UiT THE ARCTIC UNIVERSITY OF NORWAY

Faculty of Science and Technology
Department of Physics and Technology

An investigation of the spatial and temporal distribution of kinetic energy in the mesosphere

The high latitude mesosphere

—

Ole K. Nordaunet

FYS-3931: Master's thesis in space physics 30 SP - July 2020

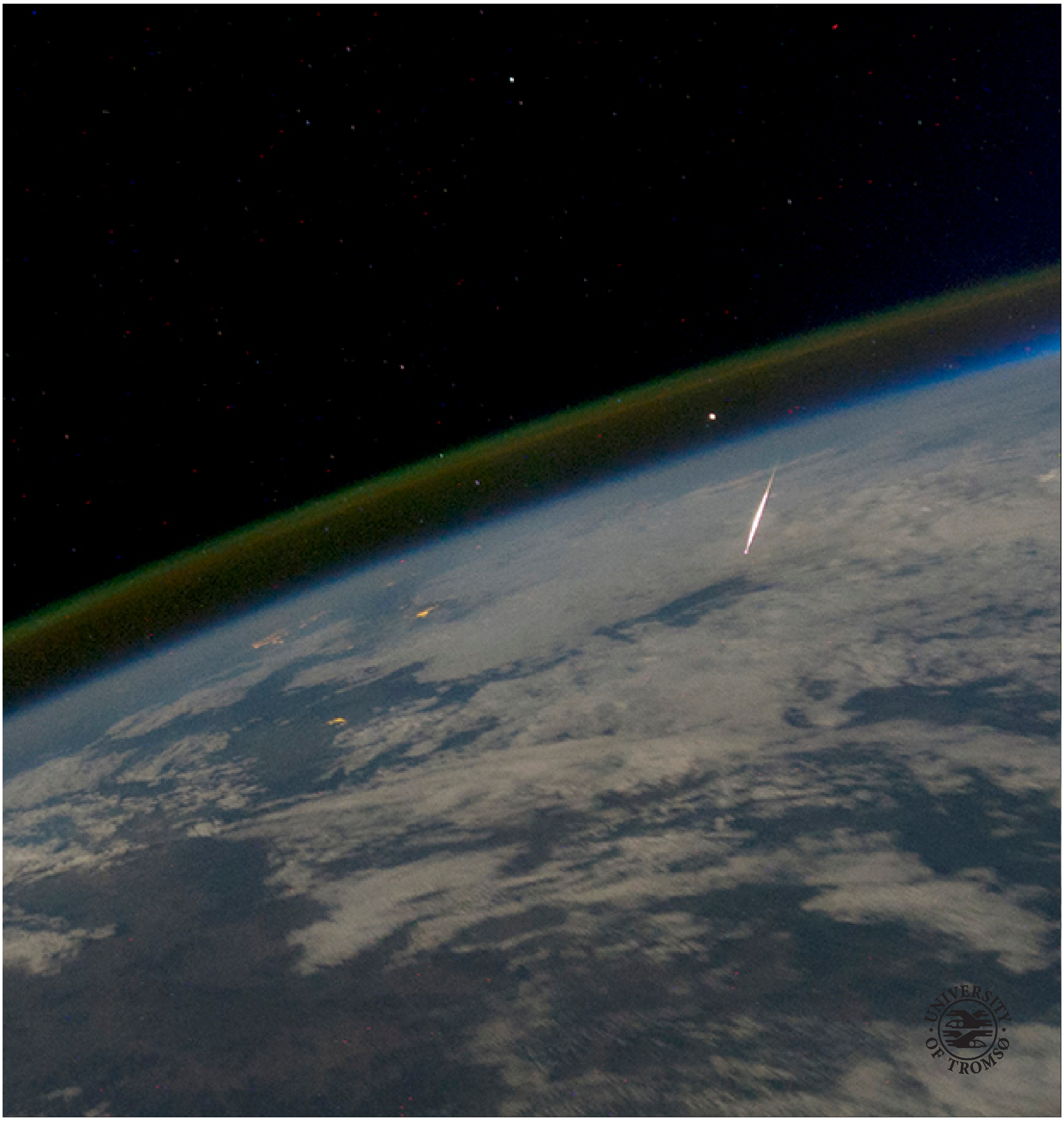

Photo on front page: Astronaut Ron Garan, 2011, NASA. https://solarsystem.nasa.gov/resources/434/looking-down-on-a-shooting-star/

Abstract

The mesosphere is perhaps the least explored region in the atmosphere with very few methods of observing. This thesis will primarily be exploring a new technique for measuring the distribution of kinetic energy in the mesosphere across a wide range of spatial and temporal scales. The method being used relies on correlation functions between pairs of meteor measurements. These measurements are made using a network of specular meteor radars located in Northern Norway. This network produced 32 million meteor measurements over a 2 year period. The correlation function estimation method has been previously used on a smaller data set, but has so far not been used for a longer data set and at high latitudes. The main advantage of the new technique is that by studying the second order statistics of the wind field, we can obtain significantly better temporal and spatial resolution than before. Such a large data set allows for great resolution for both spatial and temporal correlation functions. By using temporal correlation functions and the kinetic energy spectrum, different atmospheric wave phenomena can be studied. These include diurnal and semi diurnal tides. The horizontal and vertical correlation functions will be used to verify that the kinetic energy follows a power law, as theoretically expected by the Kolmogorov theory for turbulence. This was done by using a second order structure function applied to correlation functions. The temporal and horizontal correlation functions were used to study the summer-winter variation in kinetic energy, some variation in the temporal domain is the impact from large scale waves as well as in the power spectra were there is a steeper power law slope during the winter. As for the horizontal domain there are differences in kinetic energy in the zonal and meridional direction for both large and small scale waves. The dataset in this thesis a lot more can be found out about the mesosphere, in this thesis only a few of the possibilities are explored. The results are in agreement with earlier work, confirming the results obtained by the earlier study.

Preface

I would first and foremost like to thank my supervisor Professor Juha Vierinen for suggesting the thesis subject as it has been very interesting, and also helping me a lot during this thesis. I would also like to given an appreciation for my fellow master students for helping me stay on the right course especially during the covid-19 situation. As well as all the support from my family.

I would also like to thank Cecilie Glittum, for English correction help on a short notice.

List of Figures

Contents

1 Introduction

The mesosphere is not defined to a specific range but it is normally in the range between 50-100 km. This altitude is in the range from Armstrong limit, where astronauts need pressure suits, and the kármán line, where planes can't fly due to insufficient lift. Even though the drag is low it is still high enough for satellites being unable to stay in orbit at that altitude. The mesosphere and also the layer above called the lower thermosphere are often useful to be called as one region, the MLT region (Mesosphere and lower thermosphere region).

The mesosphere has the lowest temperatures in our atmosphere with temperatures below -100°C, the temperature decreases with increasing altitude until the end of the mesosphere where the mesospause resides. The mesospause is defined to be the coldest place in the atmosphere, the mesospause's altitude varies quite a lot depending on latitude and solstice season. The reason it is cold is due to low absorption of solar radiation by thin atmosphere and CO_2's radiative emission which causes photons to fly out towards space and heating up the thermosphere where the temperature starts increasing rapidly with altitude.

Atmospheric tides come from different periodic oscillations caused by certain phenomena such as the big oscillation from the 24 hour day-night cycle which is caused by the Sun's heating where it gets significantly hotter during the day, and colder during the night. This causes the main tide in the mesosphere called the diurnal tide, there are also the 12-hour semi-diurnal tide which is also quite significant as well as 8-hour and 6-hour tide which has a much smaller amplitude. The biggest other effect than tides are gravity waves which are formed in the troposphere and travel up and break in the mesosphere. (Smith, 2012)

1.1 Main features of mesospheric wind climatology

Climatology is the study of climate using data for very long periods of time. Climatology is a good way to make general predictions about a certain location in a region at a certain time of year. The MSIS model (mass spectrometer incoherent scatter) empirical model from (Hedin, 1991), which has been used for a long time has a lot of uncertainty when it comes to small scale variations such as gravity waves. There are also other models such as The Horizontal Wind Model (HWM07) from (Drob et al., 2008) describing the horizontal wind in the whole Earth atmosphere. However, both these atmospheric models have trouble due to the mesosphere having year to year variations. These variation are results of the climate in this region being unstable. Some reasons the climate is unstable is due to external forcing and change in the composition it self. As human pollution is changing the composition in the mesosphere, processes from above such as radiation from the Sun will have varied effects. And externally forced changes is due process from regions below the mesosphere such as the troposphere and stratosphere, where mainly gravity waves are formed which impacts the mesosphere greatly.

1.2 The role of gravity waves and tides in the MLT and in the whole Earth's atmosphere

In the MLT gravity waves is important for circulation, which means gravity waves are important for the global mean heating and cooling. Gravity waves are formed in the troposphere and they travel all the way up to the MLT where the gravity waves either dissipate or break depending on the initial amplitude. When the initial amplitude is big the gravity wave will grow exponentially until it breaks (Smith, 2012).

Gravity waves have an impact on the zonal wind in the middle-upper MLT, as the zonal wind changes when interacting with dissipating or breaking gravity waves. Gravity waves also have an effect on other waves such as planetary waves, since stationary planetary waves in the MLT have been shown to be out of phase with the planetary waves in the stratosphere (Smith, 2012).

Gravity waves also interact with the tides, wave breaking can damp or amplify depending on the phase of the tide as described in (Smith, 2012).

Tides have an impact on the circulation in the atmosphere, especially the diurnal tide, which is a migrating tide, because of its big amplitude. Gravity waves will also depend on the phase of the diurnal tide, there are also other tides such as the semi diurnal tide which is not as significant as the diurnal tide. There are also even smaller tides but those are much less significant.

1.3 Scope of this thesis

The main goal of this thesis is to estimate the spatial and temporal correlation function of the mesospheric wind vector field. The spatial correlation in the horizontal direction will be used to validate that the kinetic energy of the wind depletes as a power law of distance traveled. Previous observations of the kinetic energy spectrum in the horizontal wind have been shown to follow a power law of k^{-3} for larger scale waves on the meso-α scale, and for smaller waves on the meso-β scale it has been shown to be $k^{-\frac{5}{3}}$. This has been shown with a lot of previous data as presented in (Liu, 2019). But when it comes to the vertical wind all data have shown with the kinetic energy spectrum that for waves on the meso-β scale to be flat which is unexpected as it wouldn't follow a power-law.

Han-Li Liu has called for measurements to validate global circulation models in the mesosphere in (Liu, 2019). This thesis will help with that by using second order statistics to verify the kinetic energy power law for small scale waves, and by looking at how much of the kinetic energy in the mesosphere is due to small scale waves.

These calculations will be made by using the Multi-static, Multi-frequency Agile Radar for Investigation sof the Atmosphere (MMARIA) dataset which is located in Northern Norway. The measurements are made by a network of specular meteor radars (SMRs) which allows for significantly more measurements than a singular SMR. Using the MMARIA dataset and applying the spatial and temporal correlation function as (Vierinen et al., 2019) has done, these power laws in the energy spectrums can be shown even in the vertical direction with real data, and by applying a structure function the uncertainty will be reduced. By using the temporal correlation function phenomenon such as lunar tide, mountain waves, or planetary waves can be shown by how often the measurements correlate.

Another method of looking for power law is to use the second order structure function, which directly relates to the power law kinetic energy spectrum ($k^{-\frac{5}{3}}$ and $s^{\frac{2}{3}}$, where k is a spatial wave number and s is the related power law when using a structure function). This relation is shown in (Kolmogorov (1941)). The structure function has recently been used in this paper (Vierinen et al. (2019)) and shows that both the vertical- and horizontal wind follows the structure function ($s^{\frac{2}{3}}$ which corresponds to $k^{-\frac{5}{3}}$), even though in Vierinen et al. (2019) it's said that "With only a 24 hour data set, we only measure two periods of the 12 hour tide, which means that it is impossible to estimate error bars for these quantities and to judge if the values are meaningful or not." it is promising findings. Horizontal wind power law is also show in (Vierinen et al. (2019)).

This thesis will be one of the first studies to estimate annual variability of mesospheric kinetic energy using the new correlation function technique with a multi-year data set for the mesospheric winds.

2 Background

The mesosphere is the least known region in our atmosphere, the reason being is that it's very hard to get in situ measurements. The only way to make in situ measurements is to use sounding rockets which is an very expensive way of getting data.

Some of the reasons this is interesting to study is to see if previous studies and modeling of the mesosphere is accurate, as well as to verify previous studies on how the energy behaves in horizontal distance. For the vertical direction there haven't been any paper that has shown the kinetic energy to deplete as a power law.

2.1 Previous models of the mesosphere

One of the well known and most used models for the mesosphere is the Whole Atmosphere Community Climate Model (WACCM) which is a component of the Community Earth System Model which is a family of such models made at the National Center for Atmospheric Research (NCAR). WACCM is a model including everything from Earth's surface and up to the start of the thermosphere. There are a lot of different versions of this model including WACCM-X which is an extension going all the way up to altitudes around 500 km. Another well known and used model is the Mass-Spectrometer-Incoherent-Scatter (MSIS) which is an empirical model including temperature and densities from Earths surface up to the thermosphere, this model is mostly used for general studies of the whole atmosphere. Consequently, the model is not able to give detailed results for specific areas. All of these models and models in general have a need for higher resolution observations to account for small scale waves which not only affects the mesosphere but also the troposphere since the downward influence is note worthy (Liu, 2019). Waves such as gravity waves are noted to be important for circulation models in (Alexander et al., 2010) and (McLandress, 1998).

2.2 The difficulty of measuring the mesosphere using observation

Measuring the mesosphere is very difficult because the drag is too high for satellites to measure directly and it is not high enough for planes to get enough lift. Therefore, the only way of getting in situ measurements is by using rockets. When using rockets for measurements there are big limitations in resolution especially in the temporal domain but also in the spatial domain.

2.3 Previous measurements

The first time the kinetic energy of the horizontal wind in the atmosphere was shown with data was in 1984 in the paper by (Nastrom et al., 1984). This was done in the troposphere using research aircraft and the main result is in figure 2.1.

Using observations from a commercial aircraft in the troposphere and lower stratosphere, (Bacmeister et al., 1996) have showed that the power spectra of horizontal wind agrees relatively well with theoretical values. However, it also showed the vertical power spectrum to be flat for longer scales. This seemed to agree with WACCM as noted in (Liu, 2019). Further measurements from the mesosphere regarding both horizontal and vertical wind power spectra has yet to be done other than (Vierinen et al., 2019) which only had one day of data available.

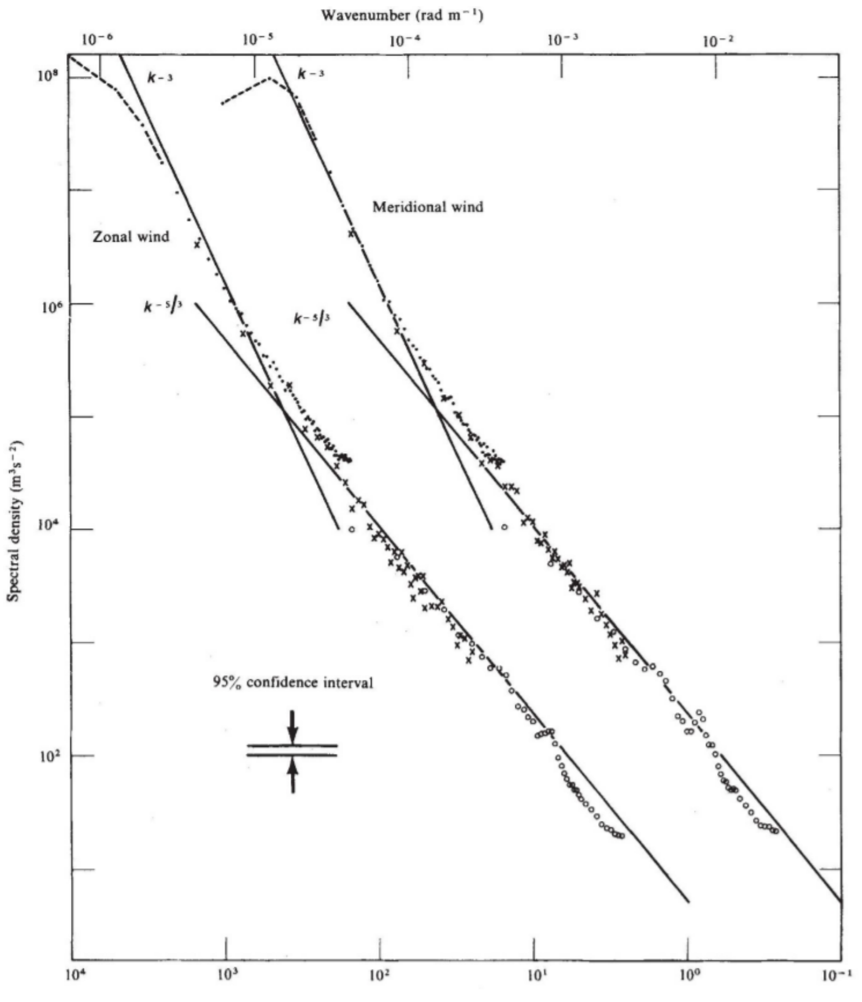

Figure 2.1: The first result of a power law in the troposphere, using measurements from research aircraft. Original paper (Nastrom et al., 1984)

2.4 A need for measurements

In the paper (Liu, 2019) a need for measurements in the mesospheric spatial domain was called out for. The reason for this need is for the circulation models to be more accurate. As of now these models have a lot of uncertainty when it comes to the smaller scale waves, due to the breaking or dissipation of smaller wave scales. This affects the mesosphere, but indeed also the lower atmosphere.

3

Theory and method

3.1 Temperature and altitude of the mesospause

One of the main reasons why the mesosphere is an interesting region to study is the big difference in the temperature in the solstice seasons, during the winter for high latitudes and all season in low latitudes the temperature in the mesospause is much smaller than in the summer in high latitudes. This low temperature has been shown by Sounding the Atmosphere by Broadband Emission Radiometry (SABER) measurements and in (Von Zahn et al., 1996) the summer in high latitudes was shown to be in the 83-89 km range as for other latitudes and time it was shown to be close to 100 km.

3.2 Waves effect on the mesosphere

A wave is a periodic disturbances of the wind field. Thus, a lot of waves affecting the wind field results in a complicated structure. Waves have three important features which are generation, propagation and dissipation. The main waves that affect the mesosphere are gravity waves, tides and planetary waves. These waves are generated outside the mesosphere, mostly from the lower atmosphere. From (Andrews and Mcintyre, 1976) it is know that waves will not interact with background atmosphere unless they are short-lived or dissipating.

3.2.1 Gravity waves

Gravity waves is a phenomena that start in the troposphere where they are produced by the equilibrium being displaced for reason such as winds going over a tall mountain, or from frontal systems. Gravity waves are small scale waves that travel from the troposphere through the stratosphere and dissipates or breaks in the mesosphere. This has a huge effect on the zonal wind in the mesosphere. Gravity waves are the main source of kinetic energy from the atmosphere below mesosphere.

3.2.2 Tides

Tides seen in the mesosphere are from gravity waves with a period of 1 day or a smaller fraction of a day. The two strongest tides are the diurnal tide with a period of 24 hours, the semi diurnal tide with a period of 12 hours which is the tide with the most effect in the middle to higher atmosphere, and is the most dominant tide in mesosphere (Pancheva et al., 2009). There are also smaller frequency waves like the 8 hour period tide as reported by (MJ et al., 1999) using temperature data from airglow imager. Even a 6 hour tide has been observed by (Smith et al., 2004).

3.2.3 Planetary waves

Planetary waves is a term for a lot of different wave phenomena. The general planetary wave are large scale disturbances with low wavenumbers. The main examples of planetary waves are Rossby waves observed in (Rossby, 1939). These waves are created due to the Coriolis effect in Earth's atmosphere.

