Faculty of Science and Technology
Department of Physics and Technology

Kelvin-Helmholtz versus Gradient drift instability: which one "wins" in the high latitude ionosphere?

Amalie Marie Kjørsvik

FYS-3931 Master's thesis in Space Physics - January 2024

Abstract

The high latitude ionosphere is highly irregular, which is caused by plasma patches, flow shears and electron precipitation. These causes are not well understood, but it is thought to be three sources to these irregularities: density gradients, arcs/flow channels and electron precipitation. It is speculated that there are two main mechanisms for the formation of these irregularities. These mechanisms are Kelvin-Helmholtz Instability (KHI), a instability with flow shears as a source and Gradient Drift Instability (GDI), which have plasma density gradient in plasma patches as a source. The reason for the interest in these irregularities, are the technological problems they cause. Signals from satellites can be shifted away from intended targets or reflected back from the atmosphere completely. The goal of the thesis is to compute the linear growth rate of these instabilities to be able to see which one is more likely to be the dominant one. I used the Swarm satellites to access data for 2015 and 2020 as they are at approximately solar maximum and minimum respectively. The data was accessed from 60 degrees latitude and upwards, as the area of interest were the auroral region and the polar cap. I used both latitude and Kp as means of restriction for the datasets, so I could analyse and compare differences for the auroral region and polar cap, geomagnetic activity and seasonal variations. I also compare the differences between the solar maximum and minimum. The results indicate that the GDI is the more dominant mechanism from the assumptions made in the thesis, as it had a consistently larger growth rate than KHI. The GDI growth rate also seems to be affected by the seasonal variations as there is more GDI growth rate in the winter months. The KHI growth rate however seems to be invariant of the seasons. The GDI growth rate also reaches larger values in 2015 than in 2020, which might have to do with the solar maximum, while the KHI growth rate is pretty consistent throughout all the results.

Acknowledgements

First I need to thank my supervisor Andres Spicher for all the support and help throughout the year. Your help has been invaluable for the completing of the thesis.
I am grateful for all my friends and family that has been there and made life fun and interesting during these last couple of years.
Lastly a huge thank you to Herman for keeping me motivated and doing whatever you could so I could succeed. Thank you! This study makes use of data from the Swarm spacecraft mission which is funded and managed by the European Space Agency. Swarm data can be accessed online at: http://earth.esa.int/swarm. I also made use of the VirES for Swarm webpage, and example code. This can be accessed at: https://vires.services/.

Contents

List of Figures

1 Introduction

Plasma patches, flow shears and electron precipitation creates irregularities in the ionosphere. These irregularities can cause technological problems, especially in the form of GPS signals from satellites being shifted away from the intended target, or reflected away from the atmosphere completely (Oksavik et al., 2012). The causes of these irregularities are not well understood however, there is thought to be three different sources: density gradients, arcs/ flow channels and electron precipitation. This thesis will look at what is considered the two main mechanisms, Gradient-Drift Instabilities (GDI) and Kelvin-Helmholtz Instabilities (KHI). The GDI has density gradients in plasma patches as the source, while the KHI has arcs/flow channels as the source (Oksavik et al., 2012).

Plasma patches are areas of plasma with at least twice the density as the background plasma (Spicher et al., 2015). They move over the polar cap and are impacted by the solar cycle, the geomagnetic activity and the seasons. Irregularities can be found in the whole patch, which can be from 100-1000km wide in the horizontal direction, however they seem to be more prominent on the trailing edges (Oksavik et al., 2012).

In the ionosphere there are flow channels and arcs. These impact the convection of the ionosphere depending on which direction they are from, the time and the seasons (Lockwood et al., 1990). When large velocity shears transverse the magnetic field irregularities can occur. These flow shears can be both perpendicular and parallel to the magnetic field, however this does not seem to have an impact on irregularity growth (Oksavik et al., 2012).

This thesis will try and understand which of these mechanisms seems to be the

most dominant. This is done with an analysis with two different sets of data, one for 2015 and one for 2020. The two years were chosen based on the solar cycle maximum and minimum, to see if there is a difference. This thesis will focus on the northern polar area of 60 degrees and upwards. The latitude will be divided into two regions, the auroral region from 60 to 75 degrees (Kataoka, Ryuho and Nakano, Shin'ya, 2021) and the polar cap from 75 to 90 degrees. This to see if the difference in activity in these regions would matter for the irregularities. There is also a restriction on Kp which are divided into three, 1-3, 3-6 and 6-9. The Kp index is chosen because it is a good indication of the geomagnetic activity, and it is divided up to look at the difference for different geomagnetic activity (Matzka et al., 2021).
I determine the more dominant instability by calculating the linear growth rate for both GDI and KHI using in situ measurements from the Swarm Alpha satellite. This is one of three identical satellites in a constellation from the European Space Agency's (ESA) first mission of Earth Observation (ESA, 2023a). They orbit the entirety of the Earth and uses approximately 95 minutes on one orbit (Team, 2023). The assumption made is that the larger the growth rate the more dominant the irregularity mechanism is. In Oksavik et al. (2012) they used in situ measurements from a sounding rocket together with ground-based data to look at spatial structures and irregularities in the F-region of the ionosphere. The time of their measurements are therefore only when the rocket are in the F-region and over a limited area. In Burston et al. (2016) they use the Dynamics Explorer 2 satellite over approximately 2 years to look at GDI, KHI current convective and a small scale "turbulance" process. This however was only during the peak of the solar cycle, and they could therefore not compare it with the solar minimum. In this thesis I try to expand the time and space the measurements are taken, and then compare two different years at different time in the solar cycle. This will hopefully give some insight on how the geomagnetic activity and region of latitude impacts the growth rates of the instabilities.
The thesis is divided into 7 chapters as well an appendix. The first chapter is the introduction. In the second chapter I present background information relevant to my investigation. The third chapter is instrumentation where I give an overview of the satellites and instruments used for data collection. The fourth chapter is the methodology where I describe how I worked to find the results, which are presented in the fifth chapter. In the sixth chapter I discuss my results in the context of already existing literature, and in the seventh chapter I present my conclusion. In the appendix the code used for the calculations is presented.

2 Background

In this section I will go through the necessary background information, so that the results will be easily understood. As this thesis is about irregularities in the high latitude ionosphere in polar area, I will introduce the ionosphere and some consequences of the magnetosphere-ionosphere coupling. This will be followed by the theory for the Kelvin-Helmholtz and Gradient-Drift irregularities.

2.1 Ionosphere

The region in the atmosphere which is partially ionized is called the ionosphere (Brekke, 2012). The two things needed for a ionosphere is a neutral atmosphere and a source of ionization. The two most common are photoionization and impact ionization (Russell et al., 2016). Photoionization is ionization of atmospheric constituent due to interactions between the solar photons. It is primarily because of solar photons ultraviolet wavelengths. Impact ionization is collisions between ionized particles and neutral constituents. Photoionization can be assumed as the most dominant source of ionization in the ionosphere for the most part. Impact ionization is sustaining the high-latitude ionosphere on the nightside, and is therefore more dominant than photoionization in this part of the ionosphere (Russell et al., 2016). The ionosphere consists of three different regions: The D-, E-, and F-region. The D-region exists at an altitude below 90 km, the E-region is between 90 and 130 km, while the F-region is above 130 km (Russell et al., 2016). As the ionosphere is only a region of ion-

ized particles, it varies depending on solar radiation, precipitating particles, aurora particles, time of the day and seasons (Brekke, 2012). For my thesis I have looked at the F-region of the ionosphere in the auroral region, which is defined as a latitude from 60-75 degrees (Kataoka, Ryuho and Nakano, Shin'ya, 2021) as well as the polar cap which is approximately from a latitude of 75-90 degrees as it is defined as being above the auroral region.

