\Delta x}}{M/2}, \tag{3.7}$$

where Δx is the sampling distance and M is the length of the FFT in either direction. Half width at half maximum (HWHM) is the width from the center of a peak to the side of the peak at an amplitude value of half the maximum amplitude, and is thus half of the FWHM of the peak.

3.2 Filtering data fields and identifying peaks

Digital filtering is the realization of the convolution integral in discrete form [9]. By convention, a filter is represented as $h(t)$. If $x(t)$ is the input signal, and $y(t)$ is the output signal, the filtering procedure can be described as

$$y(t) = x(t) * h(t). \tag{3.8}$$

Since the sampled signal is of finite length N, the filter design approach used in the model is termed a Finite Impulse Response (FIR) filter. A realization of digital filtering is achieved by performing the discrete convolution operation with the sampled input waveform $x(t)$, as seen in equation (3.8). By using the convolution theorem [13], the filtering operation in Fourier space will be

$$\mathcal{F}\{x * h\} = \mathcal{F}\{x\} \cdot \mathcal{F}\{h\}. \tag{3.9}$$

A median filter h_m is applied to the treated data of the first and fourth quadrant. Such a filter runs through all points of the treated data and replaces each point with the median of the neighbouring points, where the number of neighbouring points considered is as a "window" of certain size surrounding the data point in question. This window is symmetric about the data point treated. This filter is used to remove spikes from the data. As windowed data has a FWHM, such spikes can never represent real data, and is therefore always noise, and thus unwanted when analyzing the data set.

An average filter matrix with size equal to that of h_m is made, based on the values of h_m. Average filters, h_a, are used to highlight long-term trends and smooth out short-term fluctuations. A type of average filter, called moving average filter, is used in the model. This filter uses a neighbourhood of points in the same manner as a median filter, calculates the average of these data points and applies this to the data point in the center of the window. Average filters are commonly used to reduce random, white noise, because these filters produce the lowest noise for a given edge sharpness. As the size of the window increases, the noise becomes lower. However, the edges of the signal become less sharp, thus providing a trade-off between sharpness and noise-reduction.

The output $y(n)$ from the filtering operation, including both filters applied, will be

$$y(n) = x(n) - y_{ma}(n), \tag{3.10}$$

where $x(n)$ is the original signal, and $y_{ma}(n)$ is the output after applying the median and then average filter. Negative amplitude values as a result of the subtraction are set to zero. When y_{ma} is subtracted from x, the resulting

peaks will be of smaller amplitude, but the difference in amplitude between the peaks of the signal $y_{ma}(n)$ and the original signal $x(n)$ is greater than the difference between the noise in these signals, leading to the peaks being easier to identify after the subtraction in equation (3.10).

An example of how filtering a noisy signal may change the output is seen in figure 3.3, when a median filter h_m is applied to remove the spikes, and an average filter h_a is used to smooth out noise.

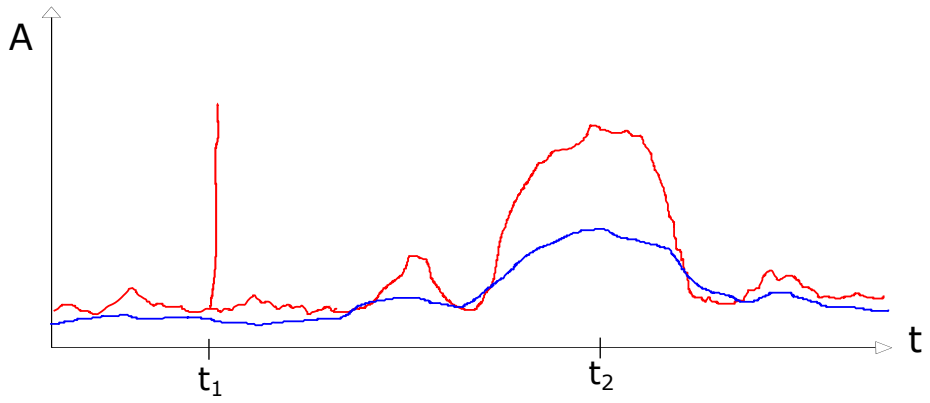

Figure 3.3: Two signals in the time domain. Red represents a noisy signal, and blue represents the same signal, but filtered. Notice how the spike at t_1 is filtered out, and the peak with maximum amplitude at t_2 has a lower amplitude after filtering.

A maximum filter is applied to the filtered data to identify wave peaks. This filter uses a window in the same manner as the prior filters. The filter finds the maximum value of the data points inside the window, and applies this to the data point in the center of the window. The filter matrix is then compared to the input data, and points where the input data is equal to the filter data is returned as local maxima, which are interpreted as wave peaks. But the spike at t_1 in figure 3.3 highlights why using an amplitude threshold to decide whether the data contains wave peaks is not enough. If the model is unable to filter out spikes or other noise, the model could accept such maximums as wave peaks, and thus start analyzing these data points. To avoid such mistakes and separate wave peaks from random peaks in the data, the model needs another prerequisite before accepting data points as

wave peaks.

Since the data is windowed before it is filtered, all peaks containing wave structures that are present in the data have the same width at half maximum, namely the FWHM, described by equation (3.5). This trait is used to separate wave peaks and noise, thus becoming the second prerequisite the model uses to identify wave peaks. After the data is processed and filtered, the model will try to calculate wave properties in the peaks confirmed to represent wave structures.

3.3 Identification of wave properties

The model must be able to find several different wave properties before being able to decide whether the input data contains wave structures, as stated in section 2.5. When filtering the transformed input data, the amplitude of the data, described by equation (2.5), is used. The abs()-function in Python returns the magnitude when the number is complex, and is thus used to find the amplitude at each data point in the processed data [14].

After the data is filtered by the methods described in section 3.2, the phase of the wave peaks are found. Since the phase of a complex number is defined only in the interval $[-\pi, \pi]$, the phase is unwrapped to show the development of the phase across the peaks. This means that discontinuities larger than π between two points in the phase data are changed to their 2π complement [15]. The phase is plotted against altitudes, so that the phase of waves developing across several altitudes are easily visible to the user. The phase is further analyzed with linear regression, described by equation (2.8). This gives both visual and mathematical representation of how the phase of the identified waves develops with altitude.

For each peak assumed to be a wave, the wavelength at the top of the peak is found by the method shown in equation (2.11). The wavelengths are plotted with a margin of error of $\pm$HWHM, to be compared at each altitude. As stated in section 2.5, this gives the possibility to see whether a wave exists across multiple altitudes.

When the analysis is completed, one is able to decide whether the analyzed data contains wave structures or not, based on the output of the model. Waves identified in the data set at different altitudes, with phases developing from one altitude to the next, and with wavelength approximately equal at each altitude, within a margin of error, indicate the same wave propagates across multiple altitudes.

3.4 Synthetic data

When the model for analyzing data was developed, a synthetic wave was used as input. This wave was defined as

$$\Psi_z(x, y) = A \cos\left(\frac{2\pi x}{\lambda_x} + \frac{2\pi y}{\lambda_y} + \frac{2\pi z}{\lambda_z}\right), \tag{3.11}$$

where A is the amplitude, $\lambda_x = \lambda_y$ is the wavelength in the horizontal directions, λ_z is the vertical wavelength and the phase constant of the wave, and the wave is monochromatic. Such a wave is demonstrated in figure 3.4. The values chosen to best simulate a gravity wave are presented in table 3.1.

Table 3.1: The wave parameters and their values for the wave used during the development of the model. The values are chosen to best represent a gravity wave.

Parameter	Sign	Value	Unit
Amplitude	A	3	Rayleigh [R]
Hor. wavelength	$\lambda_{x,y}$	28284	meters [m]
Ver. wavelength	λ_z	35000	meters [m]

The horizontal wavelength was decided using data from a study on wave characteristics from mesospheric gravity waves observed near equatorial and low-middle latitude stations [16], and the amplitude is typical for wave parameters in oxygen airglow.

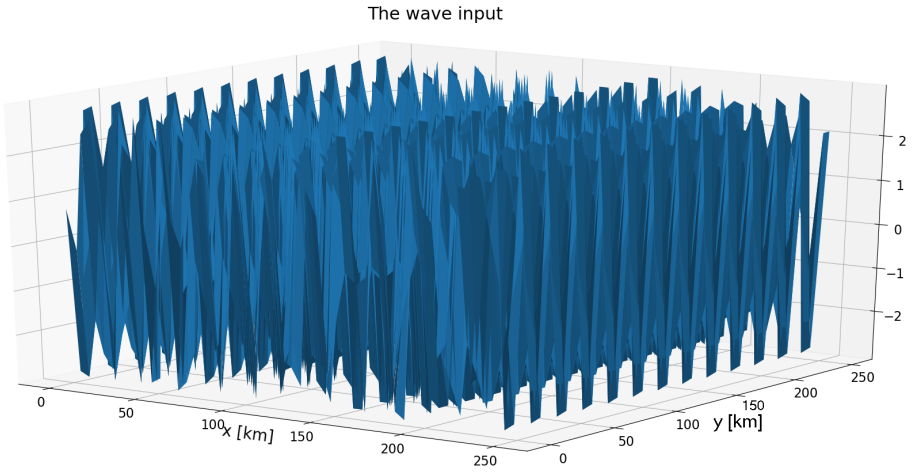

Figure 3.4: A synthetic wave provided by the model, with values as in table 3.1.

When developing the model, a synthetic wave with known values and parameters removes a possible uncertainty, which is the input of the model. Having information about the input makes the interpretation of the model's output more manageable, as one can anticipate the output and thus decide if the model's analysis of the data is correct. Lastly, noise is added to the input wave to test if the model manages to filter out unwanted data. The noise is random samples from a normal distribution with mean = 0 and standard deviation = 1.

The grid size, or sampling distance, of the synthetic data is found from equation (3.1) and is presented in table 3.2. The values of view and grid size are both in horizontal direction, and would differ if analyzing in vertical direction is preferred. From the Nyquist sampling theorem, we know that a grid size of 5 km is sufficient for sampling a wave with a wavelength known to be 20 km [17]. Zero padding and ensuring the transform is a power of two combined renders the length of the FFT to be 512 in both horizontal dimensions. This means that the data field analyzed is of size 256×512 in x- and y-direction respectively. Since the wave is monochromatic, the phase will be constant across found peaks, and is easily identified. However, analyzing a monochromatic wave is not representative for the expected input data of the MATS mission. The model thus needs to be tested with more realistic input data.

Table 3.2: The view, grid size and number of sample points n used when analyzing synthetic data.

Parameter	Value	Unit
view	250	kilometers [km]
grid size	5	kilometers [km]
n	50	integer

3.5 Data from forward model

To test the model with more realistic input data, the model examined data provided from a forward model developed by Anqi Li of Chalmers University of Technology [18].

The data provided by the forward model is a three dimensional matrix of nightglow volume emission rate in units [photons cm^{-3} s^{-1}]. The number of samples in the 3D matrix is $145 \times 200 \times 160$ in x-, y- and z-direction respectively. The view is $400 \times 1440 \times 80$ km in x-, y- and z-direction respectively. The grid resolution, size of data field and number of sample points needed to analyze the data are presented in table 3.3. Both the grid size and field of view is in x-direction in the horizontal plane, and thus the number of samples would differ if y- or z-direction is preferred. By the Nyquist sample rate, waves with wavelength less than approximately 5 km will not be sufficiently sampled [17].

Table 3.3: The view, grid size and number of sample points n used in the forward model.

Parameter	Value	Unit
view	400	kilometers [km]
grid size	2.759	kilometers [km]
n	145	integer

The model is made to analyze the data to examine whether it is able to identify wave properties in data resembling the MATS data fields gathered by the satellite. Zero padding and ensuring the transform is a power of two combined renders the length of the FFT to be 2048 in both horizontal dimensions. This means that the data field analyzed at each altitude is of size 1024×2048 in x- and y-direction respectively. According to the parameters provided with the model, the data set contains waves with horizontal wavelength 53 km, and a background wind in horizontal direction.