3.3 Reynolds decomposition

Reynolds decomposition is a mathematical technique used to separate an actual value (u) into the expected value ($\bar{u}$) and the fluctuating value (u').

$$u = \bar{u} + u' \tag{3.1}$$

In this case it is used to find the higher frequency fluctuating wind. This is done by subtracting the low frequency expected value from the actual value. Each of the components are illustrated in figure 3.1

$$u' = u - \bar{u} \tag{3.2}$$

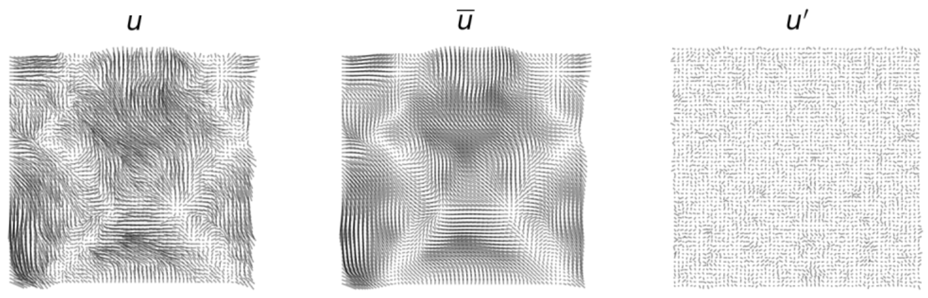

Figure 3.1: This figure illustrates Reynolds decomposition of wind, (From J. Vierinen)

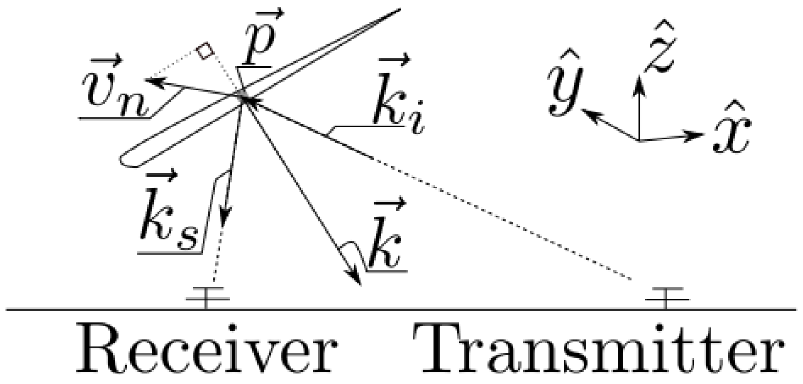

Figure 3.2: This figure illustrates how the Bragg vector is measured, (From J. Vierinen)

3.4 The technique for estimating the wind field velocity correlation function

To get the necessary measurements to estimate the wind field velocity correlation function, a network of five Specular Meteor Radars (SMRs) are used to get the measurements needed. The network allows for $10^4 - 10^5$ measurements every day depending on the time of year. In the data set used which is data from 2 years of measuring, there are 33.18 million measurements. Which can allow for up to 10^{15} pairs.

The SMRs measure the Doppler shift of the meteor trail f_D which is drifting with the neutral wind in the mesosphere. With SMR the Bragg vector ($\vec{k} = \vec{k}_s - \vec{k}_i$, showed in 3.2) can also be obtained using the incident and scattered wave vector($\vec{k_i}$, $\vec{k_s}$), this is possible because the SMRs measure the location of the meteor trail ($\vec{p}$). Using these measurements the wind vector($\vec{v_n}(t,\vec{p})$) can be represented by the given equation:

$$\vec{v_n}(t,\vec{p}) = u(t,\vec{p})\hat{x}(\vec{p}) + v(t,\vec{p})\hat{y}(\vec{p}) + w(t,\vec{p})\hat{z}(\vec{p}) \tag{3.3}$$

Here x, y and z represent the East, North and Up(ENU) coordinates where $\vec{p}$ is measured individually for each measurement.

Notation of $\vec{k}$ can be written as:

$$k^x = \vec{k} \cdot \hat{x}(\vec{p}) \qquad k^y = \vec{k} \cdot \hat{y}(\vec{p}) \qquad k^z = \vec{k} \cdot \hat{z}(\vec{p}) \tag{3.4}$$

The Doppler shift r of a measurement with error ξ is written as:

$$r = -\omega + \xi \qquad \omega = \vec{k} \cdot \vec{v} \tag{3.5}$$

Where $\vec{v}$ is the wave vector and $\vec{k}$ is the Bragg vector

$$r = -\omega + \xi \tag{3.6}$$

$$\omega = \vec{k} \cdot \vec{v} \tag{3.7}$$

$$r = u(t,\vec{p})k^x + v(t,\vec{p})k^y + w(t,\vec{p})k^z + \xi \tag{3.8}$$

ξ is assumed to be a zero mean independent normally distributed random variable. $u(t,\vec{p})$, $v(t,\vec{p})$ and $w(t,\vec{p})$ are the three components of the wind. Namely the East-West, North-South and Vertical components.

To estimate the wind field velocity correlation function we need measurements that we can correlate in time and position, given two measurements r_i and r_j which is Doppler shift measurement of wind velocity in the East, North, Up (ENU) coordinate system.

$$r_i = u(t_i,\vec{p_i})k_i^x + v(t_i,\vec{p_i})k_i^y + w(t_i,\vec{p_i})k_i^z + \xi_i \tag{3.9}$$

$$r_j = u(t_j,\vec{p_j})k_j^x + v(t_j,\vec{p_j})k_j^y + w(t_j,\vec{p_j})k_j^z + \xi_j \tag{3.10}$$

Take the expected value $\langle r_i r_j \rangle$, and all products including the zero mean ξ will be zero, assuming u, v and w are Gaussian random variables with some unknown correlation. Expressing the expected values of the product $\langle r_i r_j \rangle$ to some correlation function $G_{\alpha\beta}(\tau,\vec{s})$ where α and β are the 6 unique variations of u, v and w. τ is the temporal lag $\tau = t_i - t_j$ and $\vec{s}$ is the spatial displacement $\vec{s} = \vec{p_i} - \vec{p_j}$. The 6 unique variations of the correlation function G can be represented as:

$$G_{uu}(\tau,\vec{s}) = \left\langle u(t_i,\vec{p_i})u(t_j,\vec{p_j})\right\rangle \tag{3.11}$$
$$G_{vv}(\tau,\vec{s}) = \left\langle v(t_i,\vec{p_i})v(t_j,\vec{p_j})\right\rangle \tag{3.12}$$
$$G_{ww}(\tau,\vec{s}) = \left\langle w(t_i,\vec{p_i})w(t_j,\vec{p_j})\right\rangle \tag{3.13}$$
$$G_{uv}(\tau,\vec{s}) = \left\langle u(t_i,\vec{p_i})v(t_j,\vec{p_j})\right\rangle \tag{3.14}$$
$$G_{uw}(\tau,\vec{s}) = \left\langle u(t_i,\vec{p_i})w(t_j,\vec{p_j})\right\rangle \tag{3.15}$$
$$G_{vw}(\tau,\vec{s}) = \left\langle (t_i,\vec{p_i})w(t_j,\vec{p_j})\right\rangle \tag{3.16}$$

These $G_{\alpha\beta}(\tau,\vec{s})$ will be the wind field correlation function.

3.5 Techniques for measuring the MLT

There are no technique for measuring the full picture of the MLT, but putting a few different methods together gives a pretty good overall picture. Different methods used are: Active and passive ground based operations such as radar and lidar, rockets, satellite.

There are several different techniques for measuring the MLT using radar and lidar. Some of the more common are: Specular meteor radar (SMR), Medium frequency (MF), lidar and passive optical techniques.

Techniques when using satellites are the same as with ground based operation, but since the advantage of satellites is that there are no clouds in the way, lidars are quite common as well as limb scanning such as Sounding the Atmosphere with Broadband Emission Radiometry (SABER) and Microwave Limb Sounder (MLS) used in this study (Huang et al., 2006).

3.6 Temporal and spatial resolution with the different measuring techniques

Active ground based radar gives good local time variations as they can go on continuously. When using a network of SMR as in ((Vierinen et al., 2019), (Stober and Chau, 2015), (Chau et al., 2019)), it is possible to get one km vertical resolution and know the position and time of each measurement. This network of SMRs measures about $10^4 - 10^5$ measurements every day depending on how many radars in the network as well as the time of year. Thus, the temporal resolution is very good and the horizontal resolution is good for the local area.

More on meteor radars is given in section 3.7.

Satellites give near-global coverage, but spatial resolution will be quite high except for the use of limb scanning where the vertical resolution will be quite low (a few km, (Smith, 2012))

Rockets will give good vertical coverage (a few km, (Smith, 2012)) and good spatial resolution as it is in situ, but rocket operations takes long to plan and are very expensive which means the temporal resolution is bad.

Using lidars makes out the properties of the atmospheres using airglow emissions, but can only be used during near perfect conditions: night time and close to clear sky. This method is also limited in altitude range, since the emission only happens at certain altitudes around 87-100 km.

3.7 How a meteor radar measures the neutral wind velocity and temperature of the mesosphere

A SMR measures the Doppler shift of the meteor trail f_D which is drifting with the neutral wind in the mesosphere. SMR measures the location of the meteor trail ($\vec{p}$). Using these measurements the wind vector ($\vec{v_n}(t, \vec{p})$) can be represented by equation 3.3:

$$\vec{v_n}(t, \vec{p}) = u(t, \vec{p})\hat{x}(\vec{p}) + v(t, \vec{p})\hat{y}(\vec{p}) + w(t, \vec{p})\hat{z}(\vec{p})$$

Here x, y and z represent the East, North and Up(ENU) coordinates where $\vec{p}$ is measured individually for each measurement, this method is used in (Vierinen et al., 2019).

The temperature is estimated using the decay time of a meteor trail, with this method the accuracy is within 4-10 K as suppose to more than 10 K with previous methods. These estimations are shown to work at altitude with the most meteor detectability 86-92 km(Hocking, 1999). It is a big advantage to be able to use meteor radars to estimate the temperature as other methods (mentioned in (Smith, 2012)) for measuring the temperature rely on lidars which mostly can't be used during day time.

3.8 The expected horizontal spatial spectrum of kinetic energy of the mesosphere

The kinetic energy spectrum of the mesosphere describes how the kinetic energy decreases as the wave number increases. The kinetic energy in the horizontal direction is expected to decrease with k^{-3} for larger scale waves and $k^{-\frac{5}{3}}$ for smaller scale waves. Waves on the larger scale size has wavelength of 150-200 km or larger.

3.9 Structure function

The correlation function $G_{\alpha\beta}$ can be expressed with the use of a second order structure function assuming the wind field is a wide sense stationary and horizontally homogeneous random process (Kolmogorov, 1941). This structure function can be expressed with two pairs of meteor data, $u_\alpha(t_\alpha, \vec{p}_\alpha)$ and $u_\beta(t_\beta, \vec{p}_\beta)$ where $\vec{p}$ is the position and t is the time of the measurements and α and β denotes two different measurements. The structure function for the meteor data will be:

$$
\begin{aligned}
S_{\alpha\beta}(\tau, \vec{s}) &= \left\langle [u_\alpha(t_\alpha, \vec{p}_\alpha) - u_\beta(t_\beta, \vec{p}_\beta)] \right\rangle & (3.17)\\
&= \langle u_\alpha(t_\alpha, \vec{p}_\alpha)^2 \rangle + \langle u_\beta(t_\beta, \vec{p}_\beta)^2 \rangle - 2\langle u_\alpha(t_\alpha, \vec{p}_\alpha) u_\beta(t_\beta, \vec{p}_\beta) \rangle & (3.18)
\end{aligned}
$$

The two first components in 3.19 $\langle u_\alpha(t_\alpha, \vec{p}_\alpha)^2 \rangle + \langle u_\beta(t_\beta, \vec{p}_\beta)^2 \rangle$ will be the zero lag correlation functions $G_{\alpha\alpha}(0,0) + G_{\beta\beta}(0,0)$. And the last component $\langle u_\alpha(t_\alpha, \vec{p}_\alpha) u_\beta(t_\beta, \vec{p}_\beta) \rangle$ will be the cross correlation between the two measurements $G_{\alpha\beta}(t_\alpha - t_\beta, \vec{p}_\alpha - \vec{p}_\beta)$ or $G_{\alpha\beta}(\tau, \vec{s})$ Where τ is the temporal lag and $\vec{s}$ is the spatial lag. We get the final relation for two measurements:

$$
S_{\alpha\beta}(\tau, \vec{s}) = G_{\alpha\alpha}(0,0) + G_{\beta\beta}(0,0) - 2G_{\alpha\beta}(\tau, \vec{s}) \tag{3.19}
$$

and for the case of using the same component

$$
S_{\alpha\beta}(\tau, \vec{s}) = 2G_{\alpha\alpha}(0,0) - 2G_{\alpha\alpha}(\tau, \vec{s}) \tag{3.20}
$$

With these relations the structure function can be calculated from the correlation functions.

4 Results and discussion

In this chapter the results from the MMARIA Norway (Stober et al., 2018) data set will be presented and discussed. First some general statistics about the data set will be presented to get an overall idea of where and when the measurements occur. After that, the general picture about the data set the temporal correlation functions will be presented divided into summer and winter months which was decided to be June, July and August for summer and November, December and January for winter. Using the temporal correlation function and the spectral density we will figure out information about waves and tides, as well as to find a power law following the kinetic energy over time. For the spatial correlation function we use a second order structure function to find if the kinetic energy over space follows a power law.

4.1 Data

For this study, we used meteor radar measurements from the MMARIA Norway meteor radar network (Stober et al., 2018), which had 3 transmitters and 4 receivers. The transmitters were located in Alta, Andenes and Tromsø and the receiver stations in Alta, Andenes, Tromsø and Straumen. The Alta radar is owned and operated jointly by Nagoya University in Japan and the Tromsø Geophysical Observatory (TGO). The Tromsø radar is owned and operated jointly by the Japanese National Institute for Polar Research (NIPR) and TGO. The Andenes and Straumen radar sites are owned and operated by the Institute

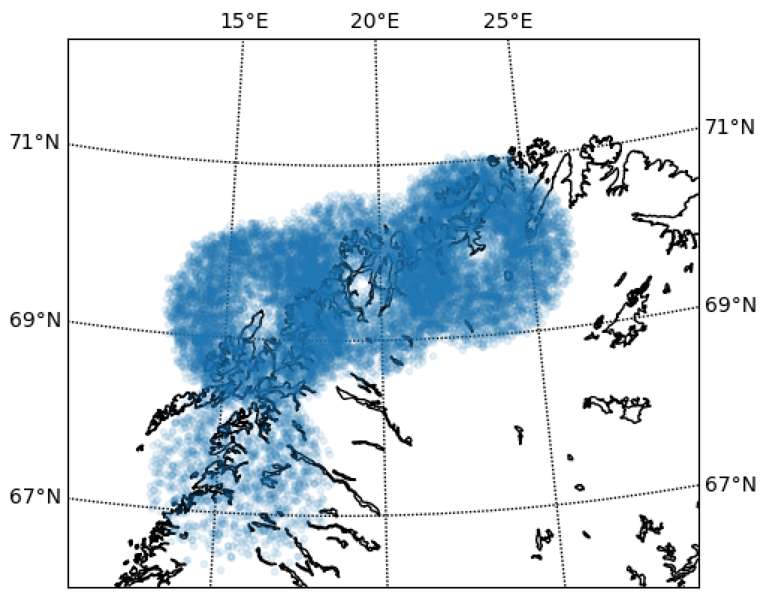

Figure 4.1: This is a plot of the measurements based on longitude and latitude showing where the measurements are. Elevation cutoff is 37° above the horizon. Only two days of data is used.

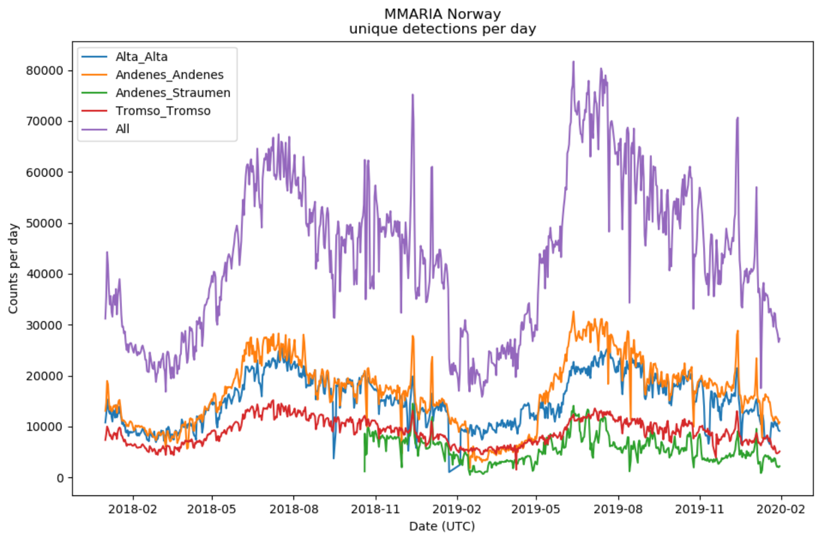

Figure 4.2: Daily number of meteor measurements, separated by transmitter and receivers.

of Atmospheric Physics in Kühlungsborn, Germany.

The benefit of using a network of radars, is that we obtain more meteor counts, a larger geographic coverage, and a diversity of Bragg scattering k-vectors. The combination of the higher meteor count and diversity of Bragg scattering k-vectors results in statistical better estimates of the correlation function of the kinetic energy. Larger geographic coverage allows the distribution of kinetic energy across a larger range of horizontal spatial scales to be studied, as the longest spatial lag is defined by the longest distance between observed meteors.

The geographic coverage of the meteor radar network is shown in Figure 4.1. From this figure we can see how big of a horizontal coverage we have. We have about 15 degrees in the zonal direction which is some where around 1000 km, and in the meridional direction we have about 5 degrees which is around 500 km. The reason the south most, Straumen, has such few measurements is due it only being a receive-only station. The radii of the circles around the receivers are limited due to only selecting meteor detections with elevation angles higher than 37° above the horizon in order to select only high quality measurements. This figure only show the coverage of the data it is only a plot of a few days of data. The Andenes-Straumen link was only activated in the end of 2018.

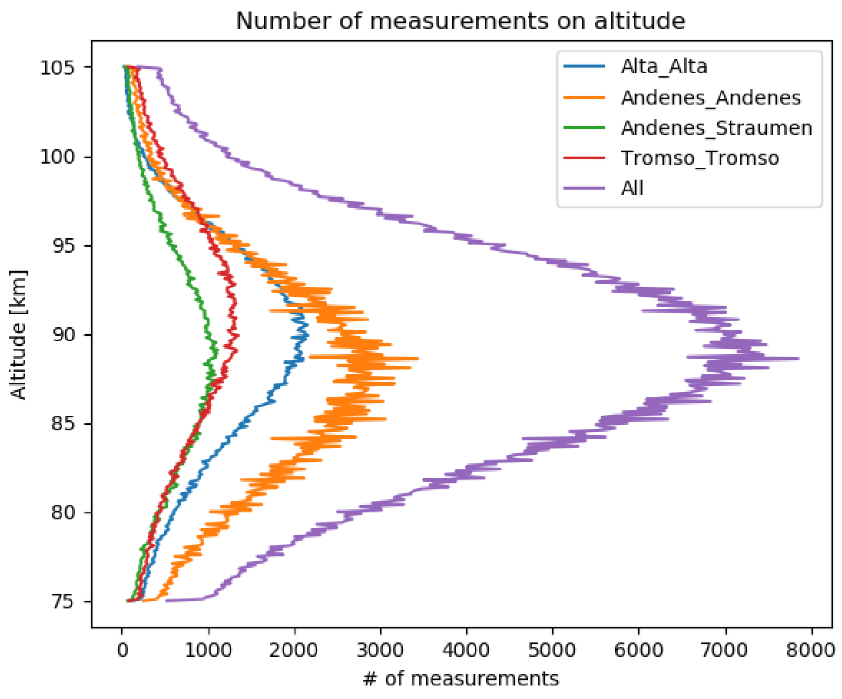

Figure 4.3: The number of measurements on altitude, for one month during the winter 2019.