2.1.1 Plasma

Plasma is defined as a gas of free electrons and ions that behaves in a collective way, which means that the plasma as a whole is neutral, however the different particles and parts of the plasma is not neutral. This is called quasi-neutrality (Piel, 2017). Because of the particles that make up the plasma, it can be acted upon by both electric and magnetic fields (Russell et al., 2016). The focus of the thesis is the ionosphere, and as such the plasma discussed will be space plasma.
Plasma patches seems to be an area in which strong irregularities can be observed (Oksavik et al., 2012). To be considered a patch the density of the patch must be at least twice the background density. They can be anything from a few kilometres to a thousand kilometres wide in the horizontal direction (Oksavik et al., 2012). Patches can be observed the whole year, but are more prominent during conditions when Kp $>$ 4, and when we are closer to the maximum of the solar cycle. The patches are observed to drift over the polar cap, and while irregularities have been observed in the entire patch structure, the stronger once have a tendency to be on the trailing edges of the patches (Oksavik et al., 2012). The plasma patches appear to be solar-induced plasma that is transported from lower latitudes in the ionosphere through the polar cap and into the auroral region (Tsunoda, 1988). It is important to the understanding of the irregularities to know that the plasma patches moves, and that there is movement in the ionosphere. It will be discussed in more detail in the next section of the background.

2.2 Magnetosphere-Ionosphere coupling

Earth has its own magnetic field, which is formed by the molten core of metals and stone. As such the Earth is like a dipole magnet, and has its own magnetosphere. The size of the magnetosphere depends on the magnetic pressure of the magnetosphere, the components of the solar-wind pressure and the balance between these two (Russell et al., 2016). There are three components of the solar-wind pressure: momentum flux of the cold stream ions flowing from the Sun, the kinetic or thermal pressure of the solar

wind plasma and the pressure of the interplanetary magnetic field (IMF) (Russell et al., 2016). These push perpendicular along the magnetopause, the outer boundary of the magnetosphere, where the magnetosphere meets the solar wind, as the solar wind passes by the magnetosphere it creates a drag which changes the shape of the magnetosphere, as well as causes the circulation of the magnetospheric plasma and transports magnetic flux. The solar wind reconnects at the magnetopause, coupling the solar wind with the magnetosphere, which causes the magnetic fields to be linked. This means that the magnetic flux from the solar wind is coupled with the magnetosphere, which the magnetospheric plasma carries together with plasma over the polar cap to the magnetotail. Here there is a larger energy reservoir which can be released rapidly, and when released, hit the night side of the planet first(Russell et al., 2016). A simple schematic of these areas of the magnetosphere can be seen in figure 2.1. This means that the solar wind is a primary driver for the circulation of plasma.

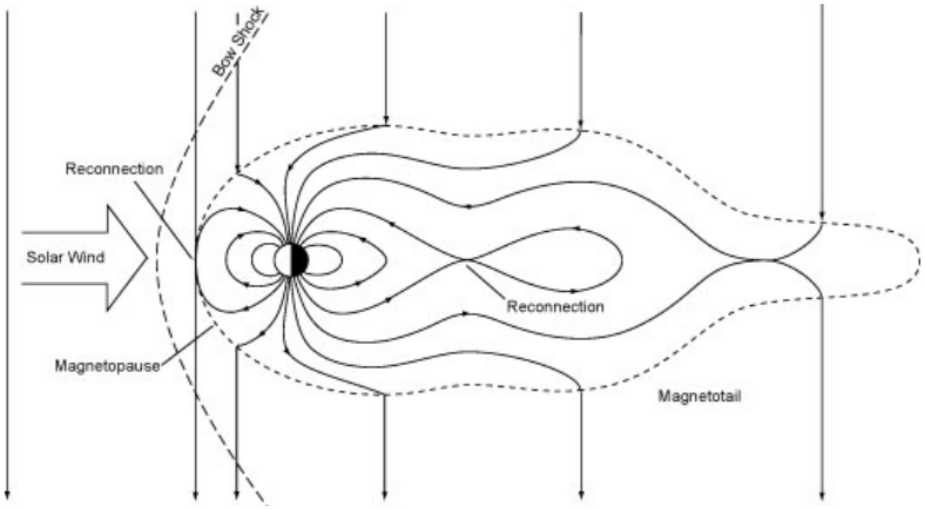

Figure 2.1: The figure shows the magnetosphere with the magnetotail, magnetopause and reconnection. Figure from Bahmer (2024)

In short the magnetic field of the Earth and the ionosphere is heavily influenced by the solar wind, and the IMF. The magnetic field and the ionosphere is heavily interconnected, which is to be taken into consideration when studying the ionosphere.
As a consequence of the magnetosphere-ionosphere coupling we have plasma convection in the ionosphere. One of the important parameters is the interplanetary magnetic field (IMF), as the way it changes in magnitude and direction contributes to the convection patterns in the high-latitude ionosphere (Heelis, 1988). Ionospheric flows are a result of the solar wind coupling with the mag-

netosphere. The flow can be viewed as two separate flows, decoupled from each other, and are time-dependent (Lockwood et al., 1990). The flow pattern happening on the dayside of the magnetopause, is driven by a direct coupling of the solar wind and magnetosphere. This flow is associated with an expanding polar cap (Lockwood et al., 1990). The flow pattern happening on the nightside of the magnetopause is driven by flux return from the geomagnetic tail. This comes from the magnetic reconnection during substorms when energy is released from the magnetotail. The nightside flow pattern is associated with a contracting polar cap (Lockwood et al., 1990).
As the IMF has a northward direction it is possible for the convection pattern to go from a two-cell to a four-cell pattern. However when the IMF is southward it seems like the plasma patches is formed (Heelis, 1988). There is also variations in the plasma convection patterns, that comes from the geographic and geomagnetic poles are misaligned. So the convection patterns reflects the state of the cusp, which the plasma convects through from the auroral region to the polar cap (Heelis, 1988). In figure 2.2 we can see the main ionospheric two-cell convection pattern, during an aurora, with southern IMF. This is from the northern hemisphere above 50 degrees latitude in magnetic local time.

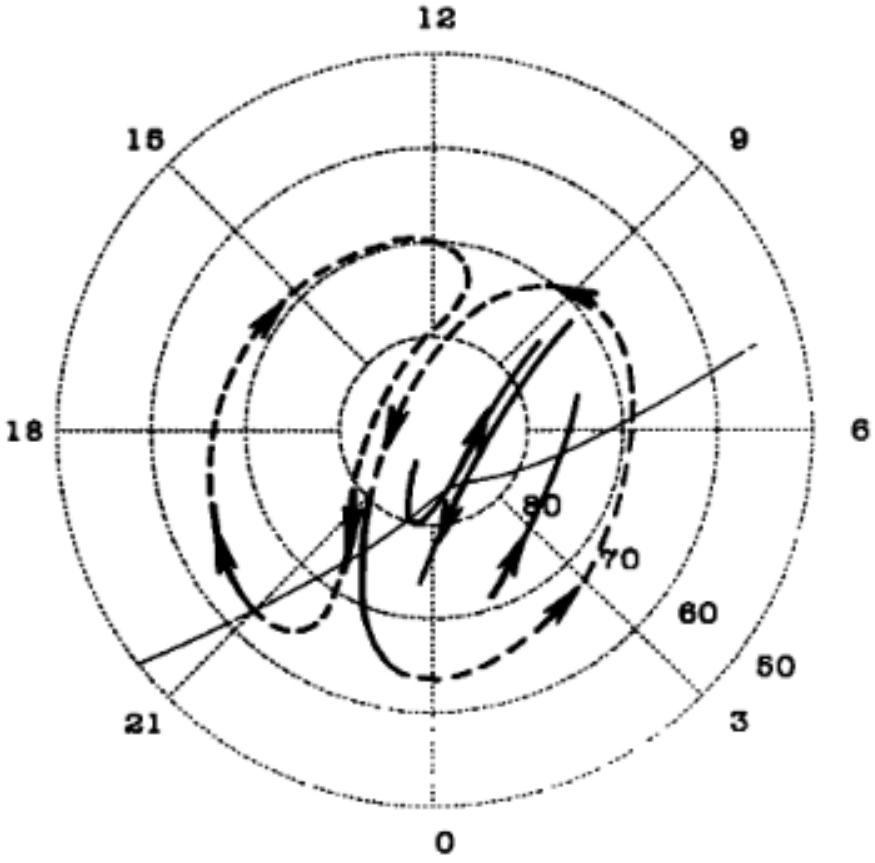

Figure 2.2: Main ionospheric convection pattern during an aurora. Figure taken from Heelis (1988)

The magnetosphere-ionosphere coupling, seems to also have a impact on the aurora. The EUV flux controls the background density of the ionospheric plasma, which in turn controls to which degree intense auroral processes can operate (Newell et al., 2001). The aurora can then be a sign of the magnetospheric processes driving currents into the ionosphere (Newell et al., 2001). Said in other words, the aurora and its strength could give us information on the state of the ionospheric currents and convections.