4 Results

4.1 Results from data treatment

The technique developed for analyzing MATS data is a model developed in the programming language Python 3.6. The code of the model is attached in appendix A. Synthetic data was used to develop a method for treating and identifying data in horizontal direction. When analyzing at several altitudes, meaning the input data is three dimensional, this horizontal analysis is performed at each altitude. The input data is transformed and analyzed, and the data returned from each horizontal analysis is the prospective wave characteristics within the data. The result of the windowing of a wave can be seen in figure 4.1. This wave has the same wave parameters as the wave in figure 3.4, but has been windowed as described in section 3.1.

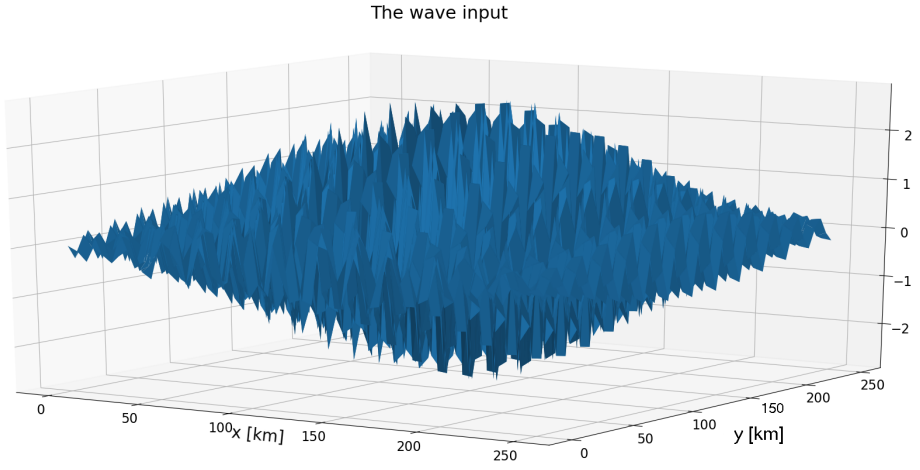

Figure 4.1: A wave with horizontal wavelength 20 km, after it has been windowed.

Figure 4.2 shows a magnitude plot of the transformed wave provided by the model and shown in figure 3.4. The plot shown in this figure is from when the wave has been sampled and windowed before being transformed, but not filtered after the transformation. The side lobes, resulting from the Hamming window, have approximately equal amplitudes, except for the distorted peak on the right side of the main peak. The amplitude of the side lobes is approximately 3/8 of the amplitude of the main peak, located at $k_x \approx \pi/2,\ k_y \approx 2\pi/3$.

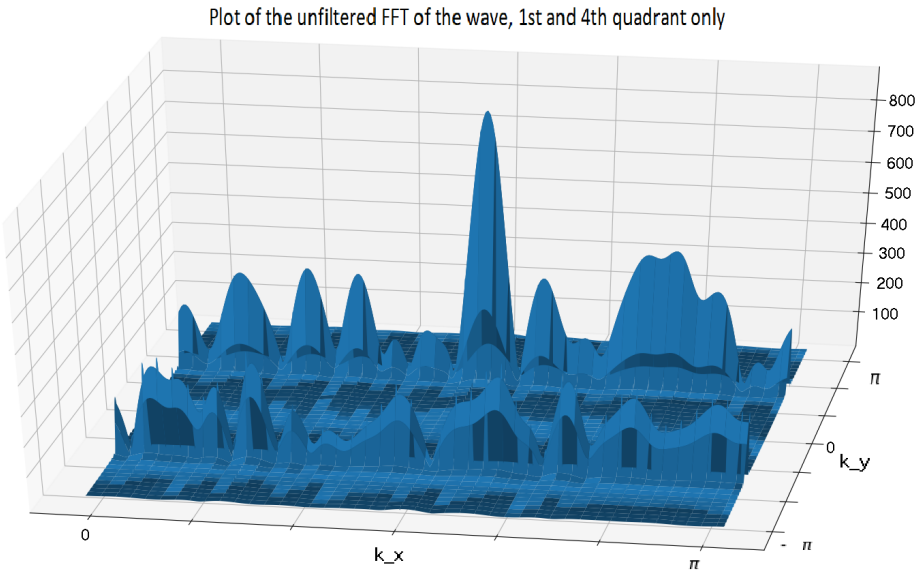

Figure 4.2: The resulting amplitude plot after the wave is transformed to Fourier space. The wave is windowed, but unfiltered.

Figure 4.3 shows the effect filtering as described in section 3.2 has on the transformed signal. This is the same signal as in figure 4.2. The amplitude of all peaks is reduced compared to before filtering. The amplitude of the side lobes is approximately 1/3 of the amplitude of the peak located at $k_x \approx \pi/2,\ k_y \approx 2\pi/3$, except for the spike at the bottom left corner. The red cross shows the location of a wave peak identified during the analysis.

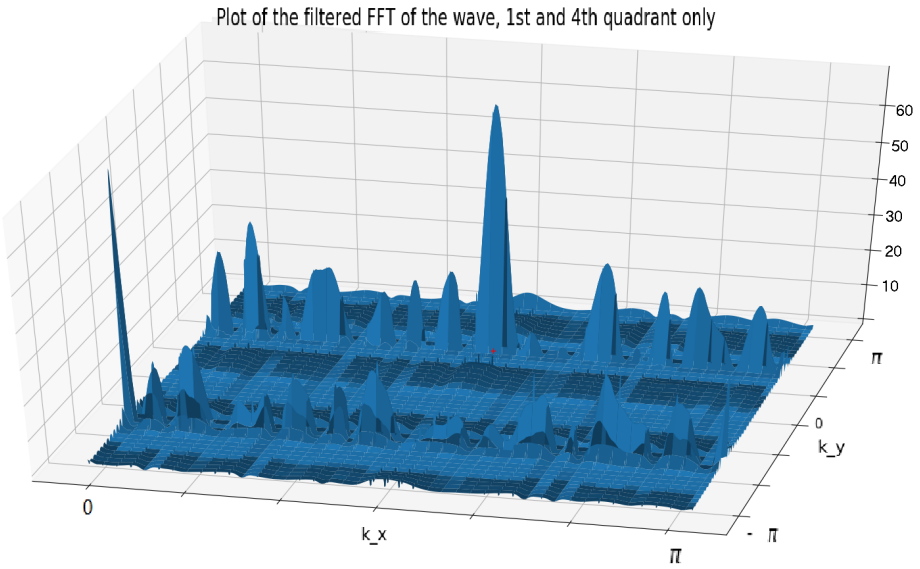

Figure 4.3: The resulting amplitude plot after the wave is transformed and filtered as described in section 3.2. The red cross mark the location of a peak identified as a wave peak during the analysis.

Two other methods for implementing the filters were also tested. The second method was by subtracting the average filter y_a from the median filter y_m, to see whether this successfully removed spikes and smoothed out the noise. The result is shown in figure 4.4. The noise peaks and side lobes have small amplitudes compared to the large peak. However, the top of the main peak is distorted and full of spikes, instead of being a smooth peak. The side lobes and noise peaks are small in amplitude, but also they have spikes on top of their peaks. The analysis was not able to find any wave peaks when this implementation was used.

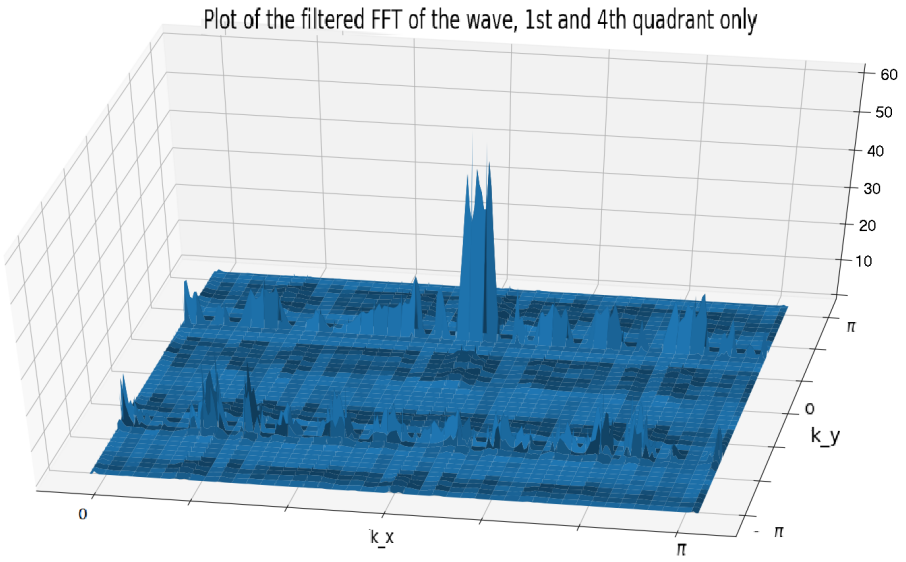

Figure 4.4: The resulting amplitude plot after the wave is transformed and filtered with $y_m - y_a$, where y_m is the output from the median filter and y_a is the output from the average filter.

Also the third method for implementing the filters was with subtraction, as the filter in figure 4.3. However, the implementation varied slightly. The wave is filtered with $y_m - y_a$, where y_m is the output from the median filter and y_a is the output from the average filter, and the filter is run through point by point. The difference in amplitude of these two outputs is found, and if the difference between the two is small, the amplitude value from the original signal is kept, but if the difference is large, the value from the filtered array is kept. The amplitude of the side lobes and noise peaks is approximately 3/10 of the amplitude of the largest peak. The resulting output after this implementation is shown in figure 4.5.

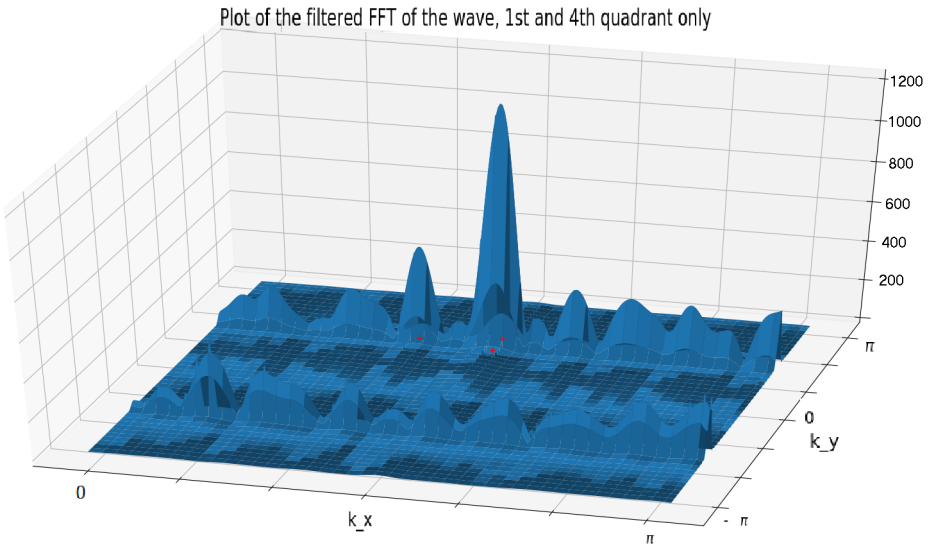

Figure 4.5: The resulting amplitude plot after the wave is transformed and filtered. The filter compares two arrays and keeps the most fitting values from both arrays at different coordinates. The red crosses mark the location of points identified as wave peaks during the analysis.

4.2 Results with synthetic data

The FWHM of the synthetic wave is calculated as described in equations (3.5) - (3.7), with values from table 3.2 and the length of the FFT, which is 256 in x-direction. The result is a FWHM of 13.312 bins, and thus a HWHM of 6.656 bins. The results from the analysis of synthetic data are from input waves with two different horizontal wavelengths, namely 20 km and 53 km. 20 km is the result when the values from table 3.1 is used, with the equation

$$\frac{1}{\lambda_h} = \sqrt{\frac{1}{\lambda_x^2} + \frac{1}{\lambda_y^2}},$$

derived from equation (2.9). The wavelength 53 km is the horizontal wavelength of the input wave in the forward model, according to the parameters provided in the model. Using the above equation gives λ_x and λ_y to be $\lambda_{x,y} = 74953$ meter. The waves with horizontal wavelength 20 km have a vertical wavelength as in table 3.1, while the waves with horizontal wavelength 53 km have a vertical wavelength of 20 km.