The number of specular meteors trail echoes per day is shown in Figure 4.2. The total number of meteors per day is maximized during the fall, when Norway is on the axis facing towards Earths orbital direction. When this is the case the Earth crashes into more meteors than it would on any other time of year. The short duration spikes in meteor count can be attributed to meteor showers. The biggest spike is due to a meteor shower called Geminids, and is caused by the object "3200 Phaethon" which is not a comet. The MMARIA data set from January 2018 to February 2020 contains 33.18 million meteor measurements in total.

Figure 4.4 shows how many measurements per 30 minutes at time of day, the timing is in UTC. The reason there is a peak in measurements at around 4-UTC is because that is when Norway is facing Earths orbital direction and are "crashing" into meteors similar to 4.2, at the nadir in the figure which will be 12 hours later, will be when Norway is on the opposite side of Earths orbital direction.

The altitude range shown in 4.3 shows that all measurements are in the range

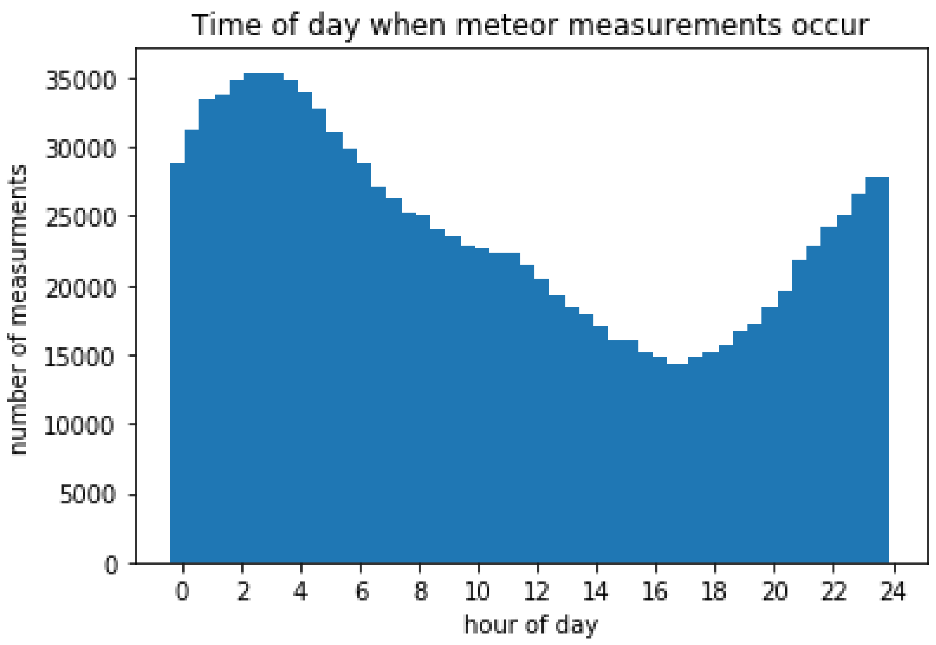

Figure 4.4: This figure shows the distribution of measurements on time of day during 5 days in winter 2019.

70-110 km, with most measurements around the 90 km altitude. The reason for this specific altitude range is due to the atmospheric density. At around 110 km the atmosphere is dense enough for meteors to start burning up, and most meteors will be burnt up at around 70 km.theerfore there are very few if any measurements at a lower altitude. The histogram of meteor detections as a function of altitude determines the altitude range at which measurements of kinetic energy can be made using specular meteor trail measurements. The altitude region where the count of meteors is highest is the region where the highest quality measurements can be made.

Given a normal distribution of meteors in a space it is logical that there will be an exponential increase in pairs as the lag gets bigger, the same applies for a time period. As showed in 4.5 we can tell that the horizontal and temporal lag increase the amount of pairs increase as expected. From this figure the spatial and temporal limitations can be seen, as of (10 km, 10 sec) the amount of measurement is sufficient. This figure is also only an example from the altitude with the most measurements, all the other altitudes looks similar and often identical.

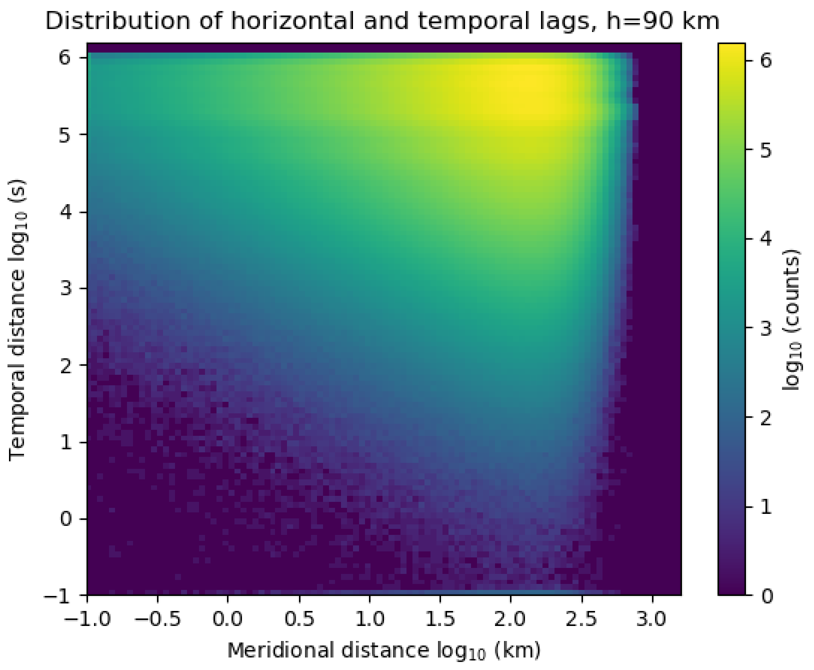

Figure 4.5: This 2D histogram plot shows the number of pairs with given horizontal and temporal lag for 90 km altitude.

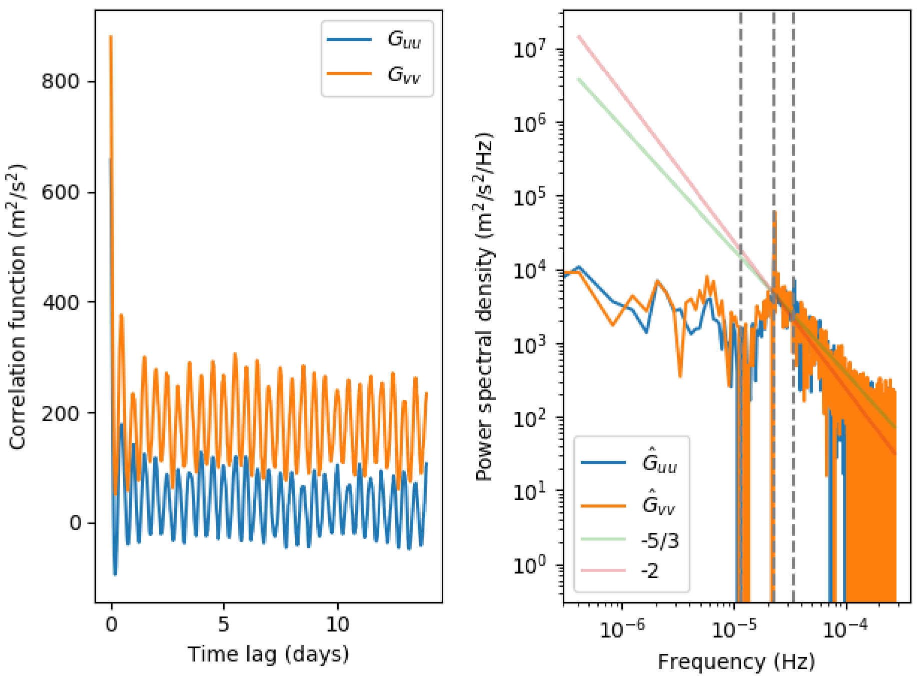

Figure 4.6: This figure uses data from summer months of 2018, 2019. The left plot in this figure shows Temporal ACF with up to 14 days of lag. And the right plot shows spectral density compared with power law of -5/3 and -2. G_{uu} represents the zonal direction and G_{vv} represents the meridional direction.

4.2 Seasonal variation in Temporal autocorrelation function

The temporal autocorrelation function is calculated by using equation 3.11 on the form $G_{\alpha\beta}(\Delta\tau, \Delta\vec{p})$ by using a certain temporal resolution also needs to be capped as there would be 10^{16} pairs if there were no cap. The temporal lag resolution is set to 1800 s and it runs for lags up to 14 days. The displacement in position is set to be 100 km, meaning that any two measurements of wind need to be separated by less than 100 km in horizontal distance. The altitude range is set to only take measurements between 85 and 95 km which is the range where there is the most measurements. The time window is limited to differentiate summer 4.6 and winter 4.7 months. Summer months are selected as May, June and July. Winter months are November, December and January. The measurement of temporal autocorrelation function is averaged over the whole dataset for winter and summer months.

Figures 4.6 and 4.7 show the temporal autocorrelation functions of the zonal

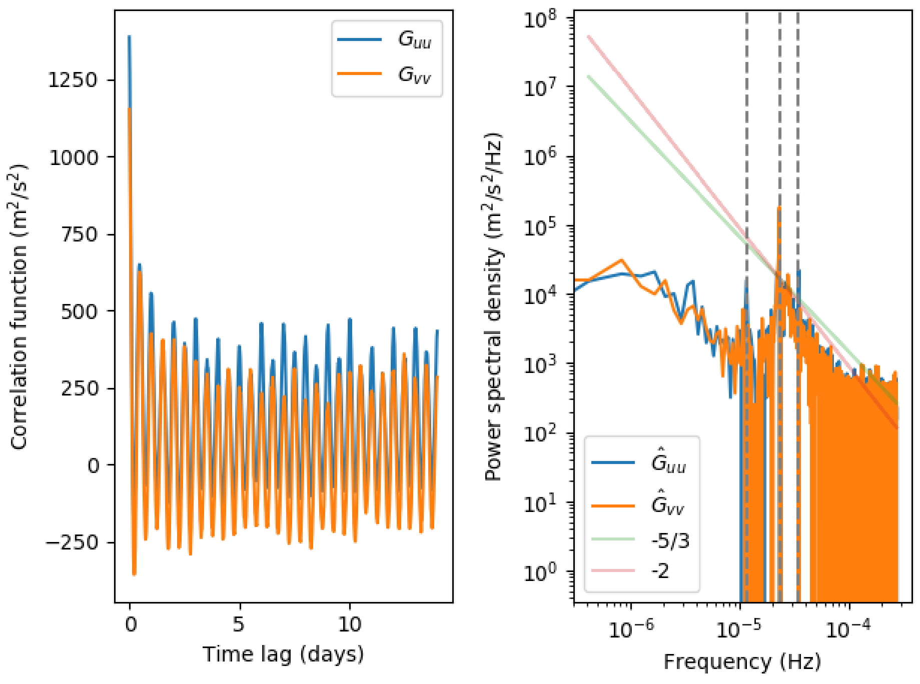

Figure 4.7: This figure uses data from winter months of 2018, 2019 and 2020. The left plot in this figure shows temporal autocorrelation function with up to 14 days of lag. And the right plot shows spectral density compared with power law of -5/3 and -2. G_{uu} represents the zonal direction and G_{vv} represents the meridional direction.

and meridional wind G_{uu}, G_{vv} for the summer and winter months, respectively. The three main features of G_{uu}, G_{vv} are: 1) the 12 hour tide, which can be seen as a sinusoidal oscillation with a 12 hour period; 2) a spike at short < 12 hour temporal lags, which represents kinetic energy in the mesospheric fluid that does not have long correlations. These are mostly due to short period gravity waves; 3) There are long time scale correlations which manifest has a nearly constant non-zero mean value for the correlation function, this component is attributed to planetary waves.

The temporal autocorrelation function can be expressed like a sinusoidal function with three components assuming these three main features are the only periodic waves effecting the temporal autocorrelation function and that the small scale waves kinetic energy is constant. This simplified function would be on the form:

$$G_{\alpha\alpha}(\tau) = A_0 \exp(-\gamma\tau) + A_1 \cos(2\pi f_0 \tau) + A_2$$

By comparing Figures 4.7 and 4.6 to the equation 4.1. A we can estimate the amount of kinetic energy that comes from what type of waves. A_0 will be the small scale waves only happening at the first peak, A_1 will be the semi diurnal tide meaning $f_0 = 2.3 \cdot 10^{-5}$ and A_2 will be the planetary waves. A simplified version of the temporal autocorrelation function is showen in figure 4.8.

Table 4.1: This table describes how many percent of kinetic energy (KE) the tide and the different waves contribute. A_0 is planetary waves, A_1 is semi diurnal tide and A_2 is gravity waves.

	A_0	A_1	A_2	A_0	A_1	A_2
	[%]	[%]	[%]	$[\frac{m^2}{s^2}]$	$[\frac{m^2}{s^2}]$	$[\frac{m^2}{s^2}]$
Summer zonal (G_{uu})	84	11	5	630	85	40
Winter zonal (G_{uu})	68	21	11	950	300	150
Summer meridional (G_{vv})	67	13	20	600	120	180
Winter meridional (G_{vv})	72	24	4	870	290	50

The kinetic energy in the mesosphere is mostly due to small scale waves which is mainly gravity waves. As we can see from 4.1 they contribute about 70% or more. For the semi diurnal tide which seem to contribute around 10% during the summer and around 20% in the winter. The planetary waves has a big seasonal dependence in the meridional direction with a contribution of 20% in the summer and only 4% in the winter.

From Figure 4.7 and 4.6 we can also tell that there is a clear difference in spectral density for the summer and winter months. For Figure 4.6 which is for summer months, it is unclear whether the spectral density follows a power law of -5/3 or -2. And for the winter 4.7 the spectral density clearly follow a power law of <-2. The reason for this is unclear. The marked lines show where the 24 hour diurnal tide, the 12 hour semi diurnal tide and the 8 hour tide occurs. The semi diurnal tide is by far the biggest spike.

4.3 Horizontal autocorrelation funciton and structure function

The horizontal autocorrelation function is calculated in the same manner as the temporal autocorrelation function. The lag resolution is however based on the horizontal distance from each measurement. This horizontal lag resolution is set to be 25 km and it runs all the way up to 500 km. The measurements are taken from the altitude range 85-95 km and the temporal range from summer months is shown in figure 4.10 and from winter months in figure 4.11. In the right plot in figures 4.10 and 4.11 the second order structure function using

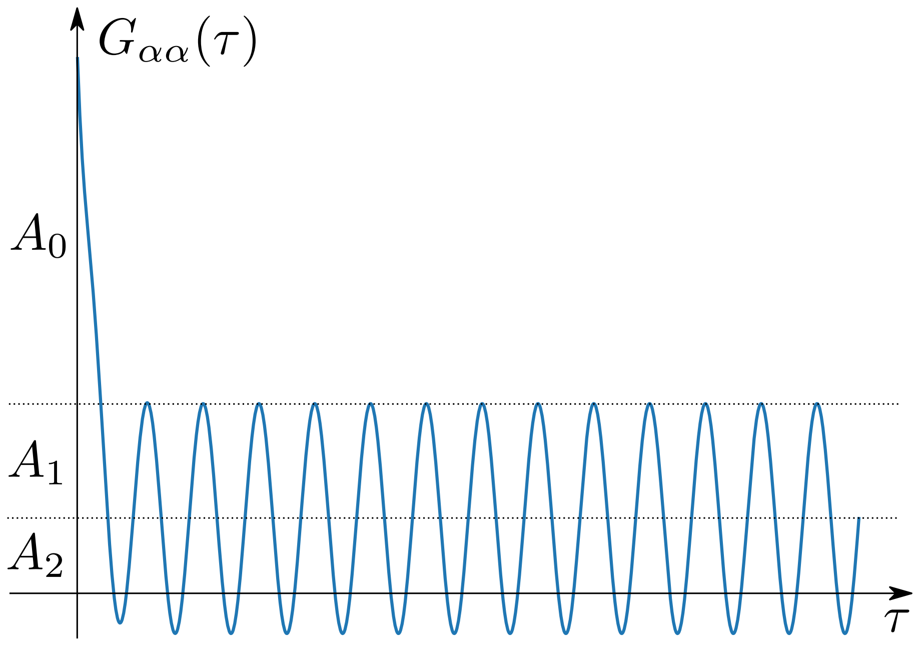

Figure 4.8: Simplified version of the temporal autocorrelation function.

the correlation function calculation shows that the structure function follows a power law if $s^{\frac{2}{3}}$ which is what is expected as this relates to the power law of $k^{-\frac{5}{3}}$ in the auto correlation function (Kolmogorov, 1941). These plots are the first plots showing a power law of $k^{-\frac{5}{3}}$ for the decrease in kinetic energy for small scale waves in the mesosphere with a resolution up to 500 km.

There are some clear seasonal differences in the horizontal wind. In the winter there is much more energy both for small and large scales waves. As well as for the waves responding to the 350-400 km have a big spike in the winter meridional wind. The structure function also indicates this increase around 350-400 km of horizontal lag. This peak could be an artefact.

By simplifying the horizontal autocorrelation function in figures 4.10 and 4.11 into an exponential function featuring the two main features, assuming the large scale waves are near constant in our 500 km horizontal lag and the small scale waves. The function can be expressed like this:

$$G_{\alpha\alpha}(s) = A_0 \exp(-ks) + A_1$$

Where A_0 will related to the small scale waves and A_1 will related to the large scale waves like planetary waves as these large scale waves will always

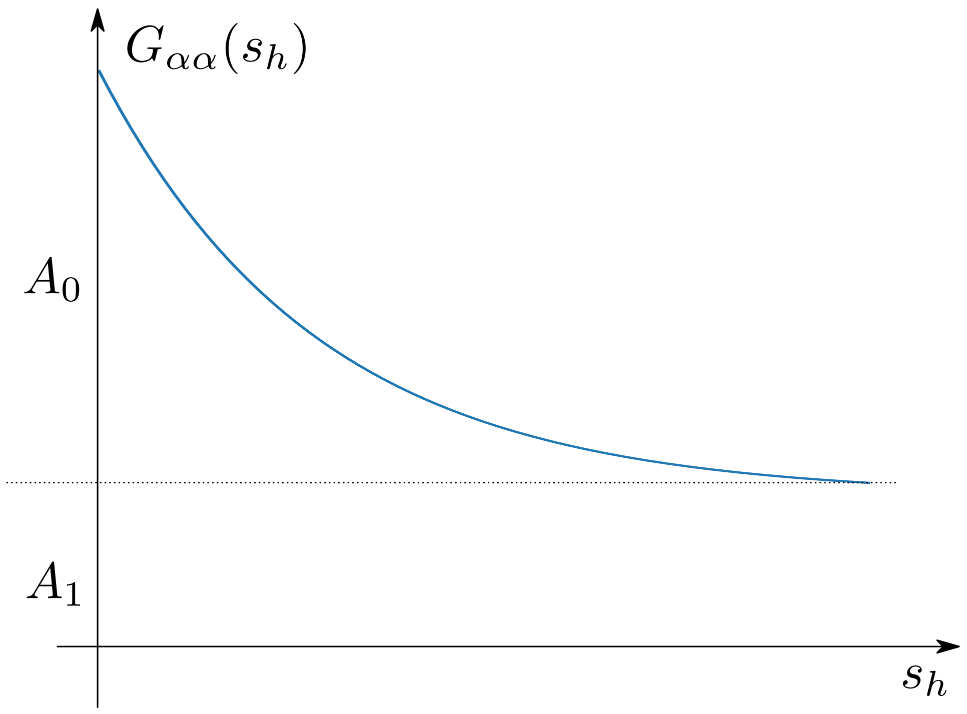

Figure 4.9: Simplified version of the horizontal autocorrelation function.

correlate equally much over 500 km distance. By using this equation the seasonal differences are easier to express. This simplification is illustrated in figure 4.9

Table 4.2 illustrates how much of the kinetic energy is contributed by planetary waves and by gravity waves separated by direction and season. In the summer planetary waves have a much greater impact on the meridional wind than the zonal wind as well as gravity having a smaller significance in the meridional wind. In the winter the impact seem to be pretty much the same in both the meridional and zonal direction for both planetary and gravity waves.