There are also seasonal variations in the high-latitude ionosphere, which come from the tilt of the planet. In the summer-period for the northern hemisphere the polar cap is drenched in sunlight. This can cause the background density to be much larger than during the wintertime (Spicher et al., 2017). The sunlight can also be the cause of for the rate of density index is high from

September to April. The solar EUV ionizes the E-region, which can result in irregularities decaying quicker in the F-region (Jin et al., 2019).

All of these processes follows from the magnetosphere-ionosphere coupling. Since a lot of the processes written about is heavily influenced by the solar wind, one way of monitoring the variations is by using the Kp index. This is a standardized index to monitor geomagnetic disturbances on a global scale. The index ranges from 0-9 and is quasi-logarithmic (Matzka et al., 2021). I have used Kp later in the thesis as one of the parameters of interest.

2.3 Instabilities

The irregularities in the ionosphere are poorly understood, however there are two main mechanisms that are believed to be the cause of the formations (Carlson and Moen, 2008). The first is the GDI which occurs on electron density gradients, which can be encountered in polar cap patches. Polar cap patches are plasma density islands, with an area of several hundred kilometres, and a density more than double the background density (Spicher et al., 2015). They are to be found traversing the polar cap. In the ionosphere the plasma density gradient and the electric field can create unstable polarized electric fields. This then creates a positive feedback loop, in which the initial perturbation from the creation of this unstable electric field becomes amplified by the gradient drift motion moving less dense plasma to areas with higher density plasma and vice versa. This is the process thought to create GDI (Tsunoda, 1988).
The second instability is the KHI which is associated with flow shears. It is thought to occur when there are large velocity shears transverse the magnetic field (Oksavik et al., 2012). There are two different types of flow shears, when the plasma flow velocity is either parallel or perpendicular to the magnetic field. KHI can be the outcome for both types, however it seems that the growth rate appears to be slow, no matter which type (Keskinen et al., 1988). There are several other mechanisms that are believed to contribute to the irregularities in the ionosphere, these however is not discussed in this thesis.

The equations used for calculating the linear growth rates for the Gradient Drift Instability (GDI) and Kelvin-Helmholtz Instability (KHI) are (Oksavik et al., 2012):

$$\gamma GD = \frac{V_0 \Delta N}{N_0 \Delta x} \tag{2.1}$$

$$\gamma KH = \frac{0.2 \Delta V}{L} \tag{2.2}$$

For equation 2.1 V_0 is the plasma drift relative to the neutral atmosphere in the direction parallel to the density gradient, N_0 is the background density, $\frac{\Delta N}{\Delta x}$ is the horizontal electron density gradient. Here $\Delta x = \Delta t * [V_r + V_{ExB}]$, where V_r is the horizontal velocity of the satellite, Δt is the time between data samples and V_{ExB} is the ion drift along the satellite track (Oksavik et al., 2012). For equation 2.2 ΔV is velocity difference and L is the velocity difference scale length (Oksavik et al., 2012). The reason to look at the growth rates is that they can give an indication of which instability is more dominant, as the faster the instabilities grow, the more irregularities are created (Oksavik et al., 2012).

3 Instrumentation

In this section I will discuss the satellites and instruments used, to collect the analysed data.

3.1 Satellites

I have used measurements collected from the European Space Agency's (ESA) satellite constellation Swarm. The constellation consists of three identical satellites, Alpha, Bravo and Charlie, and is ESA's first mission of Earth Observation (ESA, 2023a). The Alpha and Charlie satellites are in the same orbital plane, and have the same altitude of 462 km, while the Bravo satellite has an altitude of 511 km. Alpha and Charlie has a distance of 15 km or 2 seconds in the orbit (ESA, 2023a). The satellites were launched in a near polar orbit in 2013, and the mission was only supposed to last until 2017, however it has now been renewed to 2025. They orbit the entirety of the Earth and uses approximately 94 minutes on one orbit (Team, 2023). The satellites measure the geomagnetic field, the velocity which the electric field in the ionosphere can be calculated from, as well as provide information about the dynamics in the ionosphere (ESA, 2023a).

3.2 Instruments and data

For the analysis I used measurements of electron density, ion drifts cross-track velocity in the horizontal direction in satellite-track coordinates, ion-drifts along-track velocities from the vertical and horizontal TII sensor, the satellite velocity in the North-East-Center (NEC) frame, specifically in the North and East direction. I decided to use data only from the Alpha satellite, the reason which will be discussed further in the Methodology chapter. The instrument used to gather these specific measurements is the Electric Field Instrument (EFI) a 3D ionospheric imager. It measures plasma density, drift and velocity to characterise the electric field around Earth. It consists of three main parts, the Langmuir Probe, the thermal ion imager and the electronics assembly (ESA, 2023b). All the data is accessed through VirES for Swarm (ESA, 2023c).

4

Methodology

In this section the methodology is presented. Here I present the method of accessing the data, the problems I encountered and the solutions I used. I wanted to be able to use different restrictions so that I could compare the auroral region with the polar cap, compare different geomagnetic activity and look at two different years that correspond with the solar maximum and minimum. The decisions for how the data is used is also presented.

As a start for the code and analysis in this thesis I used the code from my project paper, where I analysed four cases of GDI and KHI. The cases were spread over different days, and the data accessed were from a latitude of 75 degrees and above. There were no restrictions of Kp values. I used VirES for Swarm to access the satellite data. This code is using the datasets "EFI TCT02" and "EFI IDM" to access the different values needed to calculate the growth rates of GDI and KHI. From the "TCT02" dataset I access the ion drifts cross-track velocity in the horizontal direction in satellite-track coordinates needed to calculate the KHI growth rate. For the along track ion drift velocity, the ion density and the satellite velocity in NEC frame needed to calculate the GDI growth rate are accessed from the "IDM" dataset.

The original code was not optimised and there was a lot of manual work required to perform an analysis in more than single cases, be able to select intervals of the time or orbit, or restrict latitude and Kp. Therefore

significant improvements were required. A range filter were applied for the latitude and Kp, so this could be changed easily to fit the restrictions I wanted. I have looked at GDI and KHI growth rates in the aurora oval and in the polar cap. I have also looked at the difference in KHI and GDI growth rates based on different Kp values. The range filter helped with easily being able to change the restrictions to fit what I was looking for at the time.

The next challenge encountered and that needed to be solved was that the datasets from the different instruments weren't necessarily of equal length, and they didn't always have similar timing. Unless I had equal length dataset it was not possible to plot them together, which made it hard to be able to compare and see differences and similarities, as well as making an analysis of the data. I also needed the datapoints to be taken at the same time, so that I knew that they were taken in the same place. A five second shift meant that I would compare two datapoints at different places in the ionosphere, and this would give inaccurate results. So the process of making them equal no matter which dataset was longer needed to be automated. Selecting datapoints which are closer than half a second so the values were taken at approximately the same time also needed to be automated. This was done with an if statement in a for loop. The timesets were converted to UNIX time, however for the plots they were converted back for an easier understanding.

Then I had a problem with the calculated GDI as the magnitude seemed unrealistic as I got values upwards to $10^7/s$. Therefore I put maximum and minimum limits on all the velocity, of which I used 10 000 and -10 000 m/s respectively, and density values, of which I used $1^8 cm^{-3}$ and 0 respectively, to limit the number of non physical values making the problems. The velocity values are much larger than typical ionospheric velocities, which usually reaches a maximum of 1000 m/s. There are however possible to reach larger velocities, so my limits were chosen to not restrict the data too much. Then I also put a median filter on the ion density to remove any spikes from the dataset. Here I decided on a median filter that uses a gradient between 3 points. In figure 4.2 we can see an example of how filtering affects the data. In figure 4.2 the blue dots represent data with no smoothing, the orange dots represent the data with a 3 point gradient median filter, and the green dots represent the data with a 5 point gradient median filter. As can be seen the filtering smoothes some of the gradients, so that the largest outliers are omitted from the results. I found that a 3 point gradient was a good middleground between no smoothing and the 5 point gradient, as some of the outliers are omitted, but not so many to make the data unreliable. This still didn't fix the problem completely, as seen in figure 4.3 where I have plotted the KHI growth rate with blue dots and the GDI growth rate with ion density

with orange dots for January of 2020. The figure shows growth rate on the y-axis and time on the x-axis. Here we can see that GDI reaches a growth rate of 125 000/s which seems unrealistic. This also causes the problem of not being able to visually compare with the KHI as the growth rate becomes approximately zero when plotted together with the GDI.