4.2.1 Horizontal analysis

Figure 4.6 shows an amplitude plot of the wave with wavelength 20 km, and one peak is identified as a wave peak in the plot. In figure 4.7 the wave has a wavelength of 53 km, and the amplitude plot shows that the model identified one peak as a wave peak.

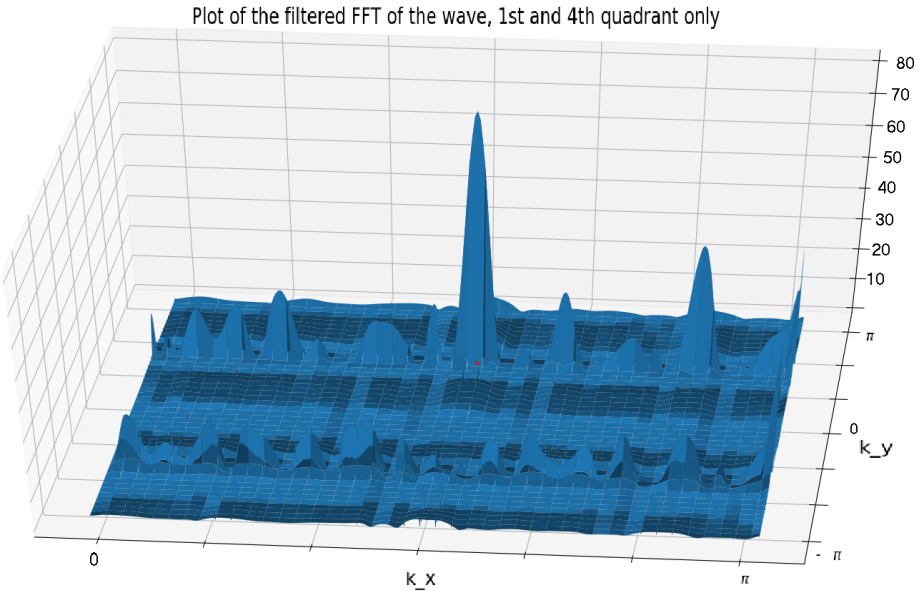

Figure 4.6: Amplitude plot with an input wave with horizontal wavelength 20 km. The red cross at the bottom of the peak marks center of the peak identified as a wave peak during analysis. The small peaks are either due to noise or side lobes from the FFT.

The waves present in figure 4.6 and 4.7 have both been processed as described in sections 3.1 and 3.2, and their wave characteristics were calculated as described in section 3.3 .

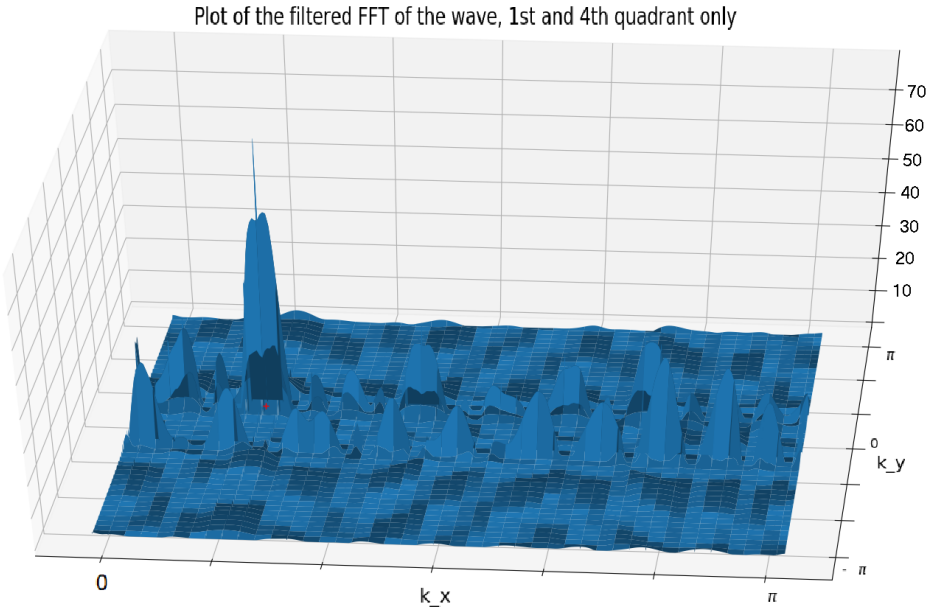

Figure 4.7: Amplitude plot with an input wave with horizontal wavelength 53 km. The red cross at the bottom of the peak marks center of the peak identified as a wave peak during analysis. The small peaks are either due to noise or side lobes from the FFT.

4.2.2 Vertical analysis

Data with a synthetic wave was also used to test the model's ability to analyze data sets with data from several altitudes. To best simulate the forward model, the altitudes chosen were the same as in this model, namely altitudes ranging from 70 km to 150 km. The output from the model is a plot of the phase of waves identified, with a second degree polynomial regression of the development of this phase, and a plot of the wavelengths of the waves identified, with a margin of error of ±HWHM. Thus, the regression is of the form

$$y(x) = ax^2 + bx + c, \tag{4.1}$$

where a, b and c are all constants, as described in section 2.5.

Figure 4.8 and 4.9 show the output plots when the data set analyzed includes a wave with horizontal wavelength of 20 km at each altitude. The regression of the phase plot in figure 4.8 gives the equation

$$y = 0.18x - 12.14. \tag{4.2}$$

Note that the regression is not in terms of π, as the y-axis in figure 4.8 is. Also, the regression starts at an altitude of 0, while the plot starts at an altitude of 70 km.

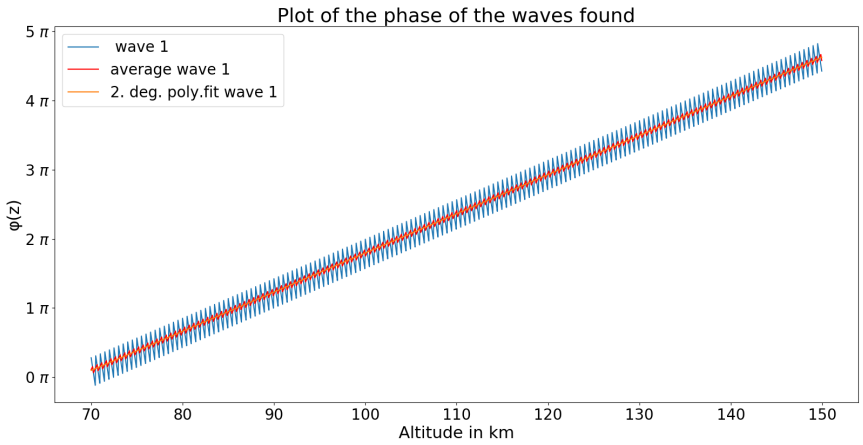

Figure 4.8: The phase of an identified wave within the data set plotted with respect to altitude. The plot shows one wave, with its corresponding average and a second degree polynomial fit of the average of the wave.

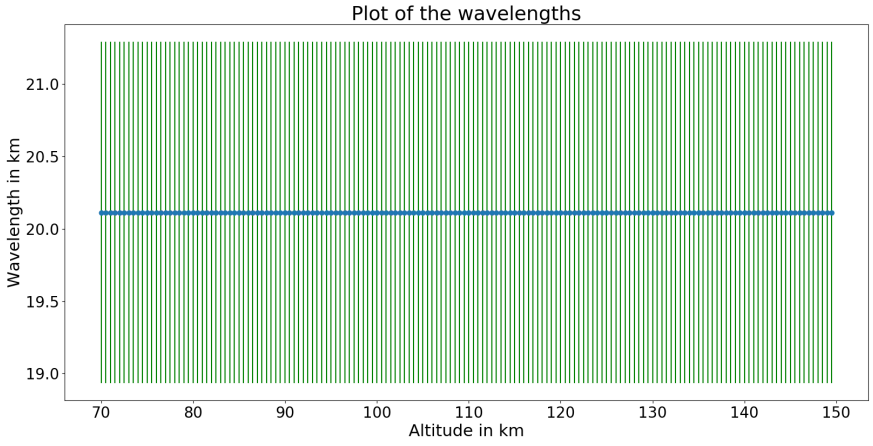

Figure 4.9: The horizontal wavelength of the waves found at each altitude during analysis. The waves had a horizontal wavelength of 20 km at each altitude according to this plot. The blue dots is the horizontal wavelength of the waves, while the green lines is the HWHM of the waves peak.

Figure 4.10 and 4.11 show the output from the model developed when the input at each altitude was a wave with horizontal wavelength 53 km. The regression in figure 4.8 gives the equation

$$y = 0.31x - 22.05. \tag{4.3}$$

Also here, the regression is not in terms of π, as in the figure, and it starts at an altitude of 0, not 70 km as in the figure.

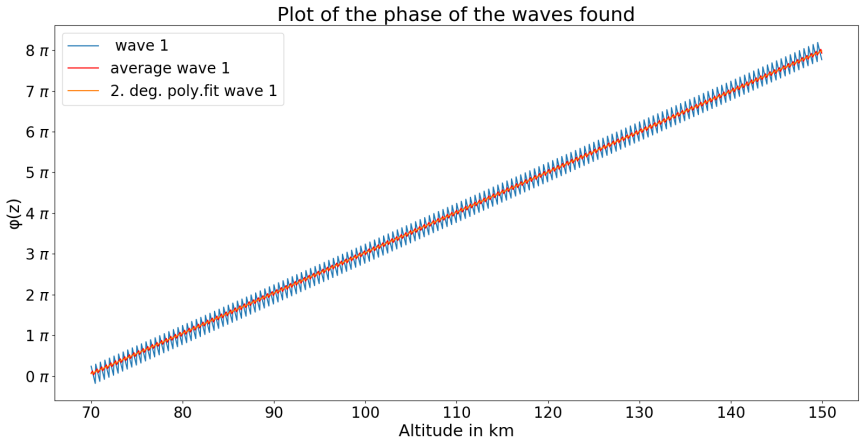

Figure 4.10: The phase of an identified wave within the data set, plotted with respect to altitude. The plot shows one wave, with its corresponding average and a second degree polynomial fit of the average of the wave.

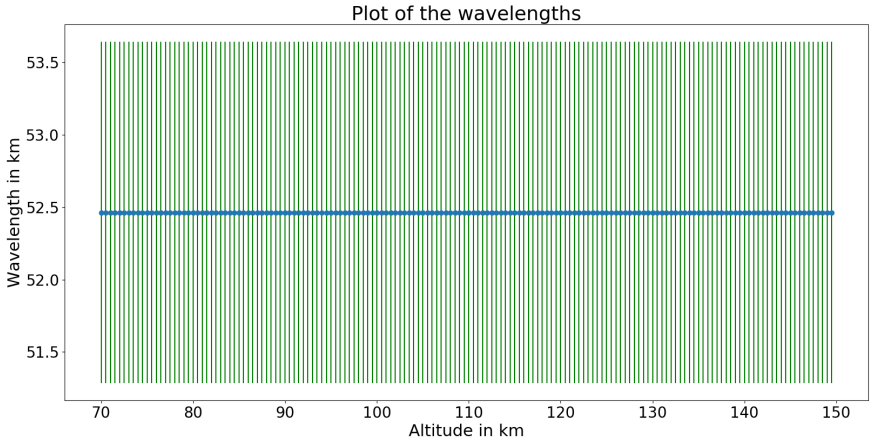

Figure 4.11: The horizontal wavelength of the waves found at each altitude during analysis. The waves found at each altitude have a horizontal wavelength of 53 km according to this plot. The blue dots is the horizontal wavelength of the waves, while the green lines is the HWHM of the waves peak.