The vertical wind might not seem as important as the horizontal wind since the kinetic energy is a lot smaller it is still very important as it affects the horizontal wind due to the momentum flux from small scale waves (Liu, 2019).

Table 4.2: This table describes how many percent of kinetic energy (KE) the different waves contribute in the horizontal wind. A_1 is planetary waves, and A_0 is gravity waves.

	A_0 [%]	A_1 [%]	A_0 $[\frac{m^2}{s^2}]$	A_1 $[\frac{m^2}{s^2}]$
Summer zonal (G_{uu})	52	48	370	320
Winter zonal (G_{uu})	43	57	550	720
Summer meridional (G_{vv})	31	69	250	560
Winter meridional (G_{vv})	43	57	470	630

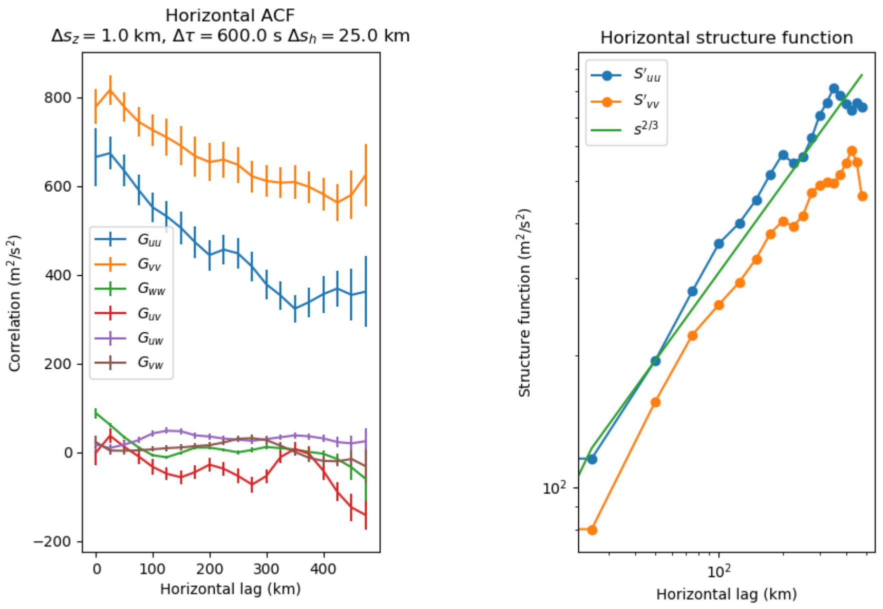

Figure 4.10: The left plot shows Horizontal autocorrelation function with with up to 500 km spatial lag with 25 km resolution, during the summer 2018 and 2019. The right plot shows the structure function corresponding to the left autocorrelation function. G_{uu} represents the zonal direction and G_{vv} represents the meridional direction.

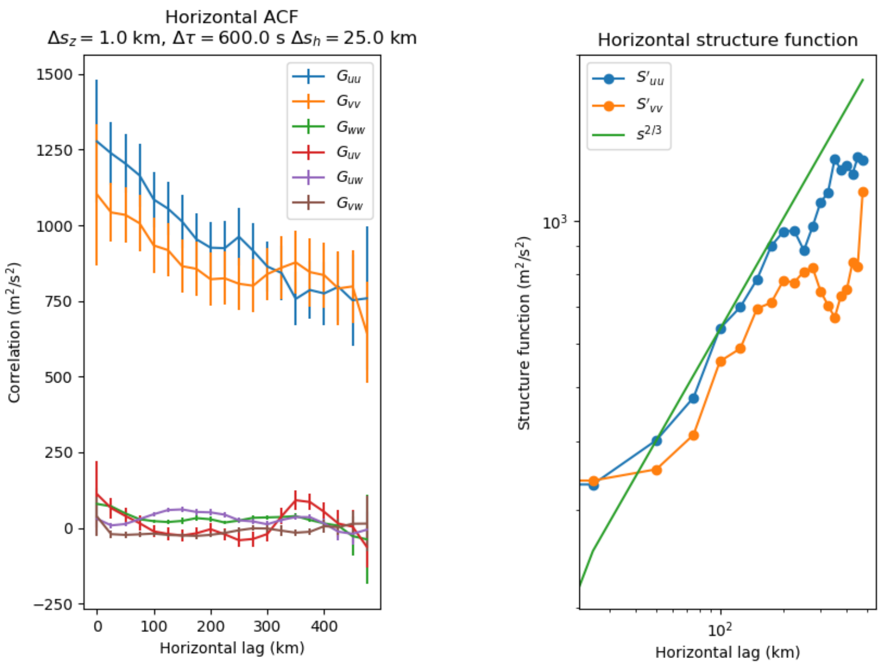

Figure 4.11: The left plot shows Horizontal autocorrelation function with with up to 500 km spatial lag with 25 km resolution, during the winter 2018, 2019 and 2020. The right plot shows the structure function corresponding to the left autocorrelation function. G_{uu} represents the zonal direction and G_{vv} represents the meridional direction.

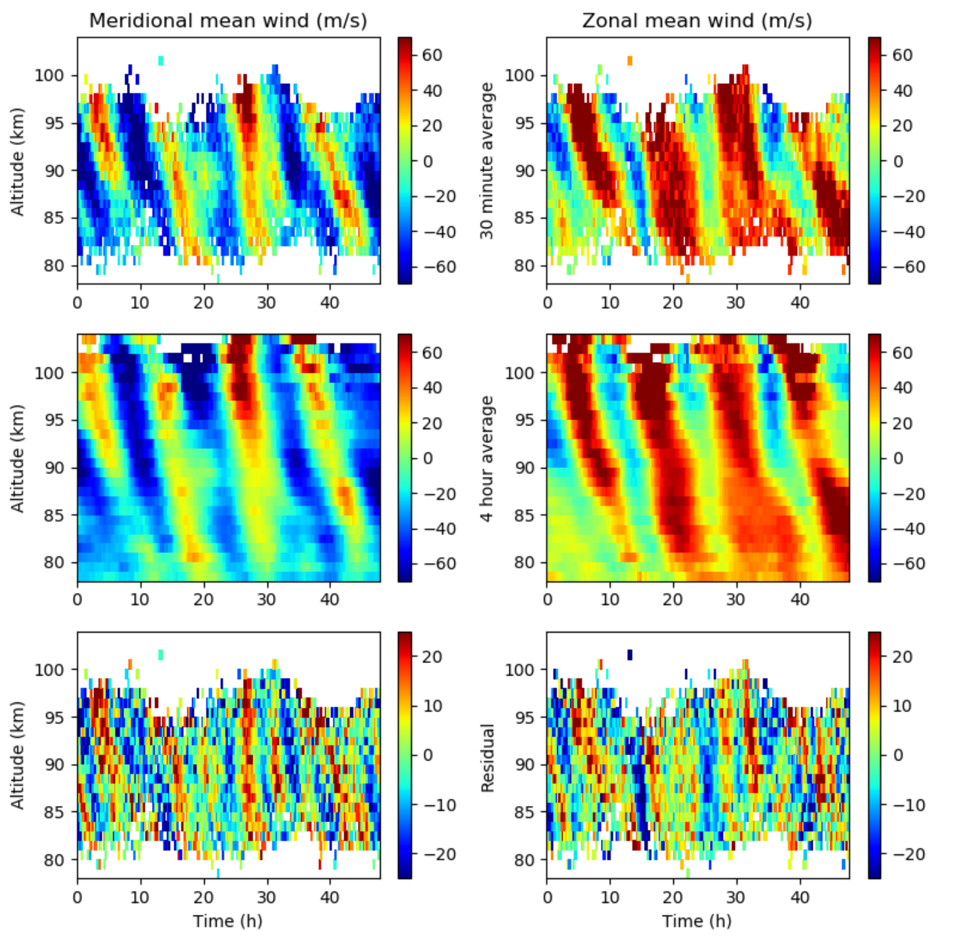

Figure 4.12: In this figure the 30 min mean wind, 4 hour mean wind and the 30 min - 4 hour fluctuating component is shown, the data is from January 2019.

4.4 The mean wind in the mesosphere

From Figure 4.12 the 30 min mean wind is plotted on top, as well as the 4 hour mean wind. Using Reynolds decomposition and letting the 30 min mean wind be $\bar{u}$ and the 4 hour mean wind be u and using equation 3.2. We get the less than 4 hour fluctuating wind showing that there are periodic waves smaller than 4 hours in period. These fluctuations are most likely large horizontal wavelength gravity waves.

4.5 Code problems/fixes

The first issue with the data was that there was duplicate measurements at the start of each month which was fixed by calling only the unique measurements.

Some other problems with the measurement was some noise appearing at 0 lag, which was avoided by filtering the 0 lag data away.

Since there was a lot of data to process and on a different format then previous data. A method for reading the data for use in pre-existing code was needed, which was simply reading the data in a slightly different way.

As the file received was on a day by day basis, a way of reading the data between certain temporal frames, as well as having a easier way to work with the data was made. And this also made the processing faster.

Reading in all of the data was a problem for the laptop uses for these programming, a fix was to read the data in at a shorter time period at a time, such as 7 days at a time, or longer depending on the use. As my laptop was also quite slow some of the data ended up being processed on a computer server.

The code used is in an open repository in Github. https://github.com/OleNordaunet/cfi. The addional software development effort to allow large datasets to be processed using the CFI method has been a join effort between me and the supervisor for this project, Juha Vierinen.

Since there was a lot of data to process and on a different format then previous data. A method for reading the data for use in pre-existing code was needed, which was simply reading the data in a slightly different way.

5 Conclusions

In this thesis a method of calculating the distribution of kinetic energy in the mesosphere is presented, as well as the seasonal differences. This method is using data from Multi-static, Multi-frequency Agile Radar for Investigations of the Atmosphere (MMARIA) to find the temporal and spatial correlation functions.

5.1 Our findings

Using the temporal correlation function and its power spectra we found that the kinetic energy mainly oscillates with the semi diurnal tide, but the diurnal and 8-hour tides are also noticeable. For the distribution of kinetic energy the small scale waves were found to be the most significant with a seasonal variation. The semi diurnal tide however seem to have a clear seasonal variance in both the zonal and meridional direction. The planetary wave has the strongest effect during summer in the meridional direction, for the zonal direction during summer the planetary wave has much lower significance. And for the winter the significance is strongest in the zonal direction, and for the meridional direction during winter the significance is low.

The power spectra of the temporal autocorrolation function shows that for small scale waves energy reduces as a power law. There is also a seasonal difference in this power spectra for small scale waves, in the summer the power spectra seems to be $k^{-\frac{5}{3}}$ but during the winter the power spectra seem to have a faster decrease in energy following k^{-2}.

The spatial correlation function is used to look for a power law describing the small scale waves decreasing in energy $k^{-\frac{5}{3}}$, and the relation to the second order structure function $s^{\frac{2}{3}}$ is also used to reduce uncertainty.

5.2 Suggestions for further work

There is a lot that can be done using the MMARIA dataset, but for the matter relevant to this thesis there is a finite amount to mention. The less than 4 hour periodic waves found in the fluctuating wind can be explored.

For the temporal energy specter the exact power law can be explored. It might be easier to find using a smaller temporal resolution.

The vertical auto correlation function can be found and related to a power law using the added code the result for the vertical auto correlation function can be shown as well as the structure function.

A
Source code

The main source code used in this thesis is the *cfi_dac.py* which is the base of all calculations done in the other scripts. The *sfi2d_hor.py* which calculates the mean wind is also important as some of the scripts relies on calculations of the mean wind during the temporal range the given code are using. *mmaria_read.py* is the last important base code which makes the dataset much easier to work with.

The three next following scripts are for making the calculations for plotting and are based on the previous main codes. The last remaining scrips are for making the plots.