In the end I found out that the "EFI IDM" dataset had too many uncertainties in their values to be used here, and decided instead to use the dataset "EFI". The "IDM" dataset were originally chosen because it provided Vi both along the orbit and the density which meant not needing to combine different datasets. The "EFI" dataset did not have the velocity along the satellite, which meant combining different datasets for the GDI growth rate calculation. From this dataset I could access the electron density, and for the velocities needed I accessed those from the "TCT02" dataset. Here I chose the ion drifts along-track velocities from the vertical and horizontal TII sensor, and the satellite velocity in the NEC frame, specifically in the North and East direction. I also selected a scale length L of 8000 m. In equation 2.1 the V_0 is the square root of the along-track velocities, ΔN is the electron density, N_0 is the background density and Δx is the velocities in north and east direction as well as Δt. In figure 2.2 ΔV is the cross-track horizontal drift velocity and L is the scale length. In equation 4.4 the KHI growth rate is plotted with blue dots, and the GDI growth rate with electron density instead of ion density is plotted with orange dots. The figure shows the growth rate on the y-axis and the time on the x-axis. The growth rates are plotted over January of 2020. This gave a better result when looking at the GDI magnitude as seen in figure 4.4, where the maximum value of the GDI growth rate is 2/s which is a much more realistic value (Oksavik et al., 2012).
Figure 4.1 shows a sketch of the satellite in orbit from the auroral region and upwards. This also shows all the parameters used to calculate the growth rates.

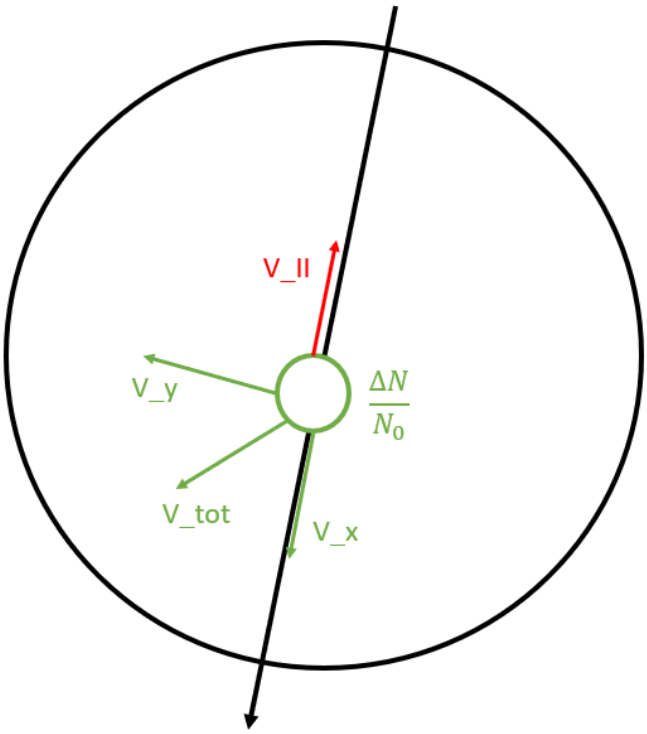

Figure 4.1: A sketch of the satellite orbit in the auroral region, with the different parameters used for the growth rate

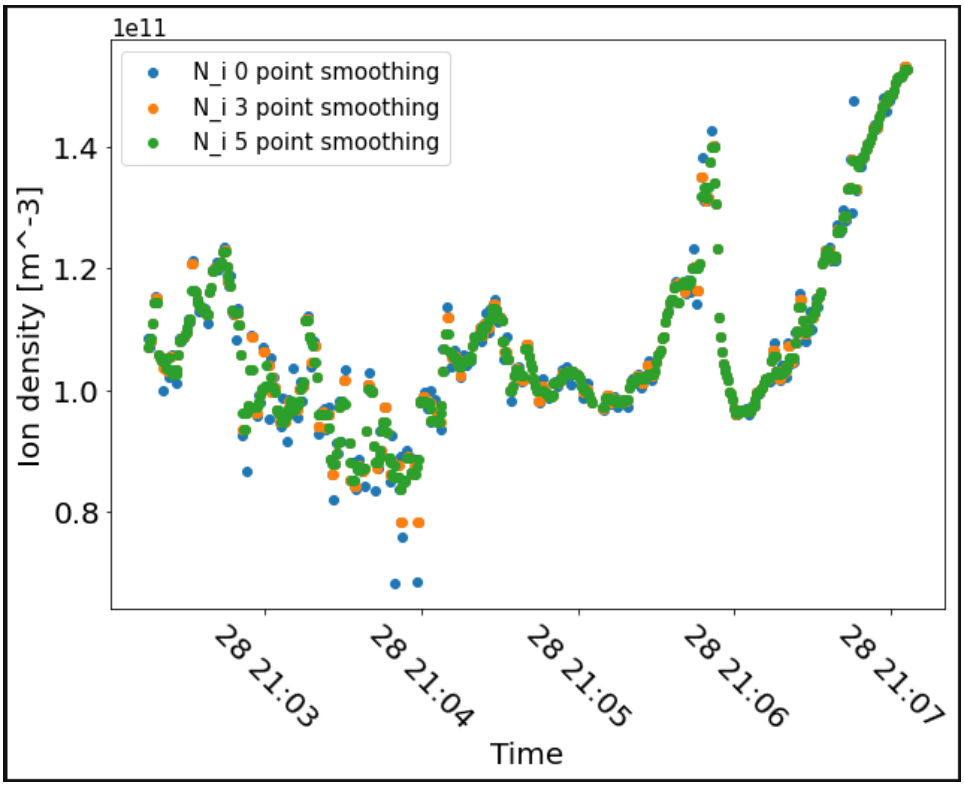

Figure 4.2: N_i density with no smoothing, a 3 point smoothing and a 5 point smoothing of the data. Date 28.06.2015 in the time period 21:00:00-22:00:00 UT

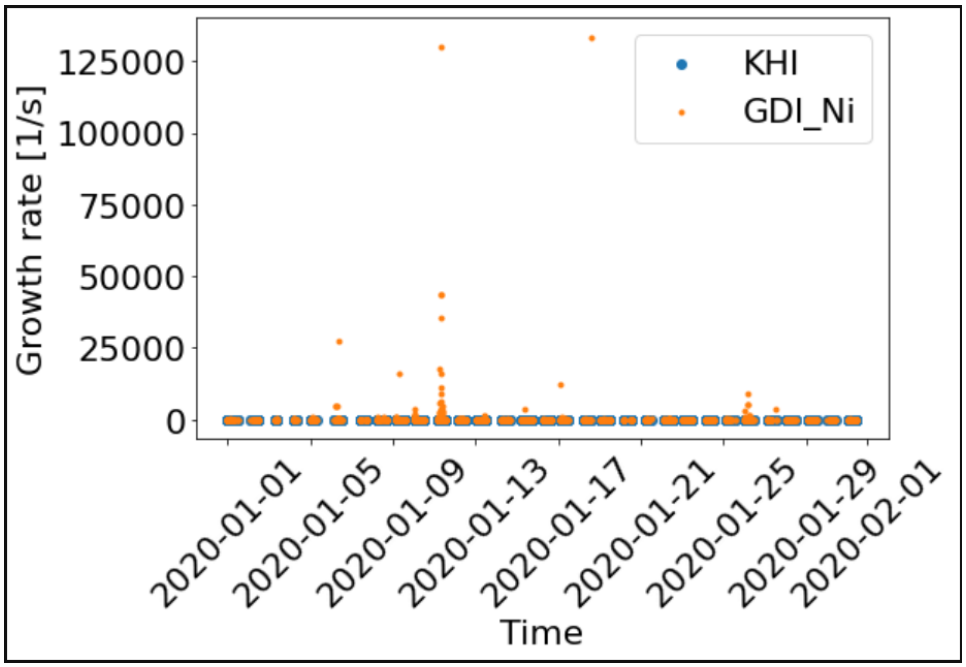

Figure 4.3: KHI growth rate vs GDI growth rate where I use the ion density to calculate GDI growth rate. We can see that there seems to be an unrealistically high growth rate for the GDI, and it is not possible to compare with the KHI growth rate as the GDI growth rate is to large. The GDI growth rate is the orange dots and the KHI growth rate is the blue dots.