However, with the same input data as above, the model sometimes identifies several waves within the data set. The resulting output in this case can be seen in figure 4.12 and 4.13. Here the input at each altitude was a wave with horizontal wavelength 53 km. The result from the regression analysis in figure 4.12 is

$$y_1 = 0.31x - 21.89 \tag{4.4}$$

and

$$y_2 = -0.89x + 61.52, \tag{4.5}$$

where y_1 and y_2 describe wave 1 and wave 2 respectively.

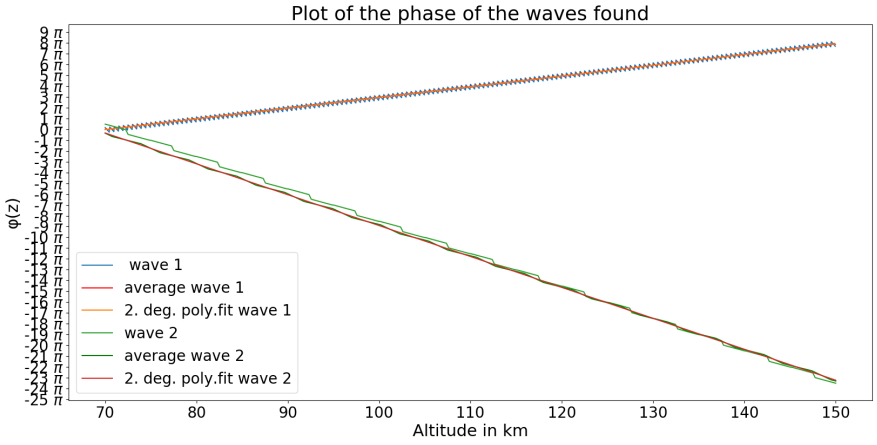

Figure 4.12: Phase plotted with respect to altitude of the waves identified within the data set. The plot shows two waves, with their corresponding average and a second degree polynomial fit of the average of the waves.

As can be seen from figure 4.13, the model identifies one wave with horizontal wavelength $\lambda_h \approx 53$ km, and another wave with horizontal wavelength $\lambda_h \approx 18$ km. The second wave is not present at all altitudes.

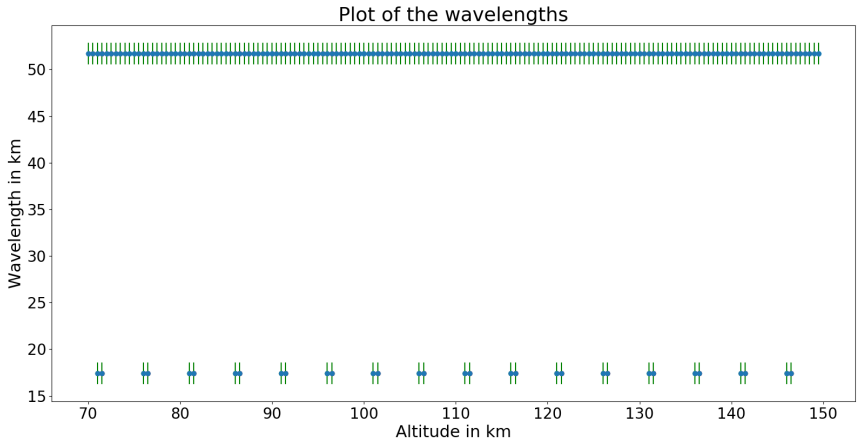

Figure 4.13: The horizontal wavelengths of the waves identified within the data set. One wave is present at all altitudes and has a horizontal wavelength of approximately 53 km, while another wave is present at most altitudes and has a horizontal wavelength of approximately 18 km. The blue dots is the horizontal wavelength of the waves, while the green lines is the HWHM of the waves peak.

4.3 Results from forward model

The model also analyzed data from the forward model in section 3.5. The FWHM was calculated with the values from table 3.2 and with the length of the FFT as 1024 in x-direction. The result was a FWHM of 18.36 bins, and thus a HWHM of 9.18 bins. The resulting output from this analysis can be seen in figure 4.14 and 4.15. The regression analysis of the phases in figure 4.14 gives the following equations:

$$y_1 = -0.01x^2 + 0.52x - 7.69 \tag{4.6}$$

and

$$y_2 = -0.06x^2 + 9.25x - 350.84. \tag{4.7}$$

As earlier, note that the regression is not in terms of π, as the y-axis in figure 4.14 is, and that the altitudes in the figure start at 70 km.

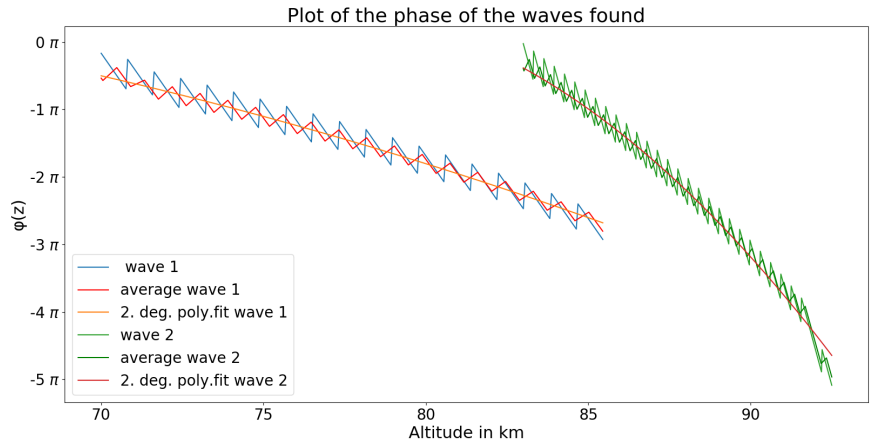

Figure 4.14: The phase of the waves identified within the data set from the forward model. The plot shows two waves, with their corresponding average and a second degree polynomial fit of the average of the waves.

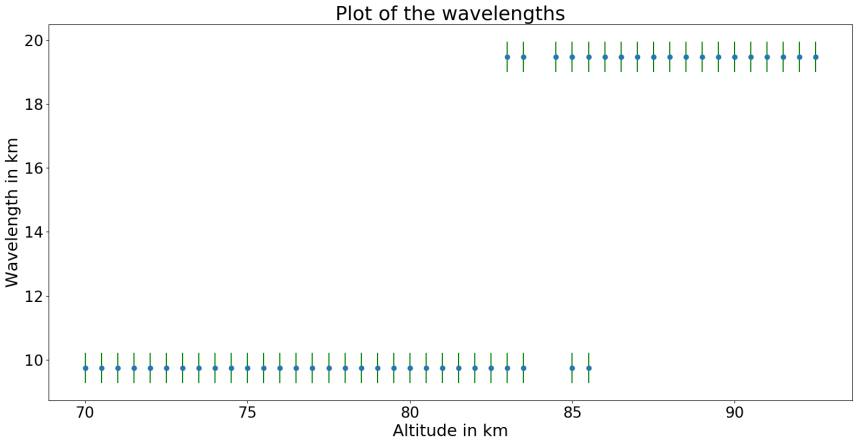

Figure 4.15: The wavelength of the waves identified within the data set from the forward model. The blue dots is the horizontal wavelength of the waves, while the green lines is the HWHM of the waves peak.

As can be seen from the above figures, the model identifies two waves, y_1 and y_2. y_1 is a wave with horizontal wavelength $\lambda_h \approx 9$ km, and it exists on altitudes ranging from 70 to 85.5 km. The second wave, y_2, is a wave with horizontal wavelength $\lambda_h \approx 19$ km, and it propagates from an altitude of 83 km to an altitude of 92.5 km, according to the results from the analysis.

5 Discussion

If the model developed is to be considered successful, it should be able to find and calculate the wave characteristics of verified waves within the data set. The model should also be able to filter out most peaks that are not a result of gravity waves in the data set, to reduce data analyzed and simplify the output data of the model.

5.1 Data treatment

Windowing the data greatly reduced the side lobes of the main peak in Fourier space, and is thus a necessary step in treating the data. The result of this windowing process, as seen in figure 4.1, is in accordance with the expected result, illustrated in figure 3.1. The Hamming window seems to be a good window function to use, and results in a large and wide main peak, which is identified with help of the FWHM of the Hamming window. The zero padding of the signal is a computationally efficient way of interpolating the signal, and could be even higher than the current zero padding of 10 times the signal's length. However, a longer zero padding means a larger FFT length, and the trade off between resolution and computing power needs to be considered, and a zero padding of 10 times the signal's length is therefore kept. Analyzing only the first and fourth quadrant greatly reduces the number of points which need to be analyzed, while also removing the potential problem occurring since every wave produces two waves in Fourier space which are symmetric with respect to the origin. If the whole data field, with all four quadrants, was to be analyzed, the model would identify each wave twice, interpreting it as if two waves are present in the data. Thus, analyzing only the first and fourth quadrant omits this problem and is another advantage of performing this reduction of the data field.

When developing a method for filtering the data, several methods for implementing the filters were tested and discarded. Figure 4.3 shows the implementation used in the model. The main peak is larger than the side lobes and noise by a factor of approximately 3, and the model was able to identify this peak as a wave peak. The two other methods of implementing the filters in section 4.1 also resulted in a distinct main peak. However, the method used in figure 4.5 led to the model consistently accepting noise peaks and side lobes as peaks resulting from a wave. Figure 4.4 shows that this method for implementing the filters leads to a distorted peak full of spikes, and the model thus was not able to identify any peaks as wave peaks. The reason

why the implementation used in figure 4.5 led to the model accepting random peaks as wave peaks is unknown to the author of this thesis. However, the author is led to believe that this method of implementing the filters holds the most potential for reducing noise compared to wave peaks, as the main peak is smooth and clean of spikes, and the ratio between the main peak and other peaks was the highest of all the filter implementations tested.

5.2 Results with synthetic data

The data returned at each altitude was reduced to only being the wavelength and phase of the peaks identified as wave peaks. If the data set would be analyzed as a 3D matrix, meaning the treated and transformed data at each altitude would be returned, the model would need more computing power. When a data set as the one from the forward model in section 3.5 is analyzed, the model would have $1024 \times 2048 \times 160$ data points to analyze if the data is treated and transformed, and then returned to be analyzed. This means less data needs to be stored when the analysis is performed at each altitude, while all the necessary data still is returned. Figure 4.6 and 4.7 show how the model successfully identifies the right peak as a wave peak when waves with different horizontal wavelengths are present in the data set analyzed. These plots are in accordance with the theory in section 2.5. Figure 4.7 shows that the model is able to identify a wave peak despite the large spike at the center of the wave peak not being filtered out.

However, the model sometimes identifies peaks as wave peaks even though they are a result of noise or side lobes of the main peak. This indicates that the identification of wave peaks is not good enough. This may be due to either the data set not being sufficiently filtered, or the accuracy of the second prerequisite, namely the check of FWHM, is too poor. In section 4.2 the FWHM of the waves within the synthetic data set is stated as FHWM = 13.312 bins, and the HWHM is stated as HWHM = 6.656 bins. However, when in Fourier space, the distance is discrete, meaning the HWHM needs to be in integers, and thus the HWHM = 7 bins. This rounding introduces an uncertainty in the calculation of HWHM that may lead to mistakes when identifying wave peaks. The errors are most likely due to a combination of the above procedures. In the horizontal analysis, the phase of the accepted peaks is returned along with the wavelength of these peaks. These values are then compared with the returned values from other altitudes in a vertical analysis, to see whether the identified waves propagate to other altitudes.

Figure 4.9, 4.11 and 4.13 show that the model can successfully calculate the wavelengths of the peaks identified within the data set. The wavelengths

plotted in figure 4.9 indicate that a wave with horizontal wavelength 20 km was identified at each altitude, and figure 4.11 indicates that a wave with horizontal wavelength 53 km was identified at each altitude. These figures, with a constant horizontal wavelength with respect to altitude are both known to be correct. The two waves identified and plotted in figure 4.13 show that the model is able to find the wavelengths of each wave when there are multiple waves identified within the data set. The figure shows a wave present at all altitudes with a horizontal wavelength of approximately 53 km. However, it also shows that the model identified a wave present at most altitudes, with a horizontal wavelength of approximately 18 km. If the wavelength calculated of the latter wave is correct is not known, however, such a wave was not present in the data set the model was tasked to analyze.