Listing A.1: *cfi_dac.py*; Core routines for correlation function inversion.

```
#!/usr/bin/env python
#
# Correlation Function Inversion
#
# find pairs using divide and conquer
#
# Estimating the mesospheric neutral wind correlation
    function with different spatial and temporal lags
# using meteor radar measurements.
#
# Juha Vierinen, 2018
#
```

```
import numpy as n
import h5py
import matplotlib.pyplot as plt
import itertools
import scipy.misc as sm
import scipy.signal as ss
import scipy.interpolate as si
import scipy.optimize as sio
import traceback
import sys

import geoid_const as gc

n.set_printoptions(precision=3)
# for parallel processing.
#from mpi4py import MPI
#comm = MPI.COMM_WORLD

# constants,
# TBD add altitude and latitude dependence
# (won't make a huge difference, but just to be
    complete).
latdeg2km=gc.latdeg2km#111.321
londeg2km=gc.londeg2km#65.122785

sf_names=["uu","vv","ww","uv","uw","vw"]
# end constants

def vel(t,alt,lats,lons,times,rgs,v,dt,dh):
    '''
     evaluate mean wind without gradients.
     tbd: implement better interpolation and linear
         gradients for mean wind.
    '''
    ti=n.array(n.round((t-times[0])/dt),dtype=n.int)
    hi=n.array(n.floor((alt-rgs[0])/dh),dtype=n.int)
    hi[hi>(v.shape[2]-1)]=v.shape[2]-1
    ti[ti<0]=0
    ti[ti>(v.shape[1]-1)]=(v.shape[1]-1)
    return(v[0,ti,hi],v[1,ti,hi])

# to use with mmaria file? h=mmaria_read().read_data(t0
    ,t1) but .value doesnt work here, must change this
    script?
def get_meas(meas_file="res/
    simone_nov2018_multilink_juha_30min_1000m.h5",
             mean_rem=False,
             plot_dops=False,
             dcos_thresh=0.8,
             mean_wind_file="res/mean_wind_4h.h5",
             data='h5file'):
    '''
    read measurements, subtract mean wind if requested.
    the mean wind is whatever is in the file
    '''

    if data=='h5file':
        h=h5py.File(meas_file,"r")

        t=n.copy(h["t"].value)
        lats=n.copy(h["lats"].value)
        lons=n.copy(h["lons"].value)
        heights=n.copy(h["heights"].value)
        braggs=n.copy(h["braggs"].value)
        dops=n.copy(h["dops"].value)
        if "dcos" in h.keys():
            dcoss=n.copy(h["dcos"].value)
        else:
            dcoss=n.zeros([len(t),2])


    else:        #Generaly only for mmaria_read (for now
    )
        h=meas_file
        t=n.copy(h["t"])
        lats=n.copy(h["lats"])
        lons=n.copy(h["lons"])
        heights=n.copy(h["heights"])
        heights=heights/1000
        braggs=n.copy(h["braggs"])
        dops=n.copy(h["dops"])
        if "dcos" in h.keys():
            dcoss=n.copy(h["dcos"])
        else:
            dcoss=n.zeros([len(t),2])
    # dcos thresh
    dcos2=n.sqrt(dcoss[:,0]**2.0+dcoss[:,1]**2.0)
    ok_idx=n.where(dcos2 < dcos_thresh)[0]
    if outlier_filter:
        dc2=n.linspace(0,1.0,num=100)
#           plt.axvline(dcos_thresh)
        ok_idx=n.where( ((n.abs(dops) < (n.abs(dcos2)
            *35+15))) & (dcos2 < dcos_thresh)  )[0]

    if plot_outlier_filter:

        plt.plot(dcos2,n.abs(dops),".",label="All␣
            measurements")
        plt.plot(dcos2[ok_idx],n.abs(dops[ok_idx]),".",
            label="Filtered␣measurements")
        plt.plot(dc2,35.0*dc2+15)
        plt.xlabel("Magnitude␣of␣Doppler␣velocity␣(m/s)
            ")
        plt.ylabel("Direction␣cosine")
        plt.show()


    t=t[ok_idx]
    lats=lats[ok_idx]
    lons=lons[ok_idx]
    heights=heights[ok_idx]
    braggs=braggs[ok_idx,:]
    dops=dops[ok_idx]
    dcoss=dcoss[ok_idx,:]

    # remove mean wind (high-pass filter)
    if mean_rem:
        hm=h5py.File(mean_wind_file,"r")
        # grid
        times=n.copy(hm["times"].value)
        rgs=n.copy(hm["rgs"].value)
        v=n.copy(hm["v"].value)
        dt=n.copy(hm["dt"].value)
        print(dt)
        dh=n.copy(hm["dh"].value)
        mlat0=n.copy(hm["lat0"].value)
        mlon0=n.copy(hm["lon0"].value)
        hm.close()
```

```
# interpolate zonal and merid wind
vu,vv,dudy,dvdy,dudx,dvdx=vel(t,heights,lats,
    lons,times,rgs,v,dt,dh)

# don't remove gradients
mean_dops=vu*braggs[:,0]+vv*braggs[:,1]+\
              dudy*(lats-mlat0)*gc.latdeg2km*
                  braggs[:,0]+\
              dvdy*(lats-mlat0)*gc.latdeg2km*
                  braggs[:,1]+\
              dudx*(lons-mlon0)*gc.londeg2km*
                  braggs[:,0]+\
              dvdx*(lons-mlon0)*gc.londeg2km*
                  braggs[:,1]


# residual vel after removing mean horizontal
    wind
# (negative sign in analysis)
dopsp = dops + mean_dops/2.0/n.pi
stdev_est=n.nanmedian(n.abs(n.nanmean(dopsp)-
    dopsp))
stdev_est2=n.nanmedian(n.abs(n.nanmean(dops)-
    dops))

if plot_dops:
     plt.plot(dops,".",label="orig")
     plt.plot(dopsp,".",label="hp")
     print(stdev_est)
     print(stdev_est2)
     plt.axhline(5*stdev_est)
     plt.axhline(-5*stdev_est)
     plt.legend()
     plt.show()
resid=n.abs(n.nanmean(dopsp)-dopsp)
ok_idx=n.where(n.isfinite(dopsp) & (resid < 5*
    stdev_est) )[0]

t=t[ok_idx]
lats=lats[ok_idx]
lons=lons[ok_idx]
heights=heights[ok_idx]
```

```
        braggs=braggs[ok_idx ,:]
        dops=dopsp[ok_idx]
        dcoss=dcoss[ok_idx ,:]
    if data=='file':
        h.close()
    return({"t":t,"lats":lats,"lons":lons,"heights":
        heights,"braggs":braggs,"dops":dops,"dcoss":
        dcoss})


def cfi(m,
        h0=90,       # The height that we want (km)
        dh=2,        # delta height (how much the height
             can differ from h0)
        ds_z=1.0,    # delta s_z how much the vertical
            lag can differ from one another
                     # vertical lag is s_z +/- ds_z/2
        s_z=0.0,     # vertical component of the spatial
             lag (s_z)
        s_x=0.0,     # east-west component of the
            spatial lag
        ds_x=1.0,    # delta s_x lag-resolution
        s_y=0.0,     # north-south component of the
            spatial lag
        ds_y=0.0,    # lag resolution
        s_h=0.0,     # horizontal distance of the
            spatial lag (used instead of s_x and s_y if
            horizontal_dist=True)
        ds_h=100.0, # lag-resolution
        tau=0.0,     # temporal lag
        dtau=300.0, # temporal lag resolution (lag can
            be tau +/- dtau/2)
        horizontal_dist=False, # do we use spatial lag
            specified using horizontal distance
                               # if true, then s_h
                                   specifies the
                                   horizontal lag,
                                   otherwise
                               # s_x and s_y specify it
                                   .
        hour_of_day=0.0,     # hour of day
        dhour_of_day=48.0,   # length of a time bin
        min_ds_h=5.0,
        min_dt=30.0,
        plot_thist=False):
   '''
    Calculate various lags. Use tree-like sorting of
        measurements to reduce the time
    to find pairs of measurements (not 100% tested).
    '''

    # this is where we read measurements from the
        measurement object.
    t=m["t"]
    heights=m["heights"]
    dops=m["dops"]
    lats=m["lats"]
    lons=m["lons"]
    braggs=m["braggs"]

    # Figure out hour of day (UTC)
    hod=n.mod(t/3600.0,24.0)

    # (idx_for_dimension_0, idx_for_dimension_1, ...)
    t_idx=n.where( (n.abs(hod-hour_of_day)<=(
        dhour_of_day/2.0)) | (n.abs(hod-24-hour_of_day)
        <=(dhour_of_day/2.0)) | (n.abs(hod+24-
        hour_of_day)<=(dhour_of_day/2.0)) )[0]

    # select only the subset of measurements
    t=t[t_idx]
    heights=heights[t_idx]
    dops=dops[t_idx]
    lats=lats[t_idx]
    lons=lons[t_idx]
    braggs=braggs[t_idx,:]

    acf=n.zeros(6)
    err=n.zeros(6)

    pairs=[]
    pair_dict={}

    tods=[]
    taus=[]
    s_xs=[]
```

```
    s_ys=[]
    s_zs=[]
    s_hs=[]

    hor_dists=[]

    n_times=int((n.max(t)-n.min(t))/(dtau))
    t0=n.min(t)

    for i in range(n_times):
        it0=i*dtau+t0
        if i == (n_times-1):
            it1=n.max(t)
        else:
            it1=i*dtau+t0+2*dtau

        # filter heights
        idx0=n.where( (heights > (h0-dh*0.5)) & (
           heights < (h0+dh*0.5)) & (t>it0) & (t<it1) )
           [0]
        idx1=n.where( (heights > (h0+s_z-0.5*dh)) & (
           heights < (h0+s_z+0.5*dh)) & (t > (it0+tau))
            & (t< (it1+tau)) )[0]

        if False:
            plt.plot(t[idx0],heights[idx0],"+")
            plt.plot(t[idx1],heights[idx1],"o")
            plt.xlim([n.min(t),n.max(t)])
            plt.show()

#          print("%d/%d s_z %1.2f h0 %1.2f h1 %1.2f tau
   %1.2f"%(i,n_times,s_z,n.mean(heights[idx0]),n.mean(
   heights[idx1]),tau))

        for ki,k in enumerate(idx0):
            lat0=lats[k]
            lon0=lons[k]
            mt0=t[k]
            hg0=heights[k]

            if horizontal_dist:
                dist_filter = (n.abs(n.sqrt( (latdeg2km
                   *(lats[idx1]-lat0))**2.0 + (
                   londeg2km*(lons[idx1]-lon0))**2.0 )-
                   s_h) < ds_h/2.0)
            else:
                dist_filter = (n.abs((latdeg2km*(lats[
                   idx1]-lat0))-s_y) < ds_y/2.0 ) & ( n
                   .abs((londeg2km*(lons[idx1]-lon0))-
                   s_x) < ds_x/2.0 )

            idxt=idx1[n.where( dist_filter &
                               ( n.abs( (heights[idx1]-
                                  hg0) - s_z ) < (ds_z
                                  /2.0) ) &
                               ( idx1 != k ) &
                               ( n.abs( t[idx1] - mt0 -
                                   tau ) < dtau/2.0 ) )
                                  [0]]
            for l in idxt:
                if "%d-%d"%(k,l) not in pair_dict:
                   hor_dist = n.sqrt( (latdeg2km*(lats
                       [l]-lat0))**2.0+(londeg2km*(lons
                       [l]-lon0))**2.0)
                   # filter our measurements that are
                       too close
                   if hor_dist > min_ds_h and n.abs(t[
                       l]-t[k])> min_dt:
                       pair_dict["%d-%d"%(k,l)]=True
                       pair_dict["%d-%d"%(l,k)]=True
                       pairs.append((k,l))
                       tods.append(t[k])
                       taus.append(t[l]-t[k])
                       s_zs.append(heights[l]-heights[
                           k])
                       s_xs.append(londeg2km*(lons[l]-
                           lon0))
                       s_ys.append(latdeg2km*(lats[l]-
                           lat0))
                       s_hs.append(n.sqrt( (latdeg2km
                           *(lats[l]-lat0))**2.0+(
                           londeg2km*(lons[l]-lon0))
                           **2.0))

   # histogram and interpolate the number of
       measurements as a function of day
```

```
tods=n.array(tods)
# 30 minute bins, 0..24 hours utc histogram
thist,tbins=n.histogram(n.mod(tods/3600.0,24),bins
    =48)

if plot_thist:
     plt.plot(tbins[0:len(thist)],thist)
     plt.show()
# make sure that weight is at least 1.0 (one
    measurement per hour)
thist[thist < 1.0]=1.0
tbins2=0.5*(tbins[0:(len(tbins)-1)]+tbins[1:(len(
    tbins))])
# start with 0 hours
tbins2[0]=0.0
# end with 24 hours
tbins2[len(tbins2)-1]=24.0
countf=si.interp1d(tbins2,thist)

n_meas=len(pairs)

A=n.zeros([n_meas,6])
Ao=n.zeros([n_meas,6])

m=n.zeros(n_meas)
mo=n.zeros(n_meas)

print("n_meas %d"%(n_meas))

ws=[]

for pi in range(n_meas):
     k=pairs[pi][0]
     l=pairs[pi][1]

     w=1.0/countf(n.mod( (t[k]-t0)/3600,24.0 ))
     ws.append(w)
     A[pi,0]=w*braggs[k,0]*braggs[l,0] # ku1*ku2
     Ao[pi,0]=braggs[k,0]*braggs[l,0] # ku1*ku2

     A[pi,1]=w*braggs[k,1]*braggs[l,1] # kv1*kv2
     Ao[pi,1]=braggs[k,1]*braggs[l,1] # kv1*kv2
```

```
        A[pi,2]=w*braggs[k,2]*braggs[l,2]  # kw1*kw2
        Ao[pi,2]=braggs[k,2]*braggs[l,2]  # kw1*kw2

        A[pi,3]=w*(braggs[k,0]*braggs[l,1]+braggs[k,1]*
            braggs[l,0])  # ku1*kv2 + kv1*ku2
        Ao[pi,3]=braggs[k,0]*braggs[l,1]+braggs[k,1]*
            braggs[l,0]  # ku1*kv2 + kv1*ku2

        A[pi,4]=w*(braggs[k,0]*braggs[l,2]+braggs[k,2]*
            braggs[l,0])  # ku1*kw2 + kw1*ku2
        Ao[pi,4]=braggs[k,0]*braggs[l,2]+braggs[k,2]*
            braggs[l,0]  # ku1*kw2 + kw1*ku2

        A[pi,5]=w*(braggs[k,1]*braggs[l,2]+braggs[k,2]*
            braggs[l,1])  # kv1*kw2 + kw1*kv2
        Ao[pi,5]=braggs[k,1]*braggs[l,2]+braggs[k,2]*
            braggs[l,1]  # kv1*kw2 + kw1*kv2

        m[pi]=w*((2*n.pi)**2.0)*dops[k]*dops[l]
        mo[pi]=((2*n.pi)**2.0)*dops[k]*dops[l]
    try:
        ws=n.array(w)
        xhat=n.linalg.lstsq(A,m)[0]

        # inverse scale weights
        resid=(mo-n.dot(Ao,xhat))

        mean_err=n.median(resid)
        resid_std=n.median(n.abs(resid-mean_err))

        if debug_plot:
            plt.plot(resid,".")
            plt.axhline(resid_std*5.0,color="red")
            plt.axhline(-resid_std*5.0,color="red")
            plt.show()

        # remove extreme outliers
        good_idx=n.where(n.abs(resid_std-mean_err) <
            5.0*resid_std)[0]

        An=A[good_idx,:]
        mn=m[good_idx]
        xhat=n.linalg.lstsq(A,m)[0]

        stdev=n.sqrt(n.mean(n.abs(resid)**2.0))
        # assuming all measurements are independent
        sigma=n.sqrt(n.diag(n.linalg.inv(n.dot(n.
           transpose(Ao),Ao))))*stdev
        acf[:]=xhat
        err[:]=sigma

    except:
        traceback.print_exc(file=sys.stdout)
        acf[:]=n.nan
        err[:]=n.nan

    if False:
        plt.hist(taus)
        plt.show()
        plt.hist(s_xs)
        plt.show()
        plt.hist(s_ys)
        plt.show()
        plt.hist(s_zs)
        plt.show()
    return(acf, err, n.mean(taus), n.mean(s_xs), n.mean
       (s_ys), n.mean(s_zs), n.mean(s_hs))


def hor_acfs(meas,h0=90,dh=2,tau=0.0,s_h=n.arange
   (0,400.0,25.0),
             ds_h=25.0, ds_z=1.0, dtau=900, title="
                hor_acf"):
    '''
     Horizontal distance spatial correlation function
    '''
    n_lags=len(s_h)
    acfs=n.zeros([n_lags,6])
    errs=n.zeros([n_lags,6])

    names=["$G_{uu}$","$G_{vv}$","$G_{ww}$",
           "$G_{uv}$","$G_{uw}$","$G_{vw}$"]

    shs=[]
    for li in range(n_lags):
        acf,err,tau,sx,sy,sz,sh= cfi(meas, h0=h0, dh=dh
            , s_z=0.0, s_h=s_h[li], ds_h=ds_h, ds_z=ds_z
            , tau=tau,dtau=dtau,
                                       horizontal_dist=
                                          True,min_dt
                                          =10.0,min_ds_h
                                          =10.0)
        shs.append(sh)
        print("s_h␣%1.2f"%(sh))
        print(acf)
        acfs[li,:]=acf
        errs[li,:]=err
    shs=n.array(shs)


    ho=h5py.File(title,"w")
    ho["h0"]=h0
    ho["dtau"]=dtau
    ho["ds_h"]=ds_h
    ho["acfs"]=acfs
    ho["errs"]=errs
    ho["shs"]=shs
    ho["sho"]=s_h
    ho.close()



    return(h0,dtau,ds_h,acfs,errs,shs,s_h,names)

def plot_hor_acfs(shs,
                  names,
                  acfs,
                  ds_z,
                  dtau,
                  ds_h,
                  err_vars,
                  colors,
                  zlag,
                  n_avg):
    plt.subplot(121)
    for i in range(6):
        #plt.plot(shs,acfs[:,i],label=names[i])
        plt.errorbar(shs,acfs[:,i],yerr=n.sqrt(err_vars
                 [:,i]/n_avg),color=colors[i],label=names[i])
    plt.legend()
    plt.xlabel("Horizontal lag (km)")
    plt.ylabel("Correlation (m$^2$/s$^2$)")
    plt.title("Horizontal ACF\n$\Delta s_z=%1.1f$ km, $
        \Delta \\tau = %1.1f$ s $\Delta s_h=%1.1f$ km"%(
        ds_z,dtau,ds_h))

    plt.subplot(122)
    # estimate structure function
    # tbd, estimate zero lag with exp function
    sfu=2.0*zlag*acfs[0,0]-2.0*acfs[:,0]
    sfv=2.0*zlag*acfs[0,1]-2.0*acfs[:,1]
    # don't show zero-lag. it doesn't make sense.
    plt.loglog(shs,sfu,"o-",label="$S'_{uu}$")
    plt.loglog(shs,sfv,"o-",label="$S'_{vv}$")
    a=sfu[2]/shs[2]**(2.0/3.0)
#    shs[0]=0.0
    plt.loglog(shs,a*shs**(2.0/3.0),label="$s^{2/3}$")
    plt.legend()
    plt.xlabel("Horizontal lag (km)")
    plt.ylabel("Structure function (m$^2$/s$^2$)")
    plt.title("Horizontal structure function")
    plt.tight_layout()
    plt.show()

def ver_acfs(meas,
             h0=90,
             dh=1,
             tau=0.0,
             s_z=n.arange(-10,10.0,1.0),
             s_h=0.0,
             ds_h=50.0,
             ds_z=1.0,
             dtau=900,
             plot_acfs=False
             ):
    '''
     Vertical lag spatial correlation function
    '''

    n_lags=len(s_z)
    acfs=n.zeros([n_lags,6])
    errs=n.zeros([n_lags,6])

    names=["$G_{uu}$","$G_{vv}$","$G_{ww}$",
           "$G_{uv}$","$G_{uw}$","$G_{vw}$"]

    szs=[]
    for li in range(n_lags):
        print("s_z␣%1.1f"%(s_z[li]))
        acf,err,tau,sx,sy,sz,sh= cfi(meas, h0=h0, dh=dh
            , s_z=s_z[li], s_h=0.0, ds_h=ds_h, ds_z=ds_z
            , tau=tau, dtau=dtau, horizontal_dist=True)
        szs.append(sz)
        print("s_z␣%1.2f"%(sz))
        print(acf)
        acfs[li,:]=acf
        errs[li,:]=err
    szs=n.array(szs)

    if plot_acfs:
        plot_ver_acf(szs,acfs,names,ds_z,dtau)
    return(szs,acfs,errs,names,ds_z,dtau)



def plot_ver_acf(szs,acfs,names,ds_z,dtau,err_vars,
    colors,n_avg):

    plt.figure(figsize=(8*1.5,6*1.5))
    plt.subplot(121)
    for i in range(6):
        plt.plot(szs,acfs[:,i],label=names[i])
        plt.errorbar(szs,acfs[:,i],yerr=n.sqrt(err_vars
            [:,i]/n_avg),color=colors[i])
    plt.legend()
    plt.xlabel("Vertical␣lag␣(km)")
    plt.ylabel("Correlation␣(m$^2$/s$^2$)")
    plt.title("Vertical␣ACF␣$\Delta␣s_z=%1.1f$,␣$\Delta
      ␣\\tau␣=␣%1.1f$␣s"%(ds_z,dtau))

    plt.subplot(122)
    # estimate structure function
    # tbd, estimate zero lag with exp function
    sfu=2.0*1.01*acfs[0,0]-2.0*acfs[:,0]
    sfv=2.0*1.01*acfs[0,1]-2.0*acfs[:,1]
    # don't show zero-lag. it doesn't make sense.
    plt.loglog(szs,sfu,"o-",label="$S'_{uu}$")
    plt.loglog(szs,sfv,"o-",label="$S'_{vv}$")
    a=sfu[2]/szs[2]**(2.0/3.0)
#     shs[0]=0.0
    plt.loglog(szs,a*szs**(2.0/3.0),label="$s^{2/3}$")
    plt.legend()
    plt.xlabel("Vertical␣lag␣(km)")
    plt.ylabel("Structure␣function␣(m$^2$/s$^2$)")
    plt.title("Vertical␣structure␣function")
    plt.tight_layout()
    plt.show()



def temporal_acfs(meas,
                 h0=91,        # height
                 dh=1,         # width of height range
                 tau=n.arange(96)*900.0,
                 dtau=300.0, # temporal lag resolution
                 ds_h=25.0,  # horizontal lag
                     resolution
                 ds_z=1.0,
                 title='title'):  # vertical lag
                     resolution
    '''
     Temporal lag spatial correlation function
    '''
    n_lags=len(tau)
    acfs=n.zeros([n_lags,6])
    errs=n.zeros([n_lags,6])

    names=["$G_{uu}$","$G_{vv}$","$G_{ww}$",
           "$G_{uv}$","$G_{uw}$","$G_{vw}$"]

    for li in range(n_lags):

        print("tau␣%1.1f"%(tau[li]))
        acf,err,mtau,msx,msy,msz,msh = cfi(meas,
                                           h0=h0,
                                           dh=dh,
                                           s_z=0.0,
                                           s_h=0.0,
                                           ds_h=ds_h,
                                           ds_z=ds_z,
                                           tau=tau[li],
                                           dtau=dtau,
                                           min_dt=
                                               min_dt,
                                           horizontal_dist
                                               =True)

        print(acf)
        acfs[li,:]=acf
        errs[li,:]=err
    return(acfs,errs,tau,dtau,ds_h,names)

def plot_temporal_acfs(acfs,errs,tau,names,ds_h,dtau,
    title):
    for i in range(6):
        plt.plot(tau,acfs[:,i],label=names[i])

    plt.legend()
    plt.xlabel("Temporal␣lag␣(s)")
    plt.ylabel("Correlation␣(m$^2$/s$^2$)")
    plt.title("%s"%(title))
    plt.savefig("C:/Users/OleK/Master_thesis/figs/fig_%
        s.png"%(title))
    plt.show()

    ho=h5py.File("%s_tacf_dtau_%1.0f_tau_%1.0f_ds_h_
        %1.2f.h5"%(title, dtau,n.max(tau),ds_h),"w")
    ho["tau"]=tau
    ho["acf"]=acfs
    ho["dtau"]=dtau
    ho["ds_h"]=ds_h
    ho.close()

    return(tau,acfs)
```

```
def example1():
    # estimate a temporally high pass filtered
        horizontal acf
    meas=get_meas(mean_rem=True, plot_dops=False,
        mean_wind_file="res/mean_wind_4h.h5")
    hor_acfs(meas,h0=92.0,dh=5,ds_z=1.0,ds_h=25.0,s_h=n
        .arange(0,400.0,25.0),dtau=600.0)

def example2():
    # estimate a high pass filtered vertical acf
    meas=get_meas(mean_rem=True,plot_dops=False,
        mean_wind_file="res/mean_wind_1h.h5")
    ver_acfs(meas,h0=89.0,dh=4.0,s_z=n.arange
        (0.0,10.0,1.0),dtau=300.0, tau=0.0, s_h=0.0,
        ds_h=100.0)

def example3():
    # estimate a temporal autocorrelation function
    meas=get_meas(mean_rem=False,plot_dops=False,
        mean_wind_file="res/mean_wind_4h.h5")

    dtau=900.0
    h_max=72.0
    n_t=int(h_max*3600.0/dtau)

    temporal_acfs(meas,h0=91.0,dh=4,ds_z=1.0,ds_h=50.0,
        dtau=dtau,tau=n.arange(float(n_t))*dtau)

def example4():
    # estimate a temporal autocorrelation function for
        high pass filtered measurements
    # at most 12 hours lag
    meas=get_meas(mean_rem=True,plot_dops=False,
        mean_wind_file="res/mean_wind_4h.h5")

    dtau=600.0
    h_max=12.0
    n_t=int(h_max*3600.0/dtau)
    temporal_acfs(meas,h0=93.5,dh=5,ds_z=1.0,ds_h=25.0,
        dtau=dtau,tau=n.arange(float(n_t))*dtau)

def example5():
    # full horizontal correlation function
    # 25 km resolution, 25 km lag spacing
    # 900 second lag resolution (+/- 450 seconds)
    meas=get_meas(mean_rem=False, plot_dops=False,
        mean_wind_file="res/mean_wind_4h.h5")
    hor_acfs(meas,h0=92.0,dh=5,ds_z=1.0,ds_h=25.0,s_h=n
        .arange(0,400.0,25.0),dtau=900.0)



def mean_wind_cf(heights=n.arange(80,111),
                 hour_of_day=n.arange(48)*0.5,
                 dtau=1800.0,    # half hour time
                     resolution
                 ds_h=300.0):    # horizontal lag
                     resolution
    # mean wind correlation functions for times of day
        at 30 minute time resolution
    # and 1 km height resolution
    #
    # read measurements. don't remove mean wind, as we
        are interested in the full correlation function
    meas=get_meas(mean_rem=False, plot_dops=False,
        mean_wind_file="res/mean_wind_4h.h5")

    n_heights=len(heights)
    n_hods=len(hour_of_day)

    # this is where we store the correlation functions
    C=n.zeros([n_hods,n_heights,6])

    for hi,h0 in enumerate(heights):
        for ti,t0 in enumerate(hour_of_day):
            print("doing␣height␣%1.2f␣hour␣of␣day␣%1.2f
                "%(h0,t0))
            acf,err,tau,sx,sy,sz,sh= cfi(meas,
                                         h0=h0, dh=2.0,
                                              # only
                                           use
```

```
                                                                    measurements
                                                                     at height
                                                                    h0 +/- dh
                                                                    /2.0
                                                                hour_of_day=t0
                                                                    , # only
                                                                    use
                                                                    measurements
                                                                     where utc
                                                                    hour of day
                                                                     is t0 +/-
                                                                    dhour_of_day
                                                                    /2
                                                                dhour_of_day
                                                                    =1.0,
                                                                s_z=0.0, ds_z
                                                                    =2.0,
                                                                s_h=0.0, ds_h=
                                                                    ds_h,
                                                                tau=0.0 ,dtau
                                                                    =1800.0,
                                                                horizontal_dist
                                                                    =True)
                print(acf)
                C[ti,hi,:]=acf
                ho=h5py.File("mean_cf.h5","w")
                ho["C"]=C
                ho["heights"]=heights
                ho["hour_of_day"]=hour_of_day
                ho.close()

        ho=h5py.File("mean_cf.h5","w")
        ho["C"]=C
        ho["heights"]=heights
        ho["hour_of_day"]=hour_of_day
        ho.close()
        return(C)


def meas_groundpoint():
        #task2, plot groundpoints of al metor data
            measurments.
        meas=get_meas(mean_rem=False, plot_dops=False,
            mean_wind_file="res/mean_wind_4h.h5")
        lats=meas["lats"]    #~45-61 lats
        lons=meas["lons"]    #~2-24 lons
        img = plt.imread("minlonslats_maxlonlats.png")
        fig, ax = plt.subplots()
        plt.xticks(n.arange(2, 33, step=2))
        plt.yticks(n.arange(44, 64, step=2))
        ax.imshow(img, extent=[1.5, 33, 44, 62])
        ax.plot(lons, lats, 'ro', markersize=0.5, alpha
            =0.1)
        plt.title('Ground␣point␣of␣meteor␣measurements')
        plt.xlabel('longitude')
        plt.ylabel('latitude')
        plt.show()

def meas_time_of_day(hour_of_day=n.arange(48)*0.5,
                     dtau=1800.0):

        meas=get_meas(mean_rem=False, plot_dops=False,
            mean_wind_file="res/mean_wind_4h.h5")
        time=meas["t"]


        n_times=int((n.max(time)-n.min(time))/(dtau))
        t0=n.min(time)
        #y-axis=number of meas during the 30 min interval
        #x-axis=30 min time bins

        hod=n.mod(time/3600.0,24)
        # histogram and interpolate the number of
            measurements as a function of day
        hods=n.array(hod)
        # 30 minute bins, 0..24 hours utc histogram
        thist, tbins=n.histogram(n.mod(hods/3600.0,24),bins
            =48)
        plt.bar(hour_of_day, thist)
        plt.title('Time␣of␣day␣when␣meteor␣measurements␣
            occur')
        plt.xlabel('hour␣of␣day')
        plt.ylabel('number␣of␣measurments')
        plt.xticks(n.arange(0,25, step=2))
        plt.show()
```

```
def mean_wind_cf_plot():

    ho=h5py.File("mean_cf.h5", "r")
    hour_of_day=n.copy(ho["hour_of_day"].value)
    heights=n.copy(ho["heights"].value)
    C=n.copy(ho["C"].value)

    a=n.genfromtxt('msis.txt')
    dens=si.interp1d(a[:,0],a[:,1])

    Guu_vv=(C[:,:,0].T+C[:,:,1].T)
    E=n.zeros(Guu_vv.shape)
    for i in range(len(heights)):
        E[i,:]=0.5*dens(heights[i])*Guu_vv[i,:]*1e3

    plt.pcolormesh(hour_of_day, heights, n.log10(E))
          #C[:,:,0]=Guu(0,0)

    plt.title('Kinetic␣energy[$log_{10}(J/m^3)$]')
    plt.xlabel('time␣of␣day␣[hours]')
    plt.ylabel('altitude[km]')

    plt.colorbar()
    plt.clim(-4,-1)

    plt.show()


def km500_horizontal_acf():
    # estimate a temporally high pass filtered
        horizontal acf
    meas=get_meas(mean_rem=True, plot_dops=False,
        mean_wind_file="res/mean_wind_4h.h5")
    hor_acfs(meas,h0=90.0,dh=5,ds_z=1.0,ds_h=25.0,s_h=n
        .arange(0,500.0,50.0),dtau=600.0)


def temporal_acf():
    # estimate a temporal autocorrelation function for
        high pass filtered measurements
    # at most 7 days lag

    meas=get_meas(mean_rem=False,plot_dops=False,
            mean_wind_file="res/mean_wind_4h.h5")

    dtau=1800.0
    h_max=7*24.0
    n_t=int(h_max*3600.0/dtau)

    temporal_acfs(meas,h0=93.5,dh=2,ds_z=1.0,ds_h=25.0,
        dtau=dtau, tau=n.arange(float(n_t))*dtau)


def vertical_acf():
    # estimate a high pass filtered vertical acf
    meas=get_meas(mean_rem=True, plot_dops=False,
        mean_wind_file="res/mean_wind_1h.h5")
    ver_acfs(meas,h0=80.0,dh=4.0,s_z=n.arange
        (0.0,20.0,1.0),dtau=300.0, tau=0.0, s_h=0.0,
        ds_h=100.0)




#mean_wind_cf()
#mean_wind_cf_plot()

#plot_tacf()

#example1()
#example2()
#example3()
#example4()
#example5()

#meas_groundpoint()
#meas_time_of_day(hour_of_day=n.arange(48)*0.5, dtau
    =1800.0)

#temporal_acf()
#if __name__ == "__main__":
#     vertical_acf()
#km500_horizontal_acf()
```