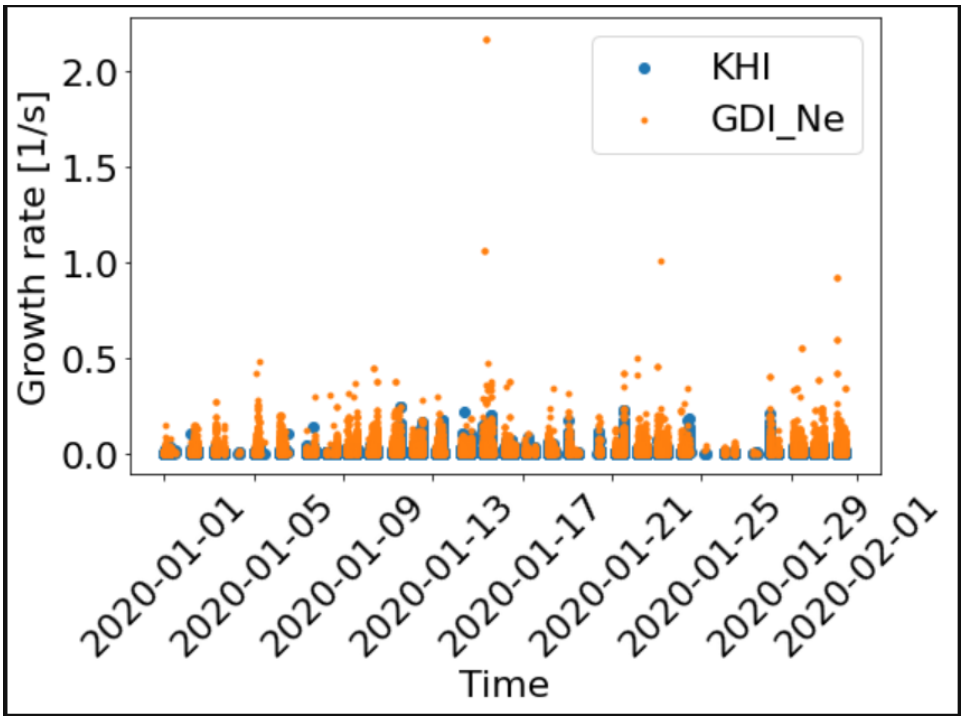

Figure 4.4: KHI growth rate vs GDI growth rate where I use the electron density to calculate the GDI growth rate. GDI growth rate is the orange dots and KHI growth rate is the blue dots.

5

Results

In this section I present the results of my thesis. Here I have decided to show a comparison with one of the cases from my project paper. I also show data to illustrate the solar cycle, the GDI and KHI growth rates for 2015 and 2020 with restrictions for Kp or latitude, as well as histograms of GDI and KHI growth rates for 2015 and 2020. I made the decision to look at Kp and latitude, to see if any of them had an impact on the GDI and KHI growth rates. Kp and latitude are looked at separately. The solar cycle is shown to see if it has any impact on the GDI or KHI growth rate. The latitudes selected are from 60 degrees latitude and upwards, as the thesis is centered around the northern polar area. The latitudes are chosen to separate the auroral region at 60-75 degrees (Kataoka, Ryuho and Nakano, Shin'ya, 2021) and the polar cap.

From figure 5.1 we can see that from the number of observed sunspots, we can infer that there was a solar maximum around 2014-2015, and a solar minimum in 2020. This was used to be able to choose which years to collect data from.

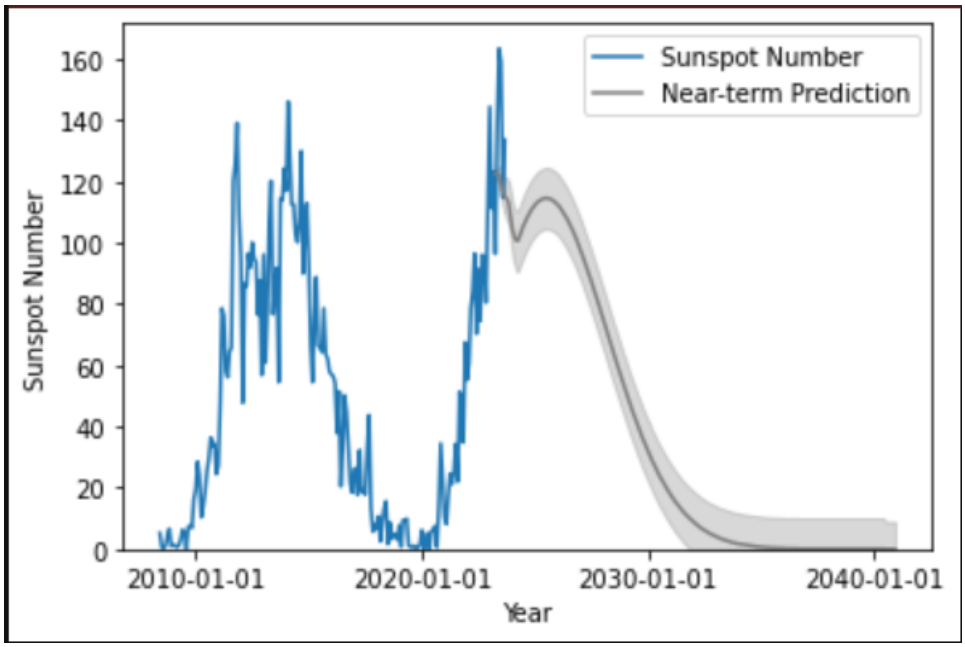

Figure 5.1: Showing the solar cycle, both predicted and observed. (SunPy, 2023)

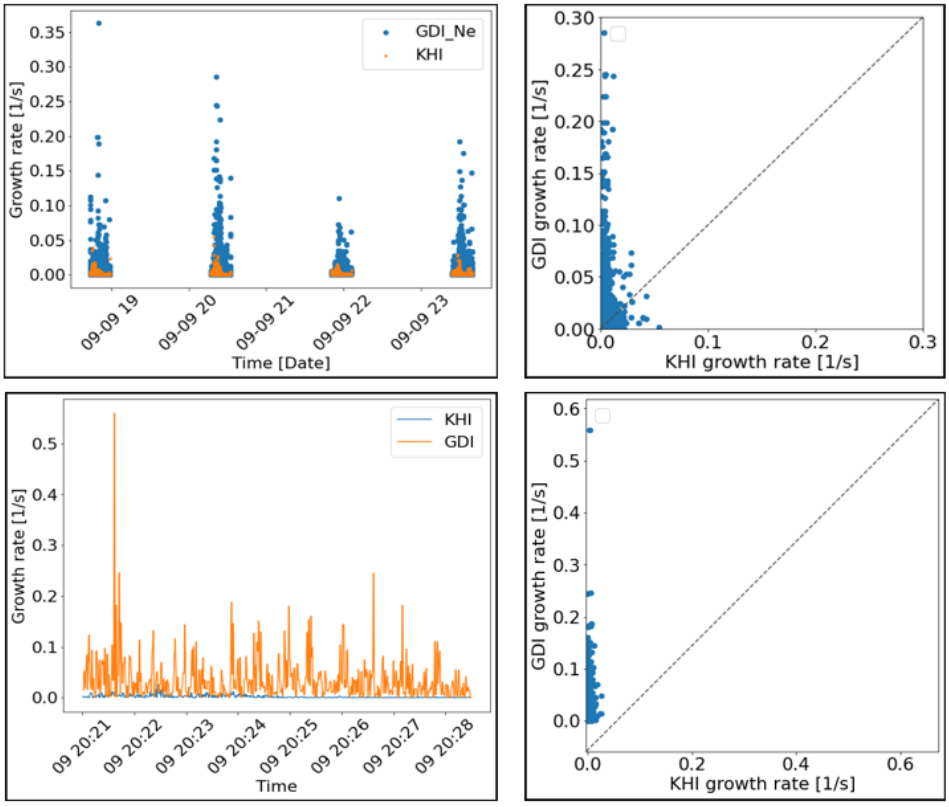

Figure 5.2: KHI and GDI growth rates plotted together and against each other 09.09.2015. In the top left plot the labels are showing the day 09-09, and the hour of the day, 19, 20, 21 etc.