Figure 4.8, 4.10 and 4.12 show the phase of the identified waves. The slope of the phase in figure 4.8 and figure 4.10 shows that the phase increases with 2π every 35 km and 20 km respectively. This is consistent with the vertical wavelength of each of the waves present within the input data, and thus verifies the model's ability to find the phase of the identified waves. In figure 4.12, the slope of the phase increases with 2π every 20 km in wave 1, which is in accordance with the phase in figure 4.10, which has the same input data. However, the phase of wave 2 is hard to verify whether is correct or not.

The figures in section 4.2.2 indicate that the model is able to successfully calculate the wavelength and phase across the peak of identified wave peaks, thus showing that the techniques for identifying these wave characteristics are sufficient. However, the model often accepts non-wave peaks as peaks resulting from a wave, adding uncertainty to the output data of the model.

5.3 Results with data from the forward model

The resulting output after the model analyzed the data set from the forward model shows that the wave identifies two waves, as can be seen in figure 4.14 and 4.15. However, this is wrong according to the parameters provided with the model, as there should only be one wave present, with a horizontal wavelength of 53 km. With the inaccuracies in the model analysis, which are evident from the results of the synthetic data, it is difficult to conclude with why these errors occur in the results. The model consistently identifies and returns the wave characteristics shown in figure 4.14 and 4.15, which indicates that the errors are not in the identification of peaks. However, the results from the synthetic data show that the model is able to correctly calculate the wave characteristics of the accepted wave peaks. This means

that identifying the cause of the errors is not possible in the current iteration of the model. One thing to note, which can be seen from the regression analysis of the phase of the waves in equation (4.6) and (4.7), is that the slope of the phases curves slightly. This is also visible in wave 2 in 4.14, and is a result of background wind shifting the waves within the data set. A steeper curve indicates higher wind velocities, which means that the curve can be used to measure background wind within a data set.

5.4 Further work

Considering that the workload of a person using this model can be greatly reduced if the model is more successful at filtering out noise and other peaks in the data, the process of identifying wave peaks should be addressed when further developing this model. A possible way of doing this is by working with the last filter implementation, as this is the implementation that should hold the most potential. There may also be a number of other filters suitable for filtering data such as the expected MATS data, and deciding whether the filters used in this model are the most efficient could be further worked with. Investigating the effects of the edge handling by the filters is also possible, but this should not be a huge problem in the current iteration of the model.

The current regression analysis of the phase could be improved upon, as the current implementation with a second degree polynomial regression is only used to describe the phase, not analyze it. Also, the plotting of the phase with respect to altitude is manually corrected, and thus needs to be addressed.

Lastly, an important improvement to the model would be to develop how the model searches through wave peaks within the data. In the current iteration, the model uses a binary search with a predetermined number of peaks as a goal and stops when the set number of peaks wanted is met. However, since the number of waves present within the data set is not known in beforehand, this method of searching for peaks is not ideal. A better way of searching through the data would be to find the peak of largest amplitude and store the information needed of this peak, then remove the peak from the data set. Continue to search through the data set and remove peaks iteratively until either the root mean square is reached or a large number of peaks are found at once, meaning the noise peaks and side lobes are reached. This method ensures that all peaks containing wave characteristics are found within the data set.

6 Conclusion

In this thesis an analysis technique for identification of wave structures in MATS data was developed. This resulted in a model which takes three dimensional data fields as input and analyze the data at each altitude. The model treats and filters the data, and returns wave characteristics of waves present within the data set. The results presented in this thesis show that the model often mistakes noise and side lobes for waves, leading to an output not consistent with the expected results when synthetic data is used. However, the model was able to find the wave characteristics of data accepted as wave peaks, leading to the conclusion that the problems and inaccuracies in the output are from the identification of waves, and not from the calculations of the wave characteristics. Thus, when further working with this model, the identification of gravity waves in noisy data sets needs to be improved. If the model is able to consistently separate wave peaks from noise, the model would be able to return the wave characteristics needed to decide whether gravity waves exist within the analyzed data. Another possible improvement would be to refine the method used when searching for wave peaks, and to develop a method for searching through data without setting prerequisites would be preferable.

The model developed during this project generally seems like a good approach for recognizing and analyzing wave structures within data sets. With further development, the model will hopefully be capable of being used during the MATS mission to successfully analyze data gathered by the satellite. If successful, this mission may further the development of global climate models and weather forecast models, both of which will be important in a possible future with more frequent extreme weather and climate change.

Bibliography

[1] D. G. Andrews, *An introduction to atmospheric physics, 2nd ed.* The Edinburgh Building, Cambridge CB2 8RU, UK: Cambridge university press, 2010.

[2] J. Picone, A. Hedin, D. P. Drob, and A. Aikin, "Nrlmsise-00 empirical model of the atmosphere: Statistical comparisons and scientific issues," *Journal of Geophysical Research: Space Physics*, vol. 107, no. A12, 2002.

[3] D. C. Fritts and M. J. Alexander, "Gravity wave dynamics and effects in the middle atmosphere," *Reviews of geophysics*, vol. 41, no. 1, 2003.

[4] R. B. Smith, A. D. Nugent, C. G. Kruse, D. C. Fritts, J. D. Doyle, S. D. Eckermann, M. J. Taylor, A. Dörnbrack, M. Uddstrom, W. Cooper, *et al.*, "Stratospheric gravity wave fluxes and scales during deepwave," *Journal of the Atmospheric Sciences*, vol. 73, no. 7, pp. 2851–2869, 2016.

[5] S. U. Department of meteorology, "Mats - the next swedish research satellite."

[6] J. Stegman, *Spectroscopic and kinetic studies of atmospheric oxygen emissions*. PhD thesis, Stockholm University, 1991.

[7] NASA, "Astronaut photo of airglow," 2011.

[8] J. W. Goodman, *Introduction to Fourier Optics*. McGraw-Hill Book Company, 1968.

[9] E. O. Brigham and E. O. Brigham, *The fast Fourier transform and its applications*, vol. 1. prentice Hall Englewood Cliffs, NJ, 1988.

[10] "Euler formulas. encyclopedia of mathematics," 2014.

[11] SciPy, "Discrete fourier transform (numpy.fft)," 2018.

[12] F. J. Harris, "On the use of windows for harmonic analysis with the discrete fourier transform," *Proceedings of the IEEE*, vol. 66, pp. 51–83, Jan 1978.

[13] E. W. Weisstein, "Convolution theorem."

[14] "Built-in functions - python 3.6.5 documentation," 2018.

[15] SciPy, "numpy.unwrap," 2018.

[16] C. Wrasse, T. Nakamura, H. Takahashi, A. Medeiros, M. J. Taylor, D. Gobbi, C. Denardini, J. Fechine, R. Buriti, A. Salatun, *et al.*, "Mesospheric gravity waves observed near equatorial and low middle latitude stations: wave characteristics and reverse ray tracing results," in *Annales Geophysicae*, vol. 24, pp. 3229–3240, 2006.

[17] J. G. Proakis and D. G. Manolakis, *Digital signal processing, fourth edition*. Pearson Education, 2014.

[18] A. Li, "A 3d-model for o2 airglow perturbations induced by gravity waves in the upper mesosphere," Master's thesis, 2017. 50.

A Code Listing

main.py

```
# Import the necessary functions and packages
from generatehorizontal import my_inputwave, transform,
    windowing  # , makeSpectrum
from analysis_functions import find_phase, amp_filtering,
    find_quadrants, binary_search, find_wavelengths,
    peakcomparing
from plotting import plotphase, plotamplitude, plotwave,
    plotwavelengths, plotphase_onealt  # , plotthewave
from inputdata import windowing3d, syntdata3d, loaddata
from analysis_altitude import analyze_each_alt
import numpy as np

# set mode either to True if you want to use a dataset in 3D,
    or False if you want a wave generated
# from this model, but only analyzed at one altitude
mode = True

if mode:
    # Use 3D dataset - define data, sampling dist and
        datalength, x and y
    print('This is a three-dimensional data set')
    # The next two lines gives input data from a model by
        Anqi Li of Chalmers university
    # wave, datalengthy, datalengthx, sampling_pointsy,
        sampling_pointsx, sampling_distx_int, sampling_distx
        \
    #    = loaddata()
    # The next line gives synthetic input data in 3
        dimensions
    wave, xx, yy, datalengthx, sampling_distx_int, samplepoints
        = syntdata3d()

    # The code below run an analysis in z_n, where n is the
        number of altitudes

```

```
24      # Windowing the data, before transforming it
25      windowed_data, window_width = windowing3d(wave,
        ↪ datalengthx, sampling_distx_int)  # window width in x
        ↪ direction
26
27      # analysis at each altitude performs FFT, reducing to two
        ↪ quadrants, filtering, detecting peaks and returning
        ↪ the
28      # phase and wavelengths of this data
29      number_of_peaks = 20  # Maximum number of peaks the model
        ↪ will look for
30      phasedata, wavelengths, average, lengthwavepeaks,
        ↪ lenwindow\
31          = analyze_each_alt(windowed_data, number_of_peaks,
            ↪ window_width, sampling_distx_int)
32
33      if len(phasedata) > 0:
34          plotphase(phasedata, average)
35          plotwavelengths(wavelengths, lenwindow)
36      else:
37          print('No waves were found')
38  else:
39      # Synthetic data analyzed at one altitude.
40      print('This is from a synthetic wave, in horizontal plane')
41      wave, x, y, datalength, sampling_dist, samplepoints =
        ↪ my_inputwave()  # generating data
42      # windowing the wave
43      windowed_wave, window_width = windowing(wave, datalength,
        ↪ sampling_dist)
44
45      # F, xf, yf = makeSpectrum(windowed_wave, sampling_dist,
        ↪ sampling_dist)  # only used to examine other FFT
        ↪ functions
46      window_func = transform(windowed_wave)  # transforming the
        ↪ wave to Fourier space
47      print('the shape of the Fourier transform is ' +
        ↪ str(np.shape(window_func)))
48
49      # After FFT, using only 1. and 4. quadrant because of
        ↪ symmetry
50      quadrant = find_quadrants(window_func)
```

```
51      print('window width: ' + str(window_width))
52      filtered_wave = amp_filtering(quadrant)  # performing a
        ↪ filtering procedure
53      print(np.shape(filtered_wave))
54      number_of_peaks = 20  # Use this to set how many peaks we
        ↪ want to find
55      # Finding peaks (local maximas) within the data set
56      peaks, xpeak_placement, ypeak_placement, xpeak_limit_left,
        ↪ xpeak_limit_right, zvalue, \
57          zvalue_leftlim, zvalue_rightlim,
            ↪ zvalue_leftlim_boundary, zvalue_rightlim_boundary \
58          = binary_search(number_of_peaks, filtered_wave,
            ↪ window_width)
59      # checking the HWHM of the peaks found
60      compared_peaks, xcoor, ycoor, lengthwavepeaks \
61          = peakcomparing(zvalue, xpeak_limit_right,
            ↪ xpeak_limit_left, zvalue_leftlim, zvalue_rightlim,
62                          zvalue_leftlim_boundary,
                            ↪ zvalue_rightlim_boundary,
                            ↪ ypeak_placement, quadrant,
                            ↪ xpeak_placement)
63      # print(compared_peaks)
64      phases = find_phase(compared_peaks)
65
66      # plotting input wave, amplitude, phase
67      plotwave(windowed_wave, datalength, datalength,
        ↪ samplepoints, samplepoints, mode)
68      plotamplitude(filtered_wave, xcoor, ycoor, mode)
69      plotphase_onealt(phases)
70      wavelength, lengthofwindow = find_wavelengths(xcoor, ycoor,
        ↪ quadrant, sampling_dist, window_width)
71      print('wavelength +- hwhm (both in meters): ' +
        ↪ str(wavelength) + ' +- ' + str(lengthofwindow))
```