Listing A.2: *sfi2d_hor.py*; Script to calculate the mean wind.

```
#!/usr/bin/env python
#
```

```
# simple mean horizontal wind
#
import h5py
import matplotlib.pyplot as plt
import glob
from mpl_toolkits.mplot3d import Axes3D
import numpy as n

import mmaria_read as mr
import cfi_dac as cfi
import cfi_config as c
import time
import datetime

import geoid_const as gc

latdeg2km=gc.latdeg2km #111.321
londeg2km=gc.londeg2km# n.pi*6371.0*n.cos(n.pi
    *69.0/180.0)/180.0#65.122785

# high time resolution mean wind
#t_avg=1800
# low time resolution mean wind (used for high pass
    filtering)
#lf_t_avg=4*3600.0
#dcos_thresh=0.8
#ofname="res/mean_wind_4h.h5"

def mean_wind(meas="res/
    simone_nov2018_multilink_juha_30min_1000m.h5",
              dt=60*60,t_step=900,dh=1.0,max_alt=105,
                  min_alt=80,dcos_thresh=0.8,
              ofname="res/mean_wind.h5",
              data='h5file'):   #data is the type of
                  input values.

    if data=='h5file':
        h=h5py.File(meas,"r")
        print(h.keys())
        heights=n.copy(h["heights"].value)
        ts=n.copy(h["t"].value)
        dops=n.copy(h["dops"].value)
        braggs=n.copy(h["braggs"].value)
        lats=n.copy(h["lats"].value)
        if "dcos" in h.keys():
            dcoss=n.copy(h["dcos"].value)
        else:
            dcoss=n.zeros([len(heights),2])

        lons=n.copy(h["lons"].value)

    else:   #for now only for mmaria_read.py
        h=meas
        heights=n.copy(h["heights"])
        heights=heights/1000
        ts=n.copy(h["t"])
        dops=n.copy(h["dops"])
        braggs=n.copy(h["braggs"])
        lats=n.copy(h["lats"])
        if "dcos" in h.keys():
            dcoss=n.copy(h["dcos"])
        else:
            dcoss=n.zeros([len(heights),2])

        lons=n.copy(h["lons"])



    dcos2=n.sqrt(dcoss[:,0]**2.0+dcoss[:,1]**2.0)

    ok_idx=n.where(dcos2 < dcos_thresh)[0]
    ts=ts[ok_idx]
    lats=lats[ok_idx]
    lons=lons[ok_idx]
    heights=heights[ok_idx]
    braggs=braggs[ok_idx,:]
    dops=dops[ok_idx]
    dcoss=dcoss[ok_idx,:]

    lat0=n.median(lats)
    lon0=n.median(lons)

    rgs=n.arange(min_alt,max_alt,int(dh))
    times=n.arange(int(n.min(ts)),int(n.max(ts)),t_step
        )
    n_rgs=len(rgs)
```

```
np=2

v=n.zeros([np,len(times),len(rgs)])
ve=n.zeros([np,len(times),len(rgs)])
ve[:,:,:]=n.nan
v[:,:,:]=n.nan

for ti,t in enumerate(times):
    print("%d/%d"%(ti,len(times)))
    ridxs=[]
    n_r=[]
    for r in rgs:
        hidx=n.where((heights > r)&(heights < (r+dh
            ))&(ts>t-dt/2)&(ts<(t+dt/2)))[0]
        ridxs.append(hidx)
        n_r.append(len(hidx))

    n_r=n.array(n_r)
    n_meas=n.sum(n_r)
    # v_x = v_x
    # v_y = v_y
    A=n.zeros([n_meas,np*n_rgs])
    m=n.zeros(n_meas)
    n_m = 0
    g_ridx2=[]
    g_ridx=[]

    for ri in range(len(rgs)):
        m_idx=n.arange(n_m,n_m+len(ridxs[ri]))
        # meas
        m[m_idx]=-2.0*n.pi*dops[ridxs[ri]]
        # theory
        # v_u
        A[m_idx,ri*np+0]=braggs[ridxs[ri],0]
        # v_v
        A[m_idx,ri*np+1]=braggs[ridxs[ri],1]

        n_m+=len(ridxs[ri])
        if len(ridxs[ri]) > 10:
            for pi in range(np):
                g_ridx2.append(ri*np+pi)
            g_ridx.append(ri)
            else:
                pass
#                print("bad range %d"%(ri))
        g_ridx2=n.array(g_ridx2)
        g_ridx=n.array(g_ridx)

        if len(g_ridx2) > 0:
            xhat=n.linalg.lstsq(A[:,g_ridx2],m)[0]
            resid=m-n.dot(A[:,g_ridx2],xhat)
            gidx=n.where(n.abs(resid) < 100.0)[0]
            #         plt.plot(resid)
            #         plt.show()

            g_ridx_f=[]
            g_ridx2_f=[]
            for ri in g_ridx:
                n_meas_per_rg=len(n.where(n.abs(A[:,np*
                    ri])>0)[0])
                if n_meas_per_rg > np:
                    g_ridx_f.append(ri)
                for pi in range(np):
                    g_ridx2_f.append(np*ri+pi)

            g_ridx_f=n.array(g_ridx_f,dtype=n.int)
            g_ridx2_f=n.array(g_ridx2_f,dtype=n.int)

            # estimate stdev
            stdev=n.sqrt(n.diag(n.linalg.inv(n.dot(n.
                transpose(A[:,g_ridx2_f]),A[:,g_ridx2_f
                ]))))*n.std(resid)

            n_gridx=len(g_ridx_f)
            for pi in range(np):
                v[pi,ti,g_ridx_f]=xhat[np*n.arange(
                    n_gridx)+pi]
                ve[pi,ti,g_ridx_f]=stdev[np*n.arange(
                    n_gridx)+pi]

    times_h=(times-times[0])/3600.0
    dt2=times[1]-times[0]
    dh2=rgs[1]-rgs[0]

    ho=h5py.File(ofname,"w")
    ho["times"]=times
    ho["rgs"]=rgs
    ho["v"]=v
    #ho["v_fluct"]=v2-v
    ho["ve"]=ve
    ho["dt"]=dt2
    ho["dh"]=dh2
    ho["lat0"]=lat0
    ho["lon0"]=lon0
    ho.close()


    return(times,times_h,v,ve,rgs,lat0,lon0,dt2,dh2)


if __name__ == "__main__":

    md=mr.mmaria_data(c.data_directory)#for many files
        in a directory
    b=md.get_bounds()

    d=md.read_data_date(d0=datetime.date(2019,1,1),d1=
        datetime.date(2019,1,3))

    lf_t_avg=4*3600
    t_avg=1800.0
    times,times_h,v,ve,rgs,lat0,lon0,dt,dh=mean_wind(
        meas=d, dt=lf_t_avg,dh=1.0,max_alt=105,min_alt
        =78,dcos_thresh=0.8,data='dict')
    #times,times_h,v,ve,rgs,lat0,lon0,dt,dh=mean_wind(
        dt=1800,dh=1.0,max_alt=105,min_alt=78,
        dcos_thresh=0.8)
    times2,times_h2,v2,ve2,rgs2,lat02,lon02,dt2,dh2=
        mean_wind(meas=d, dt=t_avg,dh=1.0,max_alt=105,
        min_alt=78,dcos_thresh=0.8,data='dict')

    print(dh)


    plt.figure(figsize=(8,8))

    plt.subplot(322)
    plt.pcolormesh(times_h,rgs,n.transpose(v2[0,:,:]),
        vmin=-70,vmax=70,cmap="jet")
    plt.title("Zonal mean wind (m/s)")
    #plt.xlabel("Time (h)")
    plt.ylabel("%d minute average"%(t_avg/60.0))
    plt.colorbar()

    plt.subplot(321)
    plt.pcolormesh(times_h,rgs,n.transpose(v2[1,:,:]),
        vmin=-70,vmax=70,cmap="jet")
    plt.title("Meridional mean wind (m/s)")
    #plt.xlabel("Time (h)")
    plt.ylabel("Altitude (km)")
    plt.colorbar()

    plt.subplot(324)
    plt.pcolormesh(times_h,rgs,n.transpose(v[0,:,:]),
        vmin=-70,vmax=70,cmap="jet")
    #plt.title("Zonal mean wind (m/s)")
    #plt.xlabel("Time (h)")
    plt.ylabel("4 hour average")
    plt.colorbar()

    plt.subplot(323)
    plt.pcolormesh(times_h,rgs,n.transpose(v[1,:,:]),
        vmin=-70,vmax=70,cmap="jet")
    #plt.title("Meridional mean wind (m/s)")
    #plt.xlabel("Time (h)")
    plt.ylabel("Altitude (km)")
    plt.colorbar()

    plt.subplot(326)
    plt.pcolormesh(times_h,rgs,n.transpose(v2[0,:,:]-v
        [0,:,:]),vmin=-25,vmax=25,cmap="jet")
    #plt.title("Zonal mean wind (m/s)")
    plt.xlabel("Time (h)")
    plt.ylabel("Residual")
    plt.colorbar()

    plt.subplot(325)
    plt.pcolormesh(times_h,rgs,n.transpose(v2[1,:,:]-v
        [1,:,:]),vmin=-25,vmax=25,cmap="jet")
    #plt.title("Meridional mean wind (m/s)")
    plt.xlabel("Time (h)")
```

```
    #plt.ylabel("Residual")
    plt.ylabel("Altitude␣(km)")
    plt.colorbar()

    plt.tight_layout()
    plt.savefig("mean_wind.png")
    plt.show()
```