In figure 5.2 we can see the KHI and GDI growth rates plotted together for the 09.09.2015 in the plots to the left, while on the right they are plotted against each other. This date was chosen as it was one of the dates used for my case by case analysis in the project paper, and it could be interesting to see the difference in the plot as the auroral region is included and there are made quite a bit of improvement of the code for this thesis. For the top left plot the GDI growth rate is plotted with blue dots, while the KHI growth rate is plotted with orange dots. In the bottom left plot the GDI growth rate is plotted with a orange line, while the KHI growth rate is plotted with a blue line. The reason for the different times in the top left and bottom left plots, are that the top plot is over the whole day of the 09.09.2015, while the bottom plot is only for 7 minutes of the day the 09.09.2015. We can see that the GDI growth rate is larger than the KHI growth rate for the whole period. The data is collected with a latitude of 60 to 90 degrees, and there was no restriction for Kp in the two top plots, while for the two bottom plots the latitude was 75 to 90 degrees, again with no restriction for Kp. The two bottom plots are from my project paper, while the two top plots are from the code that is used in this thesis. The motivation for showing these plots are so that I can compare the difference since I have included the aurora oval in the new plots. I also used a different dataset to calculate the GDI growth rate for the two top plots. We can see that the GDI growth rate is larger for all the plots also when the auroral region is included, however the two bottom plots reaches larger values of growth rate.

After showing the data for a selected day to compare with the project paper, we show the GDI and KHI growth rates for two different years, with restrictions for Kp, and with a latitude of 60 degrees. This to investigate the difference of the growth rates with different Kp values, as well as look at if the different years at different ends of the solar cycle has any impact.

Figure 5.3 shows three different plots with the growth rate on the vertical axis and the date on the horizontal axis. They are differentiated by Kp, so in the top plot, the data collected is when the Kp was between 1 and 3. The middle plot the Kp was between 3 and 6, and the bottom plot has a Kp between 6 and 9. All the data is collected in 2015 above 60 degrees of latitude. The blue dots represent the GDI linear growth rate, while the orange represents the KHI linear growth rate. As we can see there are more data points where the Kp is between 1 and 3, than when Kp is between 3 and 6 or 6 and 9. For the two top plots, we can see that the GDI growth rate is of approximately the same magnitude, with the same value as the maximum value. The bottom plot however does not reach the same maximum values for the GDI growth rate, but seem to have approximately the same values for the KHI growth rate. We can also see that there are seasonal variations for both the GDI and KHI growth rates, however

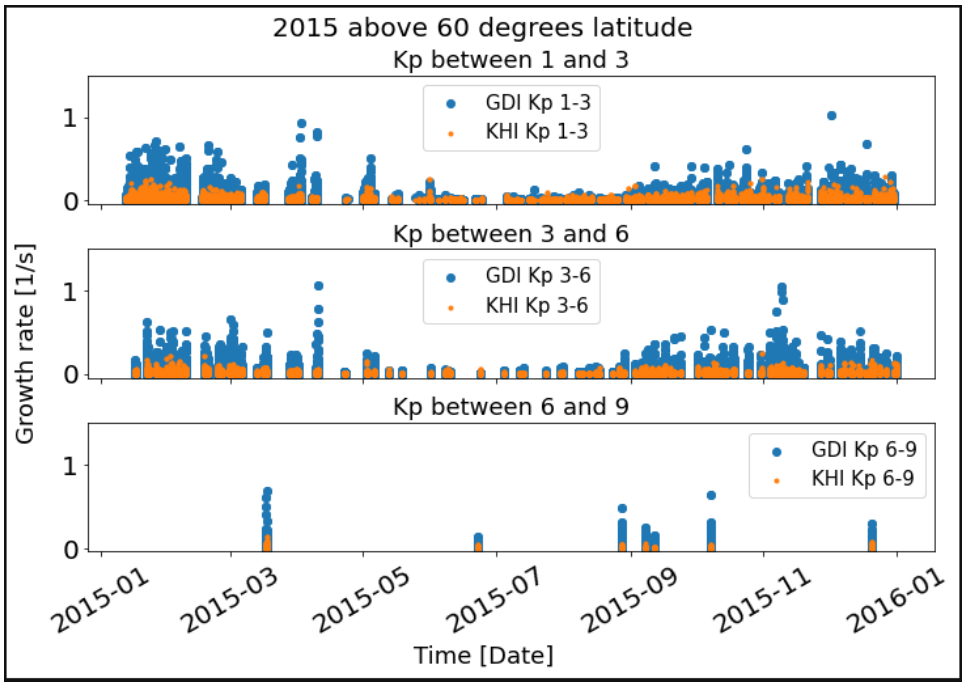

Figure 5.3: Comparison between the different Kp values for GDI and KHI growth rates during 2015 for latitudes above 60 degrees

they are more prominent for the GDI growth rate, and the growth rates are larger in the winter months of the year.

In figure 5.4 two different plots are shown, with the growth rate on the vertical axis and the date on the horizontal axis. This is shown so that it can be compared with figure 5.3, and see if the difference in years and time in the solar cycle has any impact. The data collected is from 2020 above 60 degrees latitude. It is differentiated in two plots by Kp between 1 and 3 in the top plot and Kp between 3 and 6 in the bottom plot. There is only two plots as there was no data with Kp between 6-9. Here we can see that the top plot has more data points than the bottom one, however the bottom plot has some GDI linear growth rates with a larger value. For the KHI linear growth rate it seems to be in the approximately same area of values in both plots. We can also see seasonal variations for both growth rates, however they are more prominent for the GDI growth rate, and the growth rates are larger in the winter months of the year.

After showing data from 2015 and 2020 with restrictions for Kp, we now show the GDI and KHI growth rates, with no restriction for Kp, but with the latitude divided up in two equal regions. The latitudes shown are between 60 and 75 degrees, and 75 and 90 degrees. This data will also be shown for 2015 and 2020.

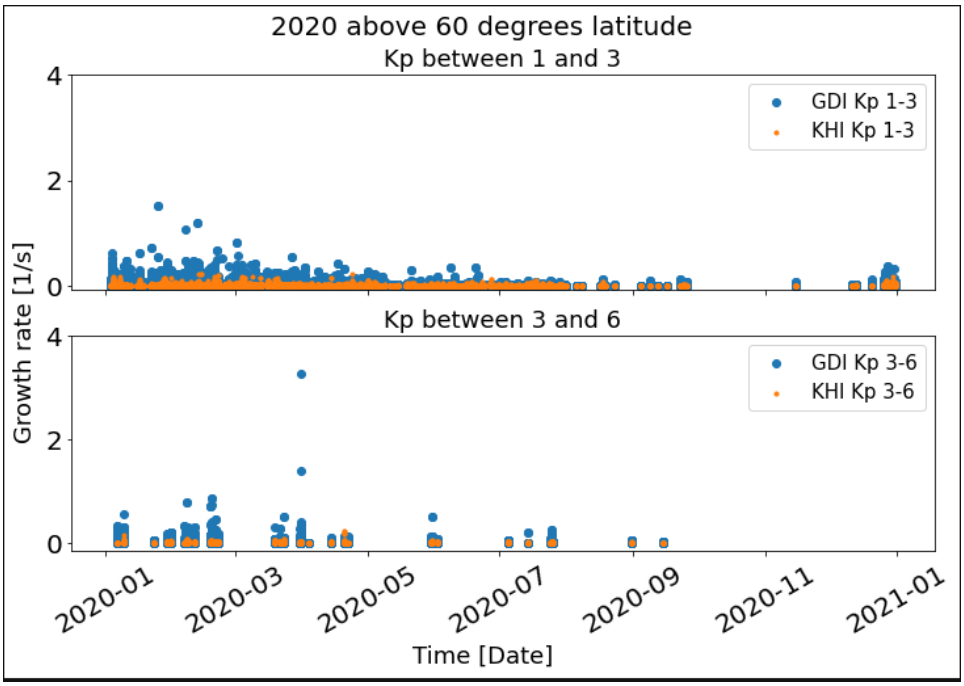

Figure 5.4: Comparison between the different Kp values for GDI and KHI growth rates during 2020 for latitudes above 60 degrees. There was no data for Kp 6-9, so there is only two plots for this figure.