analysis altitude.py

```
1  def analyze_each_alt(windowed_data, number_of_peaks,
   ↪  window_width, sampling_distx):
2      # A function that takes 3D data as input and analyzes at
       ↪  each altitude
3      from generatehorizontal import transform
4      from analysis_functions import find_quadrants,
       ↪  amp_filtering, binary_search, peakcomparing,
       ↪  find_phase, \
5          find_wavelengths, phases_altitude
6      import numpy as np
7
8      shape = np.shape(windowed_data)
9      wavepresent = []
10     phasedata = []
11     wavelengths = []
12     xcoor_vec = []
13     ycoor_vec = []
14     altitudes = []
15     lengthwavepeaks = []
16     lenwindow = 0
17     print('window width: ' + str(window_width))
18
19     # Perform a FFT2 at each altitude to transform data to
       ↪  prepare for analyzes, then performing analyzes
20     for z in range(shape[2]):
21         transformed_data = transform(windowed_data[:, :, z])  #
           ↪  alt = 70 + 0.5*z
22
23         # perform analyzes at this altitude
24         half_datafield = find_quadrants(transformed_data)
25         filtered_data = amp_filtering(half_datafield)
26         peaks, xpeak_placement, ypeak_placement,
           ↪  xpeak_limit_left, xpeak_limit_right, zvalue,
           ↪  zvalue_leftlim, \
27             zvalue_rightlim, zvalue_leftlim_boundary,
               ↪  zvalue_rightlim_boundary \
28             = binary_search(number_of_peaks, filtered_data,
               ↪  window_width)
```

```
29
30          compared_peaks, xcoor, ycoor, lengthwavepeaks = \
31              peakcomparing(zvalue, xpeak_limit_right,
                ↪  xpeak_limit_left, zvalue_leftlim,
                ↪  zvalue_rightlim,
32                            zvalue_leftlim_boundary,
                              ↪  zvalue_rightlim_boundary,
33                            ypeak_placement, half_datafield,
                              ↪  xpeak_placement)
34
35          phasedata.append(find_phase(compared_peaks))
36          altitudes.append(z)
37          wavelengths.append(np.array(find_wavelengths(xcoor,
            ↪  ycoor, half_datafield, sampling_distx,
            ↪  window_width)[0]))
38          if len(xcoor) > 0:  # if there is a wave present at
            ↪  this altitude
39              for iterator in range(len(xcoor)):
40                  wavepresent.append(z)
41                  xcoor_vec.append(xcoor[iterator])
42                  ycoor_vec.append(ycoor[iterator])
43          lenwindow = find_wavelengths(xcoor, ycoor,
            ↪  half_datafield, sampling_distx, window_width)[1]
44
45      phases_unwrapped, average_phase =
        ↪  phases_altitude(phasedata, window_width, xcoor_vec,
        ↪  ycoor_vec)
46
47      return phases_unwrapped, wavelengths, average_phase,
        ↪  lengthwavepeaks, lenwindow
```

analysis functions.py

```
1  def amp_filtering(input_wave):
2      import scipy as sp
3      import numpy as np
4      from scipy import signal
5      from scipy import ndimage
6
7      # filter the wave to only keep peaks
8      # first create a median filter, then an average filter
9      # median filter
10     median_filter = sp.signal.medfilt2d(abs(input_wave),
       ↪ kernel_size=5)
11
12     # average filter
13     average_filter = sp.ndimage.uniform_filter(median_filter,
       ↪ size=5)
14
15     filtered_array = np.subtract(abs(input_wave),
       ↪ average_filter)  # element wise subtraction
16     # the following lines are when the third filter
       ↪ implementation are used, but are not successful at the
       ↪ moment
17     # testfilter = np.empty(shape=np.shape(input_wave))
18     # y, x = np.shape(testfilter)
19     # counter1 = 0
20     # counter2 = 0
21     # for yit in range(y):  # should prob. take this away, as
       ↪ it doesn't work
22     #     for xit in range(x):
23     #         if filtered_array[yit, xit] <=
       ↪ abs(input_wave[yit, xit])*0.1:
24     #             testfilter[yit, xit] = abs(input_wave[yit,
       ↪ xit])
25     #             counter1 += 1
26     #         else:
27     #             testfilter[yit, xit] = average_filter[yit,
       ↪ xit]
28     #             counter2 += 1
```

```
29      # ------------------------------------ end of third
        ↪ implementation
30
31      filtered_array2 = np.clip(filtered_array, a_min=0,
        ↪ a_max=None)  # setting all negative values to zero
32
33      return filtered_array2
34
35
36  def find_quadrants(input_wave):  # changing to only use 1. and
    ↪ 4. quadrant
37      import numpy as np
38      f, g = np.shape(input_wave)
39      halflen = int(g/2)  # the windowed function is power of 2,
        ↪ and will thus always be int when divided by 2
40      quadrants = input_wave[:, halflen:]
41
42      return quadrants
43
44
45  def identify_peaks(filtered_wave, threshold_multiplier,
    ↪ window_width):
46      # find the max of each peak, then using half-width to
        ↪ identify as wave peaks
47      import numpy as np
48      # import scipy
49      import scipy.ndimage as ndimage
50      import scipy.ndimage.filters as filters
51
52      f, g = np.shape(filtered_wave)
53      halfwidth = round(window_width)/2
54      halfwidth_int = int(round(halfwidth))
55
56      # For finding the peaks, with help of a maximum filter
57      max_value = np.max(filtered_wave)
58      neighborhood_size = 5  # 2*int(round(window_width))
59      threshold = max_value*threshold_multiplier  # threshold
        ↪ multiplier <= 1
60      data_max = filters.maximum_filter(filtered_wave,
        ↪ neighborhood_size)
61      maxima = (filtered_wave == data_max)
```

```
62      data_min = filters.minimum_filter(filtered_wave,
        ↪  neighborhood_size)
63      diff = ((data_max - data_min) > threshold)
64      maxima[diff == 0] = 0
65
66      labeled, num_objects = ndimage.label(maxima)
67      peaks = np.array(ndimage.center_of_mass(filtered_wave,
        ↪  labeled, range(1, num_objects + 1)))
68
69      # finding the points at half maximum amplitude
70      xpeak_placement = []  # placement of the center of peak
71      ypeak_placement = []  # placement of the center of peak
72      xpeak_xvalues_left = []  # vector for position inside left
        ↪  HWHM value
73      xpeak_xvalues_right = []  # vector for position inside
        ↪  right HWHM value
74      xpeak_xvalues_left_boundary = []  # vectors for position
        ↪  outside left HWHM value
75      xpeak_xvalues_right_boundary = []  # vectors for position
        ↪  outside right HWHM value
76      if np.shape(peaks) != (0,):
77          ii, jj = np.shape(peaks)
78          for i in range(ii):
79              ypeak_placement.append(int(peaks[i, 0]))
80              xpeak_placement.append(int(peaks[i, 1]))
81
82      for itlim in range(len(xpeak_placement)):  # Checking if
        ↪  the waves will have values outside the data field
83          if halfwidth_int+2 > xpeak_placement[itlim]:  # if the
            ↪  center of the peak is lower than the value of
            ↪  width of
84              # the peak, the wave will have values outside the
                ↪  datafield
85              xpeak_xvalues_right.append(xpeak_placement[itlim] +
                ↪  (halfwidth_int-1))
86              xpeak_xvalues_left.append(0)
87
                ↪  xpeak_xvalues_right_boundary.append(xpeak_placement[itlim]
                ↪  + (halfwidth_int + 1))
88              xpeak_xvalues_left_boundary.append(0)
89
```

```
90          elif g-halfwidth_int-2 < xpeak_placement[itlim]:  # if
            ↪ the center of the peak is right of the value of
            ↪ width of
91              # the peak, the wave will have values outside the
                ↪ datafield
92              xpeak_xvalues_left.append(xpeak_placement[itlim] -
                ↪ (halfwidth_int-1))
93              xpeak_xvalues_right.append(g-1)
94
                ↪ xpeak_xvalues_left_boundary.append(xpeak_placement[itlim]
                ↪ - (halfwidth_int+1))
95              xpeak_xvalues_right_boundary.append(g - 1)
96
97          else:
98              xpeak_xvalues_right.append(xpeak_placement[itlim] +
                ↪ (halfwidth_int-1))
99              xpeak_xvalues_left.append(xpeak_placement[itlim] -
                ↪ (halfwidth_int-1))
100
                ↪ xpeak_xvalues_right_boundary.append(xpeak_placement[itlim]
                ↪ + (halfwidth_int+1))
101
                ↪ xpeak_xvalues_left_boundary.append(xpeak_placement[itlim]
                ↪ - (halfwidth_int+1))
102
103     # Identifying the wave peaks and removing the noise peaks
        ↪ with fwhm
104     zvalue = []  # amplitude value of the center of peak
105     zvalue_leftlim = []  # amplitude value for left HWHM value
        ↪ - 1 positon
106     zvalue_rightlim = []  # amplitude value for right HWHM
        ↪ value - 1 positon
107     zvalue_leftlim_boundary = []  # amplitude value for
        ↪ position outside left HWHM value
108     zvalue_rightlim_boundary = []  # amplitude value for right
        ↪ HWHM value + 1 positon
109
110     peaks_checked = []
111
112     for iterator in range(len(xpeak_placement)):
```

```
113             zvalue.append(filtered_wave[ypeak_placement[iterator],
                ↪ xpeak_placement[iterator]] / 2)
114
                ↪ zvalue_leftlim.append(filtered_wave[ypeak_placement[iterator],
                ↪ xpeak_xvalues_left[iterator]])
115
                ↪ zvalue_rightlim.append(filtered_wave[ypeak_placement[iterator],
                ↪ xpeak_xvalues_right[iterator]])
116             zvalue_rightlim_boundary.append\
117                 (filtered_wave[ypeak_placement[iterator],
                    ↪ xpeak_xvalues_right_boundary[iterator]])
118             zvalue_leftlim_boundary.append\
119                 (filtered_wave[ypeak_placement[iterator],
                    ↪ xpeak_xvalues_left_boundary[iterator]])
120
121     return peaks_checked, xpeak_placement, ypeak_placement,
        ↪ xpeak_xvalues_left, xpeak_xvalues_right, zvalue, \
122         zvalue_leftlim, zvalue_rightlim,
            ↪ zvalue_leftlim_boundary, zvalue_rightlim_boundary
123
124
125 # Check if the peaks found has a hwhm that coincides with
    ↪ window function
126 def peakcomparing(zvalue, xpeak_xvalues_right,
    ↪ xpeak_xvalues_left, zvalue_leftlim, zvalue_rightlim,
127                   zvalue_leftlim_boundary,
                       ↪ zvalue_rightlim_boundary,
                       ↪ ypeak_placement, unfiltered_wave,
                       ↪ xpeak_placement):
128     import numpy as np
129     longestwave = []  # storing the length of the waves found
130     peaks_checked = []
131     peak_coor_x = []
132     peak_coor_y = []
133     lengthwavepeaks = 0
134     if len(zvalue) > 0:  # Using the length of the longest
        ↪ peak to find the values of the peak matrix
135         for l in range(len(zvalue)):
136             longestwave.append(abs(xpeak_xvalues_right[l] -
                ↪ xpeak_xvalues_left[l]))
137         lengthwavepeaks = max(longestwave)
```

```
138         peaks_checked = np.empty((0, lengthwavepeaks),
            ↪ dtype=complex)  # the peaks checked to
139         peak_check = []  # temporary vector to store peaks to
            ↪ check for HWHM
140         for m in range(len(zvalue)):
141             if zvalue_leftlim[m] >= zvalue[m]*0.95 and
                ↪ zvalue_rightlim[m] >= zvalue[m]*0.95:
142                 if zvalue_leftlim_boundary[m] <= zvalue[m]*1.05
                    ↪ and zvalue_rightlim_boundary[m] <=
                    ↪ zvalue[m]*1.05:
143                     for n in range(xpeak_xvalues_left[m],
                        ↪ xpeak_xvalues_right[m]):
144                         peak_check.append(unfiltered_wave
145                                           [ypeak_placement[m],
                                              ↪ n])
146                     peak_coor_x.append(xpeak_placement[m])
147                     peak_coor_y.append(ypeak_placement[m])
148                     peaks_checked = np.vstack((peaks_checked,
                        ↪ peak_check))  # if check is positive,
                        ↪ add to a matrix
149                     peak_check = []
150         peaks_checked = np.array(peaks_checked)
151
152     return peaks_checked, peak_coor_x, peak_coor_y,
        ↪ lengthwavepeaks
153
154
155 # perform a binary search with respect to amplitude, starting
    ↪ from the top and working down
156 def binary_search(peaks_wanted, filtered_wave, window_width):
157     peak_vec = []
158     xpeak_placement_vec = []
159     ypeak_placement_vec = []
160     xpeak_zvalues_left = []
161     xpeak_zvalues_right = []
162     zvalue = []
163     zvalue_leftlim = []
164     zvalue_rightlim = []
165     zvalue_leftlim_boundary = []
166     zvalue_rightlim_boundary = []
167     top = 1
```

```
168     end = 0
169     found = False
170 
171     while top-end >= 0.01 and not found:
172         searchpoint = (top + end)/2
173         peak_vec, xpeak_placement_vec, ypeak_placement_vec,
            ↪ xpeak_zvalues_left, xpeak_zvalues_right, zvalue, \
174             zvalue_leftlim, zvalue_rightlim,
                ↪ zvalue_leftlim_boundary,
                ↪ zvalue_rightlim_boundary \
175             = identify_peaks(filtered_wave, searchpoint,
                ↪ window_width)
176         if len(xpeak_placement_vec) == peaks_wanted:
177             found = True
178         else:
179             if len(xpeak_placement_vec) > peaks_wanted:
180                 end = searchpoint
181             else:
182                 top = searchpoint
183 
184     return peak_vec, xpeak_placement_vec, ypeak_placement_vec,
        ↪ xpeak_zvalues_left, xpeak_zvalues_right, zvalue, \
185         zvalue_leftlim, zvalue_rightlim,
            ↪ zvalue_leftlim_boundary, zvalue_rightlim_boundary
186 
187 
188 def find_wavelengths(peaktops_x, peaktops_y, waveplane,
    ↪ grid_size, windowwidth):
189     import numpy as np
190     # checking if 2 peaks at different z have wavelength_1
        ↪ approx. equal to wavelength_2 (within FWHM/2)
191     # wavelength lambda = scale*2pi/sqrt(k_x[x]^2 +
        ↪ k_y[y]^2), where [x,y] are coordinates of the point of
        ↪ max amp
192     peaks_y_wavelength = peaktops_y
193     lam = []  # wavelengths lambda
194     f, g = np.shape(waveplane)
195     window_length = 2*grid_size*np.pi/(np.sqrt(windowwidth**2))
196     g1 = g
197     f1 = f
198     if f > g:
```

```
199             f1 = g1
200             for k in range(len(peaktops_y)):
201                 peaks_y_wavelength[k] = abs(peaks_y_wavelength[k] -
                    ↪  g1)
202                 # print(peaks_y_wavelength)
203         k_x = np.linspace(0, np.pi, g1, endpoint=False)
204         k_y = np.linspace(0, np.pi, f1, endpoint=False)
205 
206         for i in range(len(peaktops_x)):
207             lam.append(2*grid_size*np.pi /
208                        (np.sqrt(k_x[peaktops_x[i]]**2 +
209                                 k_y[peaks_y_wavelength[i]]**2)))
210         return lam, window_length
211 
212 
213 def find_phase(peaks):
214     # this function is to find and analyze the phase
215     import cmath
216     import numpy as np
217 
218     f, g = np.shape(peaks)
219     peakphase = np.zeros((f, g))
220     # peakphase_unwrapped = np.zeros((f, g))
221     for x in range(f):
222         for y in range(g):
223             peakphase[x, y] = cmath.phase(peaks[x, y])
224 
225     return peakphase
226 
227 
228 def phases_altitude(phasematrix, window_width, xcoor, ycoor):
229     import numpy as np
230     length = window_width/2  # length of each array peak
231     halflen = int(round(length))
232     print('halflen: ' + str(halflen))
233     # wavephasesunwrapped = []
234     phasescorrected = []
235     average = []
236     if len(xcoor) > 0:
237         # finding the coordinates of each unique wave peak,
            ↪  and if two peaks are within window_width,
```

```
238         # then count them as one
239         unique_xcoor = list(set(xcoor))
240         unique_ycoor = list(set(ycoor))
241         verified_xcoor = [unique_xcoor[0]]
242         verified_ycoor = [unique_ycoor[0]]
243 
244         for qx in range(1, len(unique_xcoor)):
245             if not unique_xcoor[qx-1] + halflen >
                ↪  unique_xcoor[qx] > unique_xcoor[qx-1]-halflen:
246                 verified_xcoor.append(unique_xcoor[qx])
247         for rx in range(1, len(unique_ycoor)):
248             if not unique_ycoor[rx-1] + halflen >
                ↪  unique_ycoor[rx] > unique_ycoor[rx-1]-halflen:
249                 verified_ycoor.append(unique_ycoor[rx])
250         print('coordinates in x: ' + str(unique_xcoor))
251         print('coordinates in y: ' + str(unique_ycoor))
252         print('unique coordinates in x: ' +
            ↪  str(verified_xcoor))
253         print('unique coordinates in y: ' +
            ↪  str(verified_ycoor))
254 
255         wavephases = [[] for p in range(len(verified_xcoor))]
256         phase = [[] for q in range(len(verified_xcoor))]
257         wavephasesunwrapped = [[] for r in
            ↪  range(len(verified_xcoor))]
258         counter = 0
259         average = [[] for s in range(len(verified_xcoor))]
260         # finding the phases of these waves, with help of the
            ↪  coordinates
261         for i in range(len(phasematrix)):
262             altitude_array = phasematrix[i]
263             m, n = np.shape(altitude_array)
264             for mi in range(m):
265                 for mx in range(len(verified_xcoor)):
266                     for my in range(len(verified_ycoor)):
267                         if int(xcoor[counter] + halflen) >
                            ↪  int(verified_xcoor[mx]) >
                            ↪  int(xcoor[counter] - halflen) and \
```

```
268                                     int(ycoor[counter] + halflen) >
                                        ↪  int(verified_ycoor[my]) >
                                        ↪  int(ycoor[counter] -
                                        ↪  halflen):
269
                                    ↪  wavephases[mx].append(altitude_array[mi])
270
271                 counter += 1
272
273         for it in range(len(wavephases)):
274             phase[it] = np.concatenate(wavephases[it]).ravel()
275
276         for t in range(len(phase)):
277             wavephasesunwrapped[t] = np.unwrap(phase[t])  # ,
                ↪  discont=np.pi/2)
278
279         phasescorrected = 0.5*np.array(wavephasesunwrapped)
280
281         def runningmean(x, N):
282             return np.convolve(x, np.ones((N,)) / N,
                ↪  mode='valid')[(N - 1):]
283
284         for m in range(len(phasescorrected)):
285             average[m] = runningmean(phasescorrected[m], 8)
286
287     return phasescorrected, average
```