Listing A.3: *mmaria_read.py*; Script to read the MMARIA dataset in an efficient and easy way.

```
#!/usr/bin/env python

import numpy as n
import glob
import h5py
import matplotlib.pyplot as plt
import cfi_config as c
import time
import datetime


#alpha_norm                Dataset {56452}
#braggs                    Dataset {56452, 3}
#dcos                      Dataset {56452, 2}
#dh                        Dataset {SCALAR}
#dop_errs                  Dataset {56452}
#dops                      Dataset {56452}
#dt                        Dataset {SCALAR}
#heights                   Dataset {56452}
#lats                      Dataset {56452}
#link                      Dataset {56452}
#lons                      Dataset {56452}
#rgs                       Dataset {30}
#t                         Dataset {56452}
#times                     Dataset {48}
#v                         Dataset {2, 48, 30}
#v_resid                   Dataset {56452}
#ve                        Dataset {2, 48, 30}

keys=["alpha_norm","braggs","dcos","dh","dop_errs","
    dops",
      "dt","heights","lats","link","lons","rgs","t","
         rgs",
        "t","times","v","v_resid","ve"]


class mmaria_data:
    def __init__(self,dname,debug=False):
        self.fl = glob.glob("%s/*.h5"%(dname))
        self.fl.sort()
        self.mint=[]
        self.maxt=[]
        self.debug=debug

        for f in self.fl:
            if self.debug:
                print("reading␣%s"%(f))
            h=h5py.File(f,"r")
            t=h["t"].value
            self.mint.append(n.min(t))
            self.maxt.append(n.max(t))
            h.close()
        self.mint=n.array(self.mint)
        self.maxt=n.array(self.maxt)

    def get_bounds(self):
        return([n.min(self.mint),n.max(self.maxt)])

    def read_data_date(self,d0,d1,read_all_detections=
       True):#is datetime.date(d0) valid to run a
       function, so can just put in 2019,1,1
        #d0,d1 is a date like d0=datetime.date
           (2019,1,5)

        t0=time.mktime(d0.timetuple())
        t1=time.mktime(d1.timetuple())
        return(self.read_data(t0,t1,read_all_detections
           ))

    def read_data(self,t0,t1,read_all_detections=True):
        """
        Read all meteor radar network data between
           these times (unix)
        """
        file_idx=[]
        for fi in range(len(self.fl)):
            if self.maxt[fi] > t0 and self.mint[fi] <
                t1:
                 file_idx.append(fi)

        alpha_norm=n.zeros([0],dtype=n.float32)
        braggs=n.zeros([0,3],dtype=n.float32)
        dcos=n.zeros([0,2],dtype=n.float32)
        dh=1.5
        dop_errs=n.zeros([0],dtype=n.float32)
        dops=n.zeros([0],dtype=n.float32)
        dt=0
        heights=n.zeros([0],dtype=n.float32)
        lats=n.zeros([0],dtype=n.float32)
        lons=n.zeros([0],dtype=n.float32)
        link=n.zeros([0],dtype="<U60")
        rgs=n.zeros([30],dtype=n.float32)
        t=n.zeros([0],dtype=n.float32)
        times=n.zeros([0],dtype=n.float32)
        v=n.zeros([2,0,30],dtype=n.float32)
        v_resid=n.zeros([0],dtype=n.float32)
        ve=n.zeros([2,0,30],dtype=n.float32)
        print("reading")
        h0=75
        h1=105

        for i in file_idx:#range(first_idx, last_idx+1):
#               print(i)
            h=h5py.File(self.fl[i],"r")
            if read_all_detections:
                didx=n.where( ((h["t"].value) > t0) &
                              ((h["t"].value) < t1) &
                              (h["heights"].value/1e3 >
                                  h0) &
                              (h["heights"].value/1e3 <
                                  h1) )[0]

                t = n.concatenate((t,h["t"].value[didx
                    ]))
                alpha_norm = n.concatenate((alpha_norm,
                    h["alpha_norm"].value[didx]))
                braggs = n.concatenate((braggs,h["
                    braggs"].value[didx,:]))
            dcos = n.concatenate((dcos,h["dcos"].
               value[didx,:]))
            dh=h["dh"].value
            dop_errs = n.concatenate((dop_errs,h["
               dop_errs"].value[didx]))
            dops = n.concatenate((dops,h["dops"].
               value[didx]))
            dt=h["dt"].value
            heights=n.concatenate((heights,h["
               heights"].value[didx]))
            lats=n.concatenate((lats,h["lats"].
               value[didx]))
            lons=n.concatenate((lons,h["lons"].
               value[didx]))
            link=n.concatenate((link,h["link"].
               value[didx]))
            v_resid=n.concatenate((v_resid,h["
               v_resid"].value[didx]))

        # mean horizontal wind model
        rgs=h["rgs"].value
        times=n.concatenate((times,h["times"].value
           ))
        v=n.concatenate((v,h["v"].value),axis=1)
        ve=n.concatenate((ve,h["ve"].value),axis=1)

        if self.debug:
            print("file␣idx␣%d"%(i))
    tu,idx=n.unique(t,return_index=True)
    return({"t":t[idx],
            "alpha_norm":alpha_norm[idx],
            "braggs":braggs[idx,:],
            "dcos":dcos[idx,:],
            "dh":dh,
            "dop_errs":dop_errs[idx],
            "dops":dops[idx],
            "dt":dt,
            "heights":heights[idx],
            "lats":lats[idx],
            "lons":lons[idx],
            "link":link[idx],
            "rgs":rgs,
            "v_resid":v_resid,
                      "times":times,
                      "v":v,
                      "ve":ve})


if __name__ == "__main__":
    """
    Example usage
    """
    # directory with all mmaria network data
    md=mmaria_data(c.data_directory)
    # what is the data bounds (first and last time
        stamp)
    print(md.get_bounds())

    # read all meteor radar data between these two
        timestamps
    d=md.read_data(1514774804,1514974804)

    plt.pcolormesh(d["times"],d["rgs"]/1e3,n.transpose(
        d["v"][0,:,:]),vmin=-100,vmax=100)
    plt.xlabel("Time␣(unix)")
    plt.ylabel("Altitude␣(km)")
    plt.colorbar()
    plt.show()
```

```
import time
import datetime
import numpy as n
import h5py
import matplotlib.pyplot as plt

# our internal modules
import mmaria_read as mr
import cfi_dac as cfi
import cfi_config as c
import mean_wind_est as mw

md=mr.mmaria_data(c.data_directory)#for many files in a directory
b=md.get_bounds()

def avg_hor_acfs(dcos_thresh=0.8,
                 mean_wind_time_avg=4*3600.0,
                 h0=90.0,
                 dh=5,
                 ds_h=25.0,
                 dtau=300.0,
                  s_h=n.arange(25.0,500.0,25.0),
                  ds_z=1,
                  years=[2018,2019,2020],
                  months=[5,6,7],
                  name="summer_hacf",
                  remove_mean=False,
                  n_days=31):
    n_lags=len(s_h)
    all_acfs=[]
    all_errs=[]

    n_avg=0.0

    for year in years:
        for month in months:
            d0=datetime.date(year,month,1)
            t0=time.mktime(d0.timetuple())

            for day in range(n_days):
                d=md.read_data(t0=t0+day*24*3600.0,t1=t0+(day+1)*24*3600.0)
                n_meas=len(d["t"])
                print("n_meteors %d"%(n_meas))
                if n_meas > 100:
                    if remove_mean:

                        ↪  times,times_h,v,ve,rgs,lat0,lon0,dt,dh=mw.mean_wind(meas=d,
                                                                                    ↪  dt=mean_wind_time_avg,
                                                                                    ↪  dh=1.0,
                                                                                    ↪  max_alt=105,
                                                                                    ↪  min_alt=78,
                                                                                    ↪  dcos_thresh=dcos_thresh,
                                                                                    ↪  ofname="res/tmp.h5",
                                                                                    ↪  data='dict')

                    meas=cfi.get_meas(meas_file=d,
                                      mean_rem=remove_mean,
                                      plot_dops=False,
                                      dcos_thresh=dcos_thresh,
                                      mean_wind_file="res/tmp.h5",
                                      data='mmaria')

                    #title='Vertical AFC with data from date {}.{}.{} to
                    ↪  {}.{}.{}'.format(y,m0,day,y1,m1,day)
#                        return(h0,dtau,ds_h,acfs,errs,shs,s_h,names)


                    ↪  ih0,idtau,dis_h,acfs,errs,ishs,si_h,names=cfi.hor_acfs(meas,
                                                              h0=h0,
                                                              dh=dh,
                                                              ds_z=ds_z,
                                                              ds_h=ds_h,
                                                              s_h=s_h,
                                                              dtau=dtau,
                                                              title=name)
                    all_acfs.append(acfs)#+=acfs
                    all_errs.append(errs)
                    n_avg+=1.0
    all_acfs=n.array(all_acfs)
    all_errs=n.array(all_errs)
    err_vars=n.zeros([len(s_h),6])
    acfs=n.zeros([len(s_h),6])
    for i in range(len(s_h)):
        for ci in range(6):
            err_vars[i,ci]=n.var(all_acfs[:,i,ci])
            ws=0.0
            for mi in range(int(n_avg)):
                w=1.0/all_errs[mi,i,ci]
                ws+=w
                acfs[i,ci]+=w*all_acfs[mi,i,ci]
            acfs[i,ci]=(1.0/ws)*acfs[i,ci]

    #print(all_acfs.shape)
    colors=["C0","C1","C2","C3","C4","C5"]

    cfi.plot_hor_acfs(shs=s_h,
                  names=names,
                  acfs=acfs,
                  ds_z=ds_z,
                  dtau=dtau,
                  ds_h=ds_h,
                  err_vars=err_vars,
                  colors=colors,
                  n_avg=n_avg)


    plt.savefig("C:/Users/OleK/Master_thesis/figs/fig_%s.png"%(name))
    ho=h5py.File("res/%s.h5"%(name),"w")
    ho["acf"]=acfs
    ho["s_h"]=s_h
    ho["err_var"]=err_vars/n.sqrt(n_avg)
    ho["h0"]=h0
    ho["dtau"]=dtau
    ho["ds_h"]=ds_h
    ho.close()




    # 2018 to 2020
# summer: 5,6,7
# winter: 11,12,1
# fall: 8,9,10
# spring: 2,3,4
#y=2019
#y1=2019
#m0=5
#m1=8
#day=1

avg_hor_acfs(dcos_thresh=0.8,
             mean_wind_time_avg=4*3600.0,
             h0=90.0,
             dh=5,
             ds_h=25.0,
             dtau=300.0,
             s_h=n.arange(25.0,500.0,25.0),
             ds_z=1,
             years=[2018,2019,2020],
             months=[5,6,7],
             name="summer_hacf",
             remove_mean=False,
             n_days=31)

#avg_ver_acfs(dcos_thresh=0.8,
#             mean_wind_time_avg=4*3600.0,
#             h0=80.0,
#             ds_h=100.0,
#             dtau=300.0,
#             s_z=n.arange(0.0,20.0,1.0),
#             years=[2018,2019,2020],
#             months=[11,12,1],
#             name="winter_vacf",
#             n_days=31)
```

```
import time
import datetime
import numpy as n
import h5py
import matplotlib.pyplot as plt
import os

# our internal modules
import mmaria_read as mr
import cfi_dac as cfi
import cfi_config as c
import mean_wind_est as mw

from mpi4py import MPI
comm=MPI.COMM_WORLD
size=comm.Get_size()
rank=comm.Get_rank()

md=mr.mmaria_data(c.data_directory)#for many files in a directory
b=md.get_bounds()
```

```


def avg_ver_acfs(dcos_thresh=0.8,
                 mean_wind_time_avg=4*3600.0,
                 h0=80.0,
                 ds_h=100.0,
                 dtau=300.0,
                 s_z=n.arange(0.0,20.0,1.0),
                 years=[2018,2019,2020],
                 months=[5,6,7],
                 mean_rem=False,
                 name="summer_vacf",
                 n_days=31):

    os.system("mkdir -p mpi/%s"%(name))
    os.system("rm mpi/%s/*.h5"%(name))


    pars=[]
    for year in years:
        #for month in [5,6,7]:
        for month in months:
            pars.append((year,month,1))
    n_pars=len(pars)
    print("n_pars %d"%(n_pars))
    for pi in range(rank,n_pars,size):
        year=pars[pi][0]
        month=pars[pi][1]
        d0=datetime.date(year,month,1)
        t0=time.mktime(d0.timetuple())

        d=md.read_data(t0=t0,t1=t0+n_days*24*3600)
        n_meas=len(d["t"])
        print("n_meteors %d"%(n_meas))
        if n_meas > 100:
            mwname="res/temp-%f.h5"%(time.time())
            if mean_rem:
                times,times_h,v,ve,rgs,lat0,lon0,dt,dh=mw.mean_wind(meas=d,

                                                                       ↪  dt=mean_wind_time_avg,
                                                                       dh=1.0,

                                                                       ↪  max_alt=105,

                                                                       ↪  min_alt=78,

                                                                       ↪  dcos_thresh=dcos_thresh,

                                                                       ↪  ofname=mwname,

                                                                       ↪  data='dict')

            meas=cfi.get_meas(meas_file=d,
                              mean_rem=mean_rem,
                               plot_dops=False,
                               dcos_thresh=dcos_thresh,
                               mean_wind_file=mwname,
                               data='mmaria')
            if mean_rem:
                os.system("rm %s"%(mwname))

            sz,acfs,errs,names,ds_z,tlag=cfi.ver_acfs(meas,
                                                       h0=h0,
                                                       dh=2.0,
                                                       s_z=s_z,
                                                       dtau=dtau,
                                                       tau=0.0,
                                                       s_h=0.0,
                                                       ds_h=ds_h)


            ↪  ho=h5py.File("mpi/%s/ver_res-%d-%f.h5"%(name,month,time.time()),"w")
            ho["acf"]=acfs
            ho["err"]=errs
            ho["s_z"]=s_z
            ho["ds_z"]=ds_z
            ho["ds_h"]=ds_h
            ho["dtau"]=dtau
            ho["h0"]=h0
            ho.close()


    # 2018 to 2020
# summer: 5,6,7
# winter: 11,12,1
# fall: 8,9,10
# spring: 2,3,4
#y=2019
#y1=2019
#m0=5
#m1=8
#day=1

#avg_ver_acfs(dcos_thresh=0.8,
#             mean_wind_time_avg=4*3600.0,
#             h0=80.0,
#             ds_h=100.0,
#             dtau=300.0,
#             s_z=n.arange(0.0,20.0,1.0),
#             years=[2018,2019,2020],
#             months=[5,6,7],
#             name="summer_vacf",
#             n_days=31)

def winter_large_scale():
    avg_ver_acfs(dcos_thresh=0.8,
                 mean_wind_time_avg=4*3600.0,
                 h0=88.0,
                 ds_h=500.0,
                 dtau=3600.0,
                 s_z=n.arange(0.0,20.0,1.0),
                 years=[2018,2019,2020],
                 months=[11,12,1],
                 name="winter_88_vacf_ls",
                 n_days=31)

def winter_small_scale():
    avg_ver_acfs(dcos_thresh=0.8,
                 mean_wind_time_avg=4*3600.0,
                 h0=88.0,
                 ds_h=50.0,
                 dtau=600.0,
                 s_z=n.arange(0.0,20.0,1.0),
                 years=[2018,2019,2020],
                 months=[11,12,1],
                 name="winter_88_vacf",
                 n_days=31)
def summer_large_scale():
    avg_ver_acfs(dcos_thresh=0.8,
                 mean_wind_time_avg=4*3600.0,
                 h0=88.0,
                 ds_h=500.0,
                 dtau=3600.0,
                 s_z=n.arange(0.0,20.0,1.0),
                 years=[2018,2019,2020],
                 months=[11,12,1],
                 name="summer_88_vacf_ls",
                 n_days=31)

def summer_small_scale():
    avg_ver_acfs(dcos_thresh=0.8,
                 mean_wind_time_avg=4*3600.0,
                 h0=88.0,
                 ds_h=50.0,
                 dtau=600.0,
                 s_z=n.arange(0.0,20.0,1.0),
                 years=[2018,2019,2020],
                 months=[11,12,1],
                 name="summer_88_vacf",
                 n_days=31)

#summer_small_scale()
#summer_large_scale()
#winter_small_scale()
winter_large_scale()
```

```
import time
import datetime
import numpy as n
import h5py
import matplotlib.pyplot as plt
import os

# our internal modules
import mmaria_read as mr
import cfi_dac as cfi
import cfi_config as c
import mean_wind_est as mw

from mpi4py import MPI
comm=MPI.COMM_WORLD
size=comm.Get_size()
rank=comm.Get_rank()

md=mr.mmaria_data(c.data_directory)#for many files in a directory
b=md.get_bounds()



def avg_temporal_acfs(dcos_thresh=0.8,
                      mean_wind_time_avg=4*3600.0,
                      h0=90.0,
                      dh=5,
                      ds_h=50.0,
                      dtau=300.0,
                      tau=n.arange(0.0,7*24*3600,300.0),
                      ds_z=1,
                      years=[2018,2019,2020],
                      months=[5,6,7],
                      name="summer_tacf"):
    if rank == 0:
        os.system("rm mpi/%s/tacf_res*.h5"%(name))
        os.system("mkdir -p mpi/%s"%(name))
    comm.Barrier()

    n_lags=len(tau)

    max_lag=n.max(tau)

    pars=[]
    for year in years:
        for month in months:
            pars.append([year,month,1])


    n_pars=len(pars)
    print("n_runs %d"%(n_pars))

    for pi in range(rank,n_pars,size):
        year=pars[pi][0]
        month=pars[pi][1]
        day=pars[pi][2]
        d0=datetime.date(year,month,day)
        t0=time.mktime(d0.timetuple())

        d=md.read_data(t0=t0,t1=t0+max_lag+31*24*3600)
        n_meas=len(d["t"])
        print("n_meteors %d"%(n_meas))
        if n_meas > 100:
            meas=cfi.get_meas(meas_file=d,
                              mean_rem=False,
                              plot_dops=False,
                              dcos_thresh=dcos_thresh,
                              mean_wind_file="res/tmp.h5",
                              data='mmaria')

            acfs,errs,tau,dtau,ds_h,names=cfi.temporal_acfs(meas,
                                                            h0=h0,
                                                            tau=tau,
                                                            dtau=dtau,
                                                            min_dt=10,
                                                            ds_h=ds_h)
            ofname="mpi/%s/tacf_res-%06d.h5"%(name,pi)
            print(ofname)
            ho=h5py.File(ofname,"w")
            ho["acfs"]=acfs
            ho["errs"]=errs
            ho["tau"]=tau
            ho["dtau"]=dtau
            ho["ds_h"]=ds_h
            ho["ds_z"]=ds_z
            ho["h0"]=h0
            ho["dh"]=dh
            ho.close()

def summer_day():
    avg_temporal_acfs(dcos_thresh=0.8,
                      h0=90.0,
                      dh=2,
                      ds_h=50.0,
                      dtau=100.0,
                      tau=n.arange(0.0,24*3600,100.0),
                      ds_z=1,
                      years=[2018,2019],
                      months=[5,6,7],
                      dstep=2,
                      name="summer_tacf_1_120_50km")

def summer_small_scale():
    avg_temporal_acfs(dcos_thresh=0.8,
                      h0=90.0,
                      dh=2,
                      ds_h=100.0,
                      dtau=1800.0,
                      tau=n.arange(0.0,14*24*3600,1800.0),
                      ds_z=1,
                      years=[2018,2019],
                      months=[5,6,7],
                      name="summer_tacf_test")
```

```python

def winter_small_scale():
    avg_temporal_acfs(dcos_thresh=0.8,
                      h0=90.0,
                      dh=2,
                      ds_h=100.0,
                      dtau=1800.0,
                      tau=n.arange(0.0,14*24*3600,1800.0),
                      ds_z=1,
                      years=[2018,2019,2020],
                      months=[11,12,1],
                      name="winter_tacf_koki")


winter_small_scale(name="test")
summer_small_scale(name="test")
```

```python
#!/usr/bin/env python
#
# Estimate the number of horizontal and temporal lags
#
import h5py
import numpy as n
import matplotlib.pyplot as plt
from matplotlib.colors import LogNorm
from mpi4py import MPI

import cfi_config as c
import mmaria_read as mr

comm = MPI.COMM_WORLD
rank = comm.Get_rank()

latdeg2km=gc.latdeg2km
londeg2km=gc.londeg2km

#works for mmaria_data
md=mr.mmaria_data(c.data_directory)#for many files in a directory
b=md.get_bounds()

t0=stuffr.date2unix(2019,6,1,0,0,0)
t1=stuffr.date2unix(2019,6,15,0,0,0)

# about two weeks of data in June 2019
d=md.read_data(t0,t1)

dcos_thresh=0.8
lats=d["lats"]
lons=d["lons"]
heights=d["heights"]            #hights in meters
heights=heights/1000
dcoss=d["dcos"]
t=d["t"]
```

```

dcos2=n.sqrt(dcoss[:,0]**2.0+dcoss[:,1]**2.0)
ok_idx=n.where(dcos2 < dcos_thresh)[0]

lats=lats[ok_idx]
lons=lons[ok_idx]
heights=heights[ok_idx]
t=t[ok_idx]

#h=h5py.File("res/simone_nov2018_multilink_juha_30min_1000m.h5","r")# For a
↪  single file
#lats=h["lats"].value
#lons=h["lons"].value
#heights=h["heights"].value             #heights in km
#t=h["t"].value


#works for mmaria_data
md=mr.mmaria_data(c.data_directory)#for many files in a directory
b=md.get_bounds()

d=md.read_data(1514774804,1515974804)

lats=d["lats"]
lons=d["lons"]
heights=d["heights"]             #hights in meters
heights=heights/1000
t=d["t"]



# In[]
print("total number of measurements %d"%(len(t)))

# define a histogram
n_m=len(lats)
rgs=n.arange(80,105)
dh=1.0
h_bins=100

# base 10 log horizontal distance
mer_dist_bins=n.linspace(-2,3.2,num=h_bins+1)
zon_dist_bins=n.linspace(-2,3.2,num=h_bins+1)
hor_dist_bins=n.linspace(-2,3.2,num=h_bins+1)
ver_dist_bins=n.linspace(-50,50,num=h_bins+1)

# base 10 log temporal distance
tdist_bins=n.linspace(-2,6.2,num=h_bins+1)

# zonal, meridional, horizontal, and vertical distances
HZ=n.zeros([h_bins,h_bins])
HM=n.zeros([h_bins,h_bins])
HH=n.zeros([h_bins,h_bins])
HV=n.zeros([h_bins,h_bins])
```

```

# parallel processing.
# processor number=comm.rank
# number if processes=comm.size
for ri in range(comm.rank,len(rgs),comm.size):
    HZ[:,:]=0.0
    HM[:,:]=0.0
    HH[:,:]=0.0
    HV[:,:]=0.0
    rg=rgs[ri]

    zon_dists=[]
    mer_dists=[]
    hor_dists=[]
    ver_dists=[]
    t_dists=[]

    max_zon_dist=0
    max_mer_dist=0
    max_hor_dist=0

    idx0=n.where( (heights >= rg) & (heights < (rg+dh)) )[0]

    ↪  print("number of measurements between %1.2f and %1.2f km = %d"%(rg,rg+dh,len(idx0)))

    lats0 = n.copy(lats[idx0])
    lons0 = n.copy(lons[idx0])
    t0s   = n.copy(t[idx0])
    h0s   = n.copy(heights[idx0])

    for mi in range(len(lats0)):
        if mi % 100 == 0:
            print("rank %d %d/%d"%(comm.rank,mi,len(idx0)))
        lat0=lats0[mi]
        lon0=lons0[mi]
        t0=t0s[mi]
        h0=h0s[mi]


        # horizontal distances
        hor_dists=n.sqrt((n.abs(lats0[mi:len(lats0)]-lat0)*latdeg2km)**2.0 +
        ↪  (n.abs(lons0[mi:len(lats0)]-lon0)*londeg2km)**2.0 )
        mer_dists=n.abs(lats0[mi:len(lats0)]-lat0)*latdeg2km
        zon_dists=n.abs(lons0[mi:len(lats0)]-lon0)*londeg2km
        ver_dists=h0s[mi:len(h0s)]-h0

        mmd=n.max(mer_dists)
        if mmd > max_mer_dist:
            max_mer_dist=mmd
        mzd=n.max(zon_dists)
        if mzd > max_zon_dist:
            max_zon_dist=mzd
        mhd=n.max(hor_dists)
        if mhd > max_mer_dist:
            max_hor_dist=mhd

        temporal_dists = n.abs(t[mi:len(lats0)]-t0)

        ↪  HZ0,zxe,zye=n.histogram2d(n.log10(zon_dists+0.01),n.log10(temporal_dists+0.01),bins=

        ↪  HM0,mxe,mye=n.histogram2d(n.log10(mer_dists+0.01),n.log10(temporal_dists+0.01),bins=

        ↪  HH0,hxe,hye=n.histogram2d(n.log10(hor_dists+0.01),n.log10(temporal_dists+0.01),bins=
        HZ+=HZ0
        HM+=HM0
        HH+=HH0

    plt.pcolormesh(mxe,mye,n.transpose(n.log10(HH+1)))
    cb=plt.colorbar()
    cb.set_label("$\log_{10}$ (counts)")
    plt.xlabel("Horizontal distance $\log_{10}$ (km)")
    plt.ylabel("Temporal distance $\log_{10}$ (s)")

    ↪  plt.title("Distribution of horizontal and temporal lags, h=%1.0f km\nMax distance=%1.0f
    plt.tight_layout()
    plt.savefig("./figs/fig_hor_%03d.png"%(rg))
    plt.close()
    plt.clf()

    plt.pcolormesh(zxe,zye,n.transpose(n.log10(HZ+1)))
    cb=plt.colorbar()
    cb.set_label("$\log_{10}$ (counts)")
    plt.xlabel("Zonal distance $\log_{10}$ (km)")
    plt.ylabel("Temporal distance $\log_{10}$ (s)")

    ↪  plt.title("Distribution of zonal and temporal lags, h=%1.0f km\nMax distance=%1.0f km"%(
    plt.tight_layout()
    plt.savefig("./figs/fig_zon_%03d.png"%(rg))
    plt.close()
    plt.clf()

    plt.pcolormesh(mxe,mye,n.transpose(n.log10(HM+1)))
    cb=plt.colorbar()
    cb.set_label("$\log_{10}$ (counts)")
    plt.xlabel("Meridional distance $\log_{10}$ (km)")
    plt.ylabel("Temporal distance $\log_{10}$ (s)")

    ↪  plt.title("Distribution of horizontal and temporal lags, h=%1.0f km"%(rg))

    ↪  plt.savefig("C:/Users/OleK/Master_thesis/figs/fig_%s_%1.2f.png"%(fpref,rg))
    plt.close()
    plt.clf()




    ho=h5py.File("hist_%1.2f.h5"%(rg),"w")
    ho["time_diff_log10_s"]=tdist_bins
    ho["horizontal_diff_log10_km"]=hor_dist_bins
    ho["H_hor"]=HH
    ho["H_zon"]=HZ
    ho["H_mer"]=HM
    ho["tbins"]=tdist_bins
    ho["hx"]=hxe
    ho["hy"]=hye
    ho["zx"]=zxe
    ho["zy"]=zye
    ho["mx"]=mxe
    ho["my"]=mye
    ho.close()

d.close()
```

```
from mpl_toolkits.basemap import Basemap

import numpy as n
import matplotlib.pyplot as plt
import cfi_config as c
import mmaria_read as mr
import stuffr


#works for mmaria_data
md=mr.mmaria_data(c.data_directory)#for many files in a directory
b=md.get_bounds()

t0=stuffr.date2unix(2019,1,1,0,0,0)
t1=stuffr.date2unix(2019,1,2,0,0,0)

# about two weeks of data in June 2019
d=md.read_data(t0,t1)

# direction cosine filter
dcos_thresh=0.8
lats=d["lats"]
lons=d["lons"]
dcoss=d["dcos"]
dcos2=n.sqrt(dcoss[:,0]**2.0+dcoss[:,1]**2.0)
ok_idx=n.where(dcos2 < dcos_thresh)[0]

lat0=n.median(lats)
lon0=n.median(lons)

m=Basemap(projection="stere",
          lat_0=lat0,
          lon_0=lon0,
          llcrnrlat=66,
          urcrnrlat=72,
          llcrnrlon=11,
          urcrnrlon=32,
          resolution="h")
```

```
m.drawcoastlines()

pars=n.arange(67,72,2)
m.drawparallels(pars,labels=pars)

mers=n.arange(15,30,5)
m.drawmeridians(mers,labels=mers)

x,y=m(lons[ok_idx],lats[ok_idx])
plt.plot(x,y,".",alpha=0.1)
plt.show()
```

```
#!/usr/bin/env python

import glob
import numpy as n
import matplotlib.pyplot as plt
import h5py
import datetime
import cfi_config as conf

fl=glob.glob("%s/*.h5"%(conf.data_directory))
fl.sort()
n_files=len(fl)

links=["Alta_Alta","Andenes_Andenes","Andenes_Straumen","Tromso_Tromso","All"]

for link in links:
    print(link)
    counts=[]
    countso=[]
    dates=[]

    for f in fl:
        h=h5py.File(f,"r")
        if link=="All":
            idx=n.arange(len(h["link"].value))
        else:
            idx=n.where(h["link"].value==link)[0]
        if len(idx)> 0:
            # it seems like sometimes the measurements are included twice.
            counts.append(len(n.unique(h["t"].value[idx])))
            print(n.unique(h["link"].value))
            countso.append(len(h["t"].value[idx]))
            dt_object =
            ↪  datetime.datetime.fromtimestamp(n.min(h["t"].value[idx]))
            dates.append(dt_object)
        h.close()
    plt.plot(dates,counts,label=link)

plt.legend()
plt.xlabel("Date (UTC)")
plt.ylabel("Counts per day")
plt.title("MMARIA Norway\nunique detections per day")

plt.show()