From figure 5.5 the plot is divided by latitudes of 15 degrees each. The top plot has data collected between 60 and 75 degrees, while the bottom plot has data collected between 75 and 90 degrees. The auroral region is approximately from 60-75 degrees latitude, while the polar cap is approximately from 75-90 degrees latitude. So to divide these two regions up, we can compare if the regions impact the growth rates in any way. The data is collected in 2015, and for these plots the Kp was between 1 and 9. In the top plot, where the data is from the auroral region, we can see that some of the GDI growth rates are much larger than the rest of the plot, and much larger than the bottom plot. This contrast the values in figure 5.3 where we don't reach the same maximum values. However it seems like for the majority of the growth rates for both KHI and GDI that the values are similar for both plots. Again we can see that the growth rates have seasonal changes, and there are larger growth rates in the winter months.

From figure 5.6 we have two plots which are divided with 15 degrees of latitude. The top plot had data collected between 60 and 75 degrees latitude, the auroral region. The bottom plot has data collected between 75 and 90 degrees, the

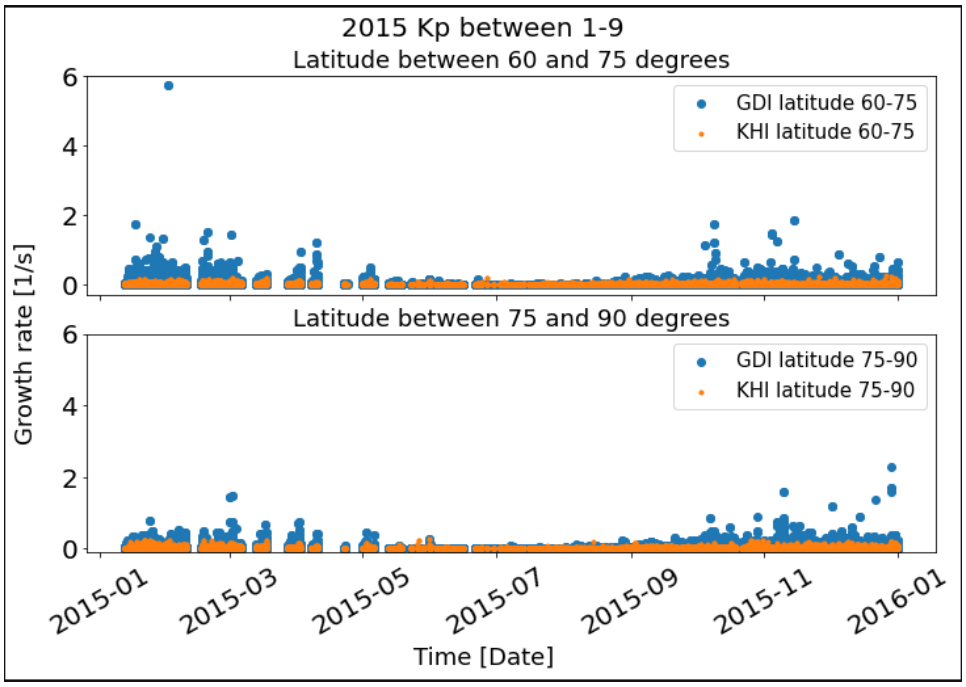

Figure 5.5: Comparison between latitudes for the GDI and KHI growth rates during 2015 for Kp between 1 and 9

polar cap. All the data is collected from 2020 with Kp between 1 and 9. The data from 2020 is shown to be able to compare with figure 5.5, and look at the difference for two years at different times in the solar cycle. For the top plot we can see that the GDI growth rates are larger for the start of the year than in the bottom plot, however the KHI growth rates are similar in both plots. Here we also has seasonal variations with larger growth rates in the winter months.

After showing data from 2015 and 2020 with restrictions for latitude, we now show the GDI and KHI growth rates in histograms. This is to be able to get a sense of the different quantities the different growth rates has. The histograms shown will be for both 2015 and 2020, with and without restrictions for Kp and latitude. This to be able to better compare the different quantities. All the histograms were normalized with the normalizing function that is built into Python, as an attempt to see if they would look any different. The end result was the same as the histograms shown here, and the normalized histograms are therefore not included.

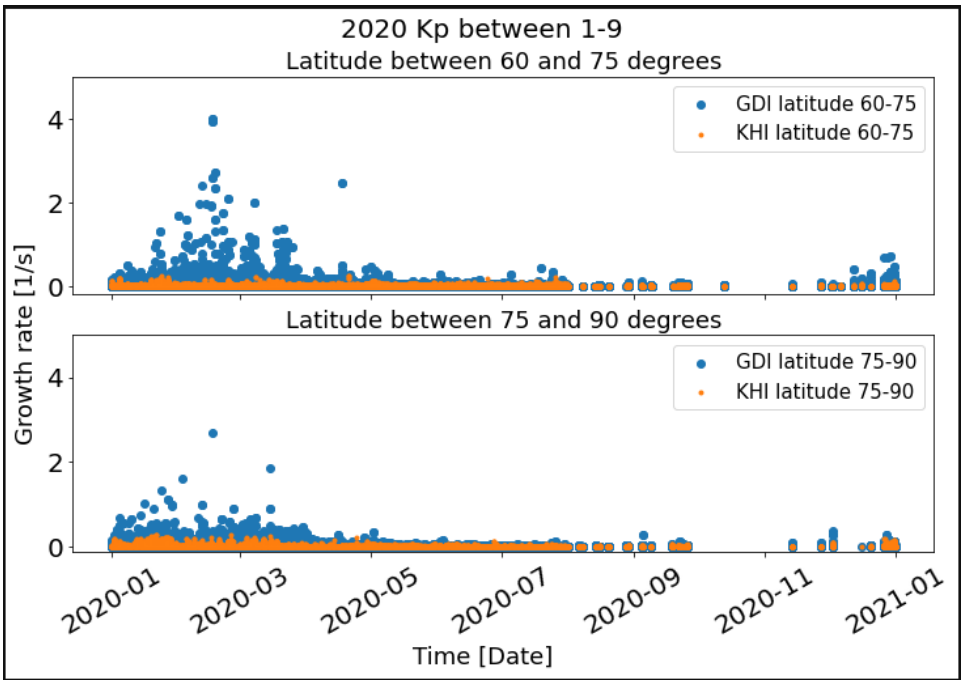

Figure 5.6: Comparison between latitudes for the GDI and KHI growth rates during 2020 for Kp between 1 and 9

In figure 5.7 there are six different histograms. The two on the top are for the GDI and KHI growth rate with Kp between 1 and 3. The two in the middle are for Kp between 3 and 6, and the two on the bottom are for Kp between 6 and 9. The data is from 2015 above 60 degrees latitude. The vertical axis gives the quantity of the growth rates in a logarithmic scale, and the horizontal axis gives the growth rates. I have also decided to make a range from 0.01/s to 1.0/s, as the values that are 0/s, were there are no growth rates, is not interesting to look at. We can see that the GDI growth rate for Kp 1 to 3 has the largest quantities, but the GDI growth rate for Kp 3 to 6 is close. The KHI growth rate is clearly smaller, and for the quantities of Kp 6 to 9 there is significantly less values.

In figure 5.8 the two top plots are for Kp between 1 and 3, while the two bottom plots are for Kp between 3 and 6. All the data is collected in 2020 above 60 degrees latitude. Here there are clearly larger GDI growth rate for Kp 1 to 3. The KHI growth rate for Kp 1 to 3 are much larger than the KHI growth rate for Kp 3 to 6.