generatehorizontal.py

```
# this script generates a synthetic wave in my_inputwave(), while windowing and transform performs windowing and FFT in
# the horizontal plane. shift_bit_length makes sure the length of FFT is power of 2, while makeSpectrum(...) is another
# function to transform the wave (but not used).


def my_inputwave():  # SCRIPT FOR GENERATING INPUT TO THE CODE
    # including necessary functions
    from math import pi
    import numpy as np
    datalength = 250000
    dist = 5000
    # Since the grid view is 250 km horizontal, 60 km vertical, and the resolution is 5*5*0,2 km
    # we get 250 km / 5 km = 50 => number of samplepoints in the horizontal
    # Number of samplepoints:
    samplepoints = int(datalength / dist)
    # dist = sample spacing: m per spacing

    # making some noise to the input signal
    noise = np.random.normal(0, 1, samplepoints)

    # defining a function describing a wave
    def f(a, b):
        return amp * np.sin(2*pi*(a/lam) + 2*pi*(b/lam) + phi + noise)

    # generating a wave to have a synthetic input for testing
    amp = 3  # amplitude in R
    phi = 35  # vertical wavelength in km
    lam = 28284  # in meters
```

```
29      x = np.linspace(0, datalength, samplepoints,
        ↪  endpoint=False)  # evenly spaced values within given
        ↪  interval,
30      y = np.linspace(0, datalength, samplepoints,
        ↪  endpoint=False)  # # start, stop, number of samples
31      xx, yy = np.meshgrid(x, y)  # makes the vectors into a
        ↪  matrix
32      psi = f(x[np.newaxis, :], y[:, np.newaxis])
33
34      return psi, xx, yy, datalength, dist, samplepoints
35
36
37  def windowing(input_wave, datalength, dist):
38      import numpy as np
39      # Perform hamming in 2D
40      m, n = np.shape(input_wave)
41      hy = np.hamming(m)
42      hx = np.hamming(n)
43      ham2d = np.sqrt(np.outer(hy, hx))
44      windowed_wave = input_wave * ham2d
45      fftlen = shift_bit_length(10*m)
46      del_f_fwhm = 1.30/datalength
47      del_f_sample = (1/(2*dist))/(fftlen/2)
48      fwhm_window = del_f_fwhm/del_f_sample
49      return windowed_wave, fwhm_window
50
51
52  def transform(input_wave):
53      import numpy as np
54      # finding the sizes of the inputs, to make a FFT of
        ↪  correct size
55      wavey, wavex = np.shape(input_wave)
56      nwavex = shift_bit_length(10 * wavex)
57      nwavey = shift_bit_length(10 * wavey)
58
59      # Performing a 2D FFT of the models input
60      psi_transformed = np.fft.fftshift(np.fft.fft2(input_wave,
        ↪  s=[nwavey, nwavex]))
61
62      return psi_transformed
63
```

```
64
65 def shift_bit_length(x):  # to find smallest power of 2
   ↪ greater than x
66     # taken from https://tinyurl.com/ybv7gnbg
67     return 1 << (x-1).bit_length()
68
69
70 def makeSpectrum(E, dx, dy, upsample=10):
71     import numpy as np
72     import numpy.fft as fft
73
74     zeropadded = np.array(E.shape) * upsample
75
76     F = fft.fftshift(fft.fft2(E, zeropadded)) / E.size
77     xf = fft.fftshift(fft.fftfreq(zeropadded[1], d=dx))
78     yf = fft.fftshift(fft.fftfreq(zeropadded[0], d=dy))
79     return F, xf, yf
```