```

```
#!/usr/bin/env python

import numpy as n
import matplotlib.pyplot as plt
import glob
import scipy.signal as ss
import h5py
import cfi_dac as cfi

name="summer_tacf_koki"
#name="winter_tacf_koki"
fl=glob.glob("D:\Data\mpi\mpi/%s/*.h5"%(name))
fl.sort()
h=h5py.File(fl[0],"r")
tau=n.copy(h["tau"].value)
acfs=n.copy(h["acfs"].value)
acfs[:,:]=0.0
ws=n.copy(h["acfs"].value)
ws[:,:]=0.0
print(ws)
print(acfs)

h.close()
n_avg=1.0
for f in fl:
    h=h5py.File(f,"r")
    w=1/(h["errs"].value)

    wacf=w*n.copy(h["acfs"].value)
    # print(n.sum(wacf))
    # print(n.sum(n.isnan(wacf)))

  #  print(n.sum(w))
   # print(n.sum(n.isnan(w))    )
    if (n.sum(n.isnan(wacf))) == 0:
        acfs+=wacf
        ws+=w


    n_avg+=1.0
    if False:
        plt.plot(tau,w[:,0])
        plt.show()

        plt.plot(tau,h["acfs"].value[:,0])
        plt.plot(tau,h["acfs"].value[:,1])
        plt.show()

    h.close()
print(ws)
acfs=acfs/ws
wf=ss.hann(len(tau)*2)[len(tau):(2*len(tau))]
plt.subplot(121)
plt.plot(tau/(24*3600),acfs[:,0],label="$G_{uu}$")
plt.plot(tau/(24*3600),acfs[:,1],label="$G_{vv}$")
plt.xlabel("Time lag (days)")
plt.ylabel("Correlation function (m$^2$/s$^2$)")
plt.legend()

plt.subplot(122)
wf=1.0

Suu=n.fft.hfft(wf*acfs[:,0])
Svv=n.fft.hfft(wf*acfs[:,1])

dt=n.diff(tau)[0]
f=n.fft.fftfreq(len(Suu),d=dt)
#print(Suu)
Suu[0]=0
fi=n.argmax(Suu)
print(fi)
print(Suu[fi])
f0=n.abs(f[fi])
print(f0)
#b=Suu[fi]/f0**(a)

plt.loglog(n.real(n.fft.fftshift(f)),n.fft.fftshift(Suu),label="$\hat{G}_{uu}$")
plt.loglog(n.real(n.fft.fftshift(f)),n.fft.fftshift(Svv),label="$\hat{G}_{vv}$")

b=Suu[fi]/f0**(-5/3.0)
print(b)
plt.loglog(n.abs(n.fft.fftshift(f)),0.1*b*n.abs(n.fft.fftshift(f))**(-5/3.0),label="-5/3",alpha=
b=Suu[fi]/f0**(-2)
plt.loglog(n.abs(n.fft.fftshift(f)),0.1*b*n.abs(n.fft.fftshift(f))**(-2),label="-2",alpha=0.3)
plt.legend()
plt.xlabel("Frequency (Hz)")
plt.ylabel("Power spectral density (m$^2$/s$^2$/Hz)")
plt.axvline(1.0/(24*3600.0),color="gray",linestyle="--")
plt.axvline(1.0/(12*3600.0),color="gray",linestyle="--")
plt.axvline(1.0/(8*3600.0),color="gray",linestyle="--")
plt.tight_layout()
plt.show()


```

```
import time
import datetime
import numpy as n
import h5py
import matplotlib.pyplot as plt

```

```python
# our internal modules
import mmaria_read as mr
import cfi_dac as cfi
import cfi_config as c
import mean_wind_est as mw
import glob

def read_acf(name="summer_88_vacf"):
    fl=glob.glob("mpi/%s/*.h5"%(name))
    fl.sort()

    print(fl)
    h=h5py.File(fl[0],"r")
    acf=n.copy(h["acf"].value)
    err=n.copy(h["err"].value)
    s_z=n.copy(h["s_z"].value)
    dtau=n.copy(h["dtau"].value)
    h0=n.copy(h["h0"].value)
    #ds_h=n.copy(h["ds_h"].value)
    h.close()
    acf[:,:]=0
    err[:,:]=0
    all_acfs=[]
    all_errs=[]
    all_sz=[]
    n_avg=0
    for f in fl:
        h=h5py.File(f,"r")
        #   plt.plot(h["acf"].value[:,0])
        #    plt.show()
        if n.sum(n.isnan(h["acf"].value)) == 0:
            all_acfs.append(n.copy(h["acf"].value))
            all_errs.append(n.copy(h["err"].value))
            all_sz.append(n.copy(h["s_z"].value))
            n_avg+=1
        h.close()
    all_acfs=n.array(all_acfs)
    all_errs=n.array(all_errs)
    all_sz=n.array(all_sz)
    err_vars=n.zeros([len(s_z),6])
    acfs=n.zeros([len(s_z),6])
    s_z[:]=0.0
    for i in range(len(s_z)):
        s_z[i]=n.mean(all_sz[:,i])
        for ci in range(6):
            err_vars[i,ci]=n.var(all_acfs[:,i,ci])
            ws=0.0
            for mi in range(int(n_avg)):
                if n.sum(n.isnan(all_acfs[mi,i,ci]))== 0:
                    w=1.0/all_errs[mi,i,ci]
                    ws+=w
                    acfs[i,ci]+=w*all_acfs[mi,i,ci]
                else:
                    print("nans")
            acfs[i,ci]=(1.0/ws)*acfs[i,ci]
            err_vars[i,ci]=1/ws

    return(s_z,acfs,err_vars)

s_z,acfs_h,errs= read_acf(name="summer_vacf")
s_z,acfs_l,errs_l= read_acf(name="summer_vacf_ls")

acfs=acfs_h#-acfs_l
plt.subplot(121)
plt.plot(s_z,acfs_h[:,0],label=cfi.cf_names[0])
plt.plot(s_z,acfs_l[:,0],label=cfi.cf_names[1])
#plt.plot(s_z,acfs_h[:,1])
#plt.plot(s_z,acfs_l[:,1])
plt.xlabel("Vertical lag (km)")
plt.ylabel("Correlation function (m$^2$/s$^2$)")
plt.title("Autocorrelation function")
plt.subplot(122)
plt.loglog(s_z,2*1.2*acfs[1,0]-2*acfs[:,0])
plt.loglog(s_z,2*1.2*acfs[1,1]-2*acfs[:,1])
plt.loglog(s_z,100.0*s_z**(2.0/3.0))
plt.xlabel("Vertical lag (km)")
plt.ylabel("Structure function (m$^2$/s$^2$)")
plt.title("Structure function")
plt.show()
```

```
import time
import datetime
import numpy as n
import h5py
import matplotlib.pyplot as plt

# our internal modules
import mmaria_read as mr
import cfi_dac as cfi
import cfi_config as c
import mean_wind_est as mw
import glob

name="summer_hacf"
#name="winter_hacf"
#name="summer_hp_hacf"

fl=glob.glob("D:\Data\mpi\mpi/%s/*.h5"%(name))
fl.sort()

print(fl)
h=h5py.File(fl[0],"r")
acf=n.copy(h["acfs"].value)
err=n.copy(h["errs"].value)
s_h=n.copy(h["s_h"].value)
dtau=n.copy(h["dtau"].value)
ds_z=n.copy(h["ds_z"].value)
```

```
ds_h=n.copy(h["ds_h"].value)
h.close()
acf[:,:]=0
err[:,:]=0
all_acfs=[]
all_errs=[]
all_sh=[]
n_avg=0.0
for f in fl:
    h=h5py.File(f,"r")
 #   plt.plot(h["acfs"].value[:,0])
#    plt.show()
    if n.sum(n.isnan(h["acfs"].value)) == 0:
        all_acfs.append(n.copy(h["acfs"].value))
        all_errs.append(n.copy(h["errs"].value))
        all_sh.append(n.copy(h["s_h"].value))
        n_avg+=1.0
    h.close()


all_acfs=n.array(all_acfs)
all_errs=n.array(all_errs)
all_sh=n.array(all_sh)
err_vars=n.zeros([len(s_h),6])
acfs=n.zeros([len(s_h),6])
s_h[:]=0.0
for i in range(len(s_h)):
    s_h[i]=n.mean(all_sh[:,i])
    for ci in range(6):
        err_vars[i,ci]=n.var(all_acfs[:,i,ci])
        ws=0.0
        for mi in range(int(n_avg)):
            if n.sum(n.isnan(all_acfs[mi,i,ci]))== 0:

                w=1.0/all_errs[mi,i,ci]
#                print(w)
                ws+=w
                acfs[i,ci]+=w*all_acfs[mi,i,ci]
            else:
                print("nans")
#        print(ws)
        acfs[i,ci]=(1.0/ws)*acfs[i,ci]


names=["$G_{uu}$","$G_{vv}$","$G_{ww}$",
       "$G_{uv}$","$G_{uw}$","$G_{vw}$"]
colors=["C0","C1","C2","C3","C4","C5"]
cfi.plot_hor_acfs(shs=s_h,
                  names=names,
                  acfs=acfs,
                  ds_z=1.0,
                  dtau=dtau,
                  ds_h=ds_h,
                  err_vars=err_vars,
                 colors=colors,
                 zlag=1.1,
                 n_avg=n_avg)

    # plt.savefig("C:/Users/OleK/Master_thesis/figs/fig_%s.png"%(name))
    # ho=h5py.File("res/%s.h5"%(name),"w")
    # ho["acf"]=acfs
    # ho["s_h"]=s_h
    # ho["err_var"]=err_vars/n.sqrt(n_avg)
    # ho["h0"]=h0
    # ho["dtau"]=dtau
    # ho["ds_h"]=ds_h
    # ho.close()




```

Bibliography

Alexander, M., Geller, M., McLandress, C., Polavarapu, S., Preusse, P., Sassi, F., Sato, K., Eckermann, S., Ern, M., Hertzog, A., et al. (2010). Recent developments in gravity-wave effects in climate models and the global distribution of gravity-wave momentum flux from observations and models. *Quarterly Journal of the Royal Meteorological Society*, 136(650):1103–1124.

Andrews, D. and Mcintyre, M. E. (1976). Planetary waves in horizontal and vertical shear: The generalized eliassen-palm relation and the mean zonal acceleration. *Journal of the Atmospheric Sciences*, 33(11):2031–2048.

Bacmeister, J. T., Eckermann, S. D., Newman, P. A., Lait, L., Chan, K. R., Loewenstein, M., Proffitt, M. H., and Gary, B. L. (1996). Stratospheric horizontal wavenumber spectra of winds, potential temperature, and atmospheric tracers observed by high-altitude aircraft. *Journal of Geophysical Research: Atmospheres*, 101(D5):9441–9470.

Chau, J. L., Urco, M., Vierinen, J., Volz, R., Clahsen, M., Pfeffer, N., and Trautner, J. (2019). Novel specular meteor radar systems using coherent mimo techniques to study the mesosphere and lower thermosphere.

Drob, D., Emmert, J., Crowley, G., Picone, J., Shepherd, G., Skinner, W., Hays, P., Niciejewski, R., Larsen, M., She, C., et al. (2008). An empirical model of the earth's horizontal wind fields: Hwm07. *Journal of Geophysical Research: Space Physics*, 113(A12).

Hedin, A. E. (1991). Extension of the msis thermosphere model into the middle and lower atmosphere. *Journal of Geophysical Research: Space Physics*, 96(A2):1159–1172.

Hocking, W. (1999). Temperatures using radar-meteor decay times. *Geophysical Research Letters*, 26(21):3297–3300.

Huang, F., Mayr, H., Reber, C., Russell, J., Mlynczak, M., and Mengel, J. (2006). Stratospheric and mesospheric temperature variations for the quasi-biennial and semiannual (qbo and sao) oscillations based on measurements from saber (timed) and mls (uars).

Kolmogorov, A. N. (1941). The local structure of turbulence in incompressible viscous fluid for very large reynolds numbers. *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences*, 434(1890).

Liu, H.-L. (2019). Quantifying gravity wave forcing using scale invariance. *Nature communications*, 10.

McLandress, C. (1998). On the importance of gravity waves in the middle atmosphere and their parameterization in general circulation models. *Journal of Atmospheric and Solar-Terrestrial Physics*, 60(14):1357–1383.

MJ, T., WR Jr, P., et al. (1999). Comparison of terdiurnal tidal oscillations in mesospheric oh rotational temperature and na lidar temperature measurements at mid-latitudes for fall/spring conditions. *Earth, planets and space*, 51(7-8):877–885.

Nastrom, G., Gage, K., and Jasperson, W. (1984). Kinetic energy spectrum of large-and mesoscale atmospheric processes. *Nature*, 310(5972):36–38.

Pancheva, D., Mukhtarov, P., Andonov, B., Mitchell, N. J., and Forbes, J. (2009). Planetary waves observed by timed/saber in coupling the stratosphere–mesosphere–lower thermosphere during the winter of 2003/2004: part 1—comparison with the ukmo temperature results. *Journal of Atmospheric and Solar-Terrestrial Physics*, 71(1):61–74.

Rossby, C.-G. (1939). Relation between variations in the intensity of the zonal circulation of the atmosphere and the displacements of the semi-permanent centers of action. *J. Mar. Res.*, 2:38–55.

Smith, A. K. (2012). Global dynamics of the mlt. *Surveys in Geophysics*, 33(6):1177–1230.

Smith, A. K., Pancheva, D. V., and Mitchell, N. J. (2004). Observations and modeling of the 6-hour tide in the upper mesosphere. *Journal of Geophysical Research: Atmospheres*, 109(D10).

Stober, G. and Chau, J. (2015). A multistatic and multifrequency novel approach for specular meteor radars to improve wind measurements in the mlt region. *Radio Science*, 50(5):431–442.

Stober, G., Chau, J. L., Vierinen, J., Jacobi, C., and Wilhelm, S. (2018). Retrieving horizontally resolved wind fields using multi-static meteor radar observations.

Vierinen, J., Chau, J. L., Charuvil, H., Urco, J. M., Clahsen, M., Avsarkisov, V., Marino, R., and Volz, R. (2019). Observing mesospheric turbulence with specular meteor radars: A novel method for estimating second-order statistics of wind velocity. *Earth and Space Science*, 6(7):1171–1195.

Von Zahn, U., Höffner, J., Eska, V., and Alpers, M. (1996). The mesopause altitude: Only two distinctive levels worldwide? *Geophysical research letters*, 23(22):3231–3234.