inputdata.py

```
# This is 3D data. loaddata() reads from data provided by Anqi Li, while syntdata3d() is the same wave as generated in
# only one height, but in 3 dimensions. Transform3d and windowing3d performs a FFT and windowing respectively, in 3D


def loaddata():
    import numpy as np
    data = np.load('Vat_Pertur_t.npy', mmap_mode='r')
    # parameters = np.load('wave_para.npy')  # uncomment if the parameters in the data set are wanted
    yview, xview, zview, n = np.shape(data)
    datalengthy = 1440000
    datalengthx = 400000
    print('the number of data points in the input data is: ' + str(xview) + ' in x-dir and ' + str(yview) +
          ' in y-dir, at ' + str(zview) + ' altitudes z_n, and we have ' + str(n) + ' set of this data')
    dataset = np.zeros(shape=(yview, xview, zview))
    # Make data to a 3d matrix
    for z in range(zview):
        dataset[:, :, z] = data[:, :, z, 0]
    sampling_dist_x = datalengthx / xview
    sampling_dist_xint = int(round(sampling_dist_x))
    # print(parameters)  # uncomment if the parameters in the data set are wanted

    return dataset, datalengthy, datalengthx, yview, xview, sampling_dist_xint, sampling_dist_x


def windowing3d(input_data, datalengthx, sampling_dist):
    import numpy as np
    from generatehorizontal import windowing
    # takes 3d data and applies a window to it
    yview, xview, zview = np.shape(input_data)
```

```
30      windowed_data = np.empty(shape=[yview, xview, zview])
31      window_width = 0
32      for i in range(zview):
33          windowed_data[:, :, i], window_width =
            ↪ windowing(input_data[:, :, i], datalengthx,
            ↪ sampling_dist)
34      print(str(np.shape(windowed_data)) + ' is the shape of the
        ↪ windowed array')
35
36      return windowed_data, window_width
37
38
39  def syntdata3d():
40      # including necessary functions
41      from math import pi
42      import numpy as np
43      datalength = 250000
44      dist = 5000
45      # Since the grid view is 250 km horizontal, 60 km
        ↪ vertical, and the resolution is 5*5*0,2 km
46      # we get 250 km / 5 km = 50 => number of samplepoints in
        ↪ the horizontal
47      # Number of samplepoints:
48      samplepoints = int(datalength / dist)
49      # dist = sample spacing: m per bin
50
51      # making some noise to the input signal
52      noise = np.random.normal(0, 1, samplepoints)
53
54      # defining a function describing a wave
55      def f(xvec, yvec, zvec):
56          return amp * np.cos(2*pi*(xvec/lam) + 2*pi*(yvec/lam) +
            ↪ 2*pi*(zvec/lamz) + noise)
57
58      # generating a wave to have a synthetic input for testing
59      amp = 3  # amplitude in R
60      lamz = 20  # vertical wavelength in km = phase
61      lam = 74953  # in meters
62      x = np.linspace(0, datalength, samplepoints,
        ↪ endpoint=False)  # evenly spaced values within given
        ↪ interval,
```

```
63      y = np.linspace(0, datalength, samplepoints,
        ↪  endpoint=False)  # # start, stop, number of samples
64      psiz = np.empty(shape=[samplepoints, samplepoints, 160])
65      xx, yy = np.meshgrid(x, y)  # makes the vectors into a
        ↪  matrix
66      for z in range(160):
67          psiz[:, :, z] = f(x[np.newaxis, :], y[:, np.newaxis],
            ↪  z)
68      print(np.shape(psiz))
69      return psiz, xx, yy, datalength, dist, samplepoints
```

plotting.py

```
1  # This script is for plotting the different wave
   ↪ characteristics, spectrums or waves. Plotwave is for
   ↪ plotting the input
2  # wave in 2D (horizontal plane), while plotamplitude plots
   ↪ the amplitude of the transformed (and filtered) input.
3  # Plotphase plots the development of the phase with respect
   ↪ to altitude, while plotwavelengths plots the wavelengths
4  #  with respect to altitude. Plotphaseonealt plots the phase
   ↪ of a peak at one altitude, and is only used in !mode.
5  # Plotthewave plots the wave, but is only used with the
   ↪ alternative generate function makeSpectrum
6
7
8  def plotwave(genwave, datalengthy, datalengthx, samplepointsx,
   ↪ samplepointsy, mode):
9      # this function is to plot the generated wave, fft, and
       ↪ phase
10     import numpy as np
11     import matplotlib.pyplot as plt
12     from mpl_toolkits.mplot3d import Axes3D
13
14     if mode:  # values for x and y from Anqis thesis (result
       ↪ plot)
15         x = np.linspace(-200, 200, samplepointsx)  # evenly
           ↪ spaced values within given interval,
16         y = np.linspace(-670, 770, samplepointsy)  # # start,
           ↪ stop, number of samples
17         x, y = np.meshgrid(x, y)  # makes the vectors into a
           ↪ matrix
18
19     else:
20         x = np.linspace(0, datalengthx, samplepointsx)  #
           ↪ evenly spaced values within given interval,
21         y = np.linspace(0, datalengthy, samplepointsy)  # #
           ↪ start, stop, number of samples
22         x, y = np.meshgrid(x, y)  # makes the vectors into a
           ↪ matrix
23
```

```
24      # plot the generated wave: genwave
25      fig = plt.figure()
26      ax = Axes3D(fig)
27      ax.plot_surface(x/1000, y/1000, genwave)
28      ax.tick_params(axis='both', which='major', labelsize=16)
29      plt.title('The wave input', fontsize=22)
30      plt.xlabel('x [km]', fontsize=20)
31      plt.ylabel('y [km]', fontsize=20)
32      plt.show()
33
34      return 0
35
36
37  def plotamplitude(filtered_wave, xpeak_placement,
    ↪  ypeak_placement, mode):
38      import numpy as np
39      import matplotlib.pyplot as plt
40      from mpl_toolkits.mplot3d import Axes3D
41      import matplotlib.ticker as tck
42      # making vectors to place the peaks in the plot of the
        ↪  filtered wave
43      xpeak_values = []
44      ypeak_values = []
45
46      f, g = np.shape(filtered_wave)
47      if mode:
48          g1 = g
49          if f > g:
50              for k in range(len(ypeak_placement)):
51                  ypeak_placement[k] = abs(ypeak_placement[k] +
                    ↪  g1)
52
53      xx = np.linspace(-np.pi, np.pi, g)
54      yy = np.linspace(-np.pi, np.pi, f)
55      for xi in range(len(xpeak_placement)):
56          xpeak_values.append(xx[xpeak_placement[xi]])
57          ypeak_values.append(yy[ypeak_placement[xi]])
58
59      # plot the filtered wave
60      xx, yy = np.meshgrid(xx, yy)
61      fig = plt.figure()
```

```
62      ax = Axes3D(fig)
63      ax.xaxis.set_major_formatter(tck.FormatStrFormatter('%g
        ↪ $\pi$'))
64      ax.xaxis.set_major_locator(tck.MultipleLocator(base=1.0))
        ↪ # using this and line above to get pi on x-axis
65      ax.yaxis.set_major_formatter(tck.FormatStrFormatter('%g
        ↪ $\pi$'))
66      ax.yaxis.set_major_locator(tck.MultipleLocator(base=1.0))
        ↪ # using this and line above to get pi on y-axis
67      ax.plot_surface(xx, yy, filtered_wave)
68      ax.plot(xpeak_values, ypeak_values, 'r+')
69
70      plt.title('Plot of the filtered FFT of the wave, 1st and
        ↪ 4th quadrant only', fontsize=16)
71      plt.xlabel('k_x', fontsize=14)
72      plt.ylabel('k_y', fontsize=14)
73      plt.show()
74      return 0
75
76
77  def plotphase(phases, average):
78      import numpy as np
79      import matplotlib.pyplot as plt
80      import matplotlib.ticker as tck
81      wave1 = np.array(phases[0])
82      average1 = np.array(average[0])
83      wave2 = []
84      average2 = []
85      altvectorwave2 = []
86      altvectorave2 = []
87      y2 = []
88
89      # plot the phase
90      ax = plt.subplot()
91      ax.yaxis.set_major_formatter(tck.FormatStrFormatter('%g
        ↪ $\pi$'))
92      ax.yaxis.set_major_locator(tck.MultipleLocator(base=1.0))
        ↪ # using this and line above to get pi on axis
93
94      altvectorwave1 = np.linspace(70, 150, len(wave1),
        ↪ endpoint=False)
```

```
95      altvectorave1 = np.linspace(70, 150, len(average1),
        ↪  endpoint=False)
96
97      # perform a polynomial regression:
98      firstpoly = np.polyfit(altvectorave1, average1, deg=2,
        ↪  full=True)
99
100     def y1(x1):
101         return firstpoly[0][0]*x1**2 + firstpoly[0][1]*x1 +
            ↪  firstpoly[0][2]
102
103     print('mathematical description of first waves phase: ' +
        ↪  str(firstpoly[0]))
104
105     y1z = y1(altvectorwave1)
106
107     if len(phases) > 1:
108         wave2 = np.array(phases[1])
109         average2 = np.array(average[1])
110         altvectorwave2 = np.linspace(70, 150, len(wave2))
111         altvectorave2 = np.linspace(70, 150, len(average2))
112         # perform a polynomial regression
113         secondpoly = np.polyfit(altvectorave2, average2, deg=2,
            ↪  full=True)
114
115         def y2(x2):
116             return secondpoly[0][0] * x2 ** 2 +
                ↪  secondpoly[0][1] * x2 + secondpoly[0][2]
117
118         print('mathematical description of second waves phase:
            ↪  ' + str(secondpoly[0]))
119
120     plt.plot(altvectorwave1, wave1 / np.pi, label=' wave 1')
121     plt.plot(altvectorave1, average1 / np.pi, color='red',
        ↪  label='average wave 1')
122     plt.plot(altvectorwave1, y1z / np.pi, label='2. deg.
        ↪  poly.fit wave 1')
123     if len(phases) > 1:
124         y2z = y2(altvectorwave2)
125         plt.plot(altvectorwave2, wave2 / np.pi, label='wave 2')
```

```
126         plt.plot(altvectorave2, average2 / np.pi,
            ↪  color='green', label='average wave 2')
127         plt.plot(altvectorwave2, y2z / np.pi, label='2. deg.
            ↪  poly.fit wave 2')
128     ax.tick_params(axis='both', which='major', labelsize=20)
129     plt.xlabel('Altitude in km', fontsize=22)
130     plt.ylabel('\u03C6(z)', fontsize=22)
131     plt.legend(fontsize=20)
132     plt.title('Plot of the phase of the waves found',
        ↪  fontsize=26)
133     plt.show()
134
135     return 0
136
137
138 def plotwavelengths(wavelengths, window_length):
139     import numpy as np
140     import matplotlib.pyplot as plt
141
142     half_window_length = window_length/2
143     print('window length of wavelength ' +
        ↪  str(half_window_length))
144     length = len(wavelengths)
145
146     zvec = np.linspace(70, 150, length, endpoint=False)
147     ax = plt.subplot()
148     for xe, ye in zip(zvec, wavelengths):
149         ax.plot([xe] * len(ye), ye/1000, 'o')
150         ax.errorbar([xe] * len(ye), ye/1000,
            ↪  yerr=half_window_length/1000, fmt='o', ecolor='g')
151     ax.tick_params(axis='both', which='major', labelsize=20)
152     plt.title('Plot of the wavelengths', fontsize=26)
153     plt.xlabel('Altitude in km', fontsize=22)
154     plt.ylabel('Wavelength in km', fontsize=22)
155     plt.show()
156     return 0
157
158
159 def plotphase_onealt(phases):
160     import numpy as np
161     import matplotlib.pyplot as plt
```

```
162     import matplotlib.ticker as tck
163
164     # plot the phase
165     ax = plt.subplot()
166     ax.yaxis.set_major_formatter(tck.FormatStrFormatter('%g
        ↪ $\pi$'))
167     ax.yaxis.set_major_locator(tck.MultipleLocator(base=1.0))
        ↪ # using this and line above to get pi on axis
168     m, n = np.shape(phases)
169     xphase = np.linspace(0, n, n, endpoint=False)
170     for l in range(m):
171         ax.plot(xphase, phases[l, :] / np.pi, 'o')
172     plt.title('Plot of the phase of the waves found',
        ↪ fontsize=16)
173     plt.show()
174     return 0
175
176
177 def plotthewave(F, xf, yf):
178     # import numpy as np
179     import matplotlib.pyplot as plt
180     from mpl_toolkits.mplot3d import Axes3D
181     # x = np.linspace(-np.pi, np.pi, len(xf)) / np.pi
182     # y = np.linspace(-np.pi, np.pi, len(yf)) / np.pi
183     # x, y = np.meshgrid(xf, yf)
184     fig = plt.figure()
185     ax = Axes3D(fig)
186     ax.plot_surface(xf, yf, F)
187     plt.title('The wave input with alternate transform',
        ↪ fontsize=16)
188     plt.xlabel('x', fontsize=14)
189     plt.ylabel('y', fontsize=14)
190     plt.show()
191     return 0
```