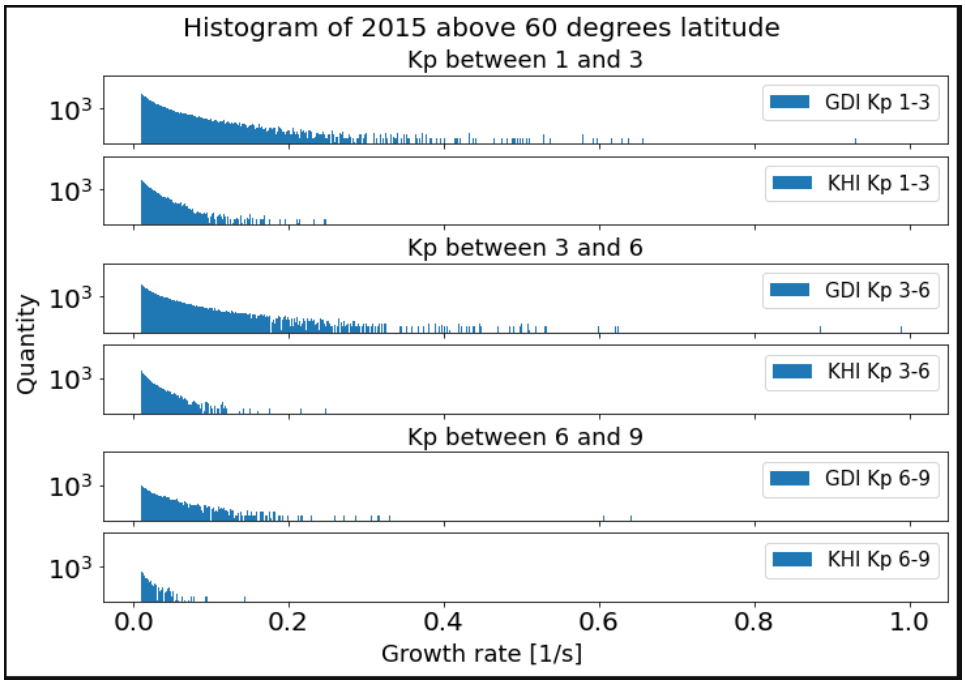

Figure 5.7: A histogram for the GDI and KHI growth rates for different Kp values in 2015

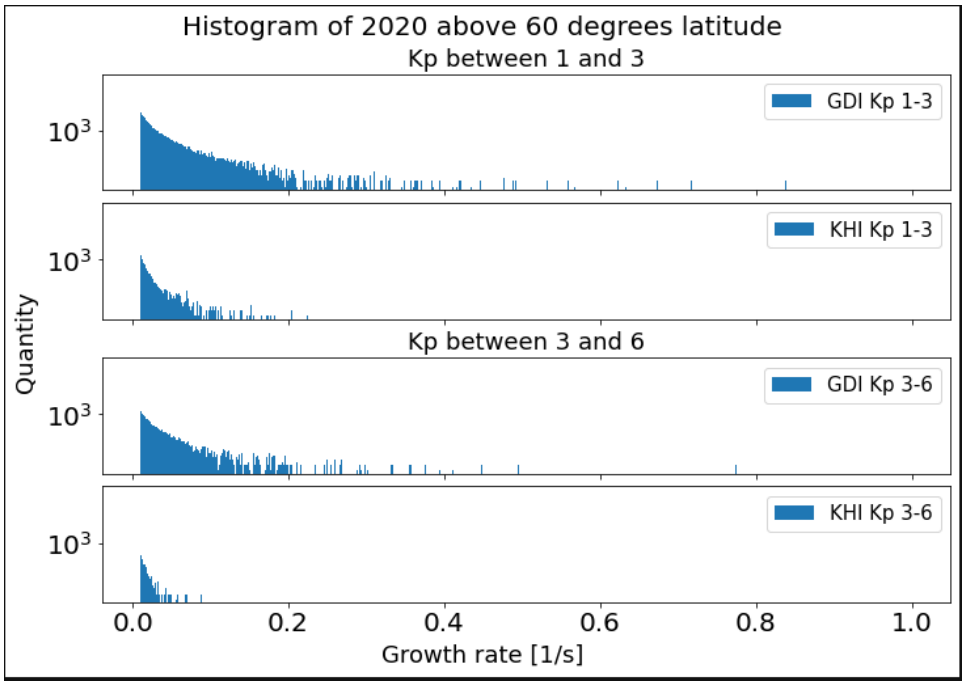

Figure 5.8: A histogram for the GDI and KHI growth rates for different Kp values in 2020. Here there was no data for Kp 6-9.

In figure 5.9 the two top plots have data from 60-75 degrees latitude, while the two bottom plots have data from 75-90 degrees latitude. All the data is collected in 2015 with Kp between 1 and 9. The quantities for the different GDI growth rates is very similar at the different latitudes, which looks to be the same for the quantities for the different KHI growth rates at the different latitudes.

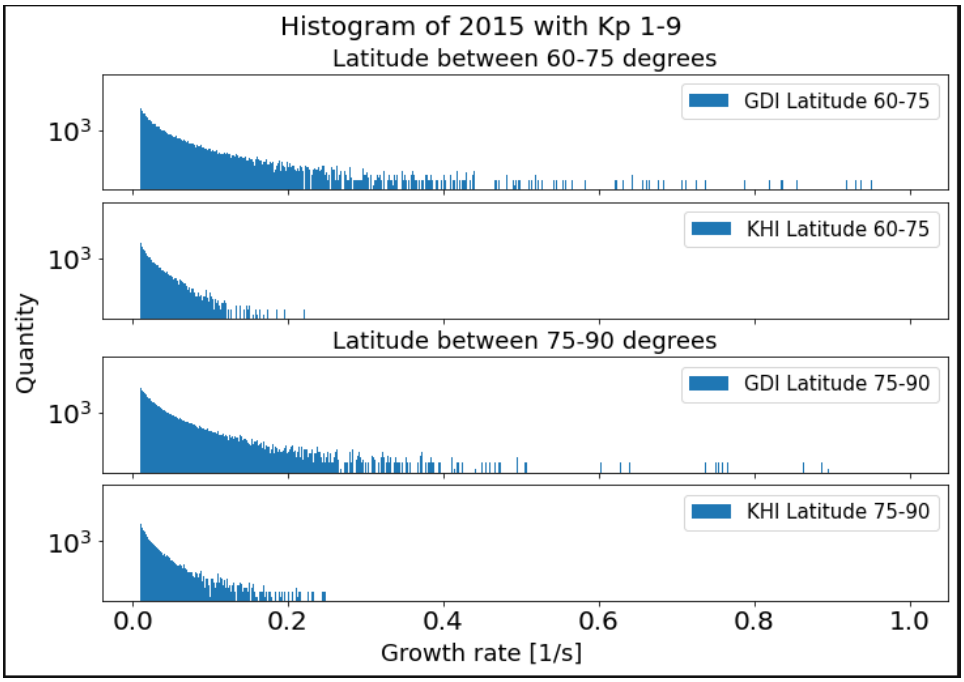

Figure 5.9: A histogram for the GDI and KHI growth rates for different latitudes in 2015

In figure 5.10 the two top plots have data from latitude 60-75 degrees, and the two bottom plots have data from latitude 75-90 degrees. All the data is collected from 2020 with Kp between 1 and 9. Here it seems like the GDI growth rate for latitude 75-90 degrees has larger quantities than the GDI growth rate for latitude 60-75 degrees. The KHI growth rate for latitude 75-90 degrees also has larger quantities than the KHI growth rate for latitude 60-75 degrees.

In figure 5.11 the two top plots have data from latitude 60-75 degrees, and the two bottom plots have data from latitude 75-90 degrees. This figure is to compare the GDI growth rate between solar maximum and solar minimum. All the data is collected with Kp between 1 and 9. As can be seen the GDI growth rate for 60 to 75 degrees in 2015 is larger than the GDI growth rate for the same degrees in 2020. The same is the case for the GDI growth rate with latitude 75 to 90 degrees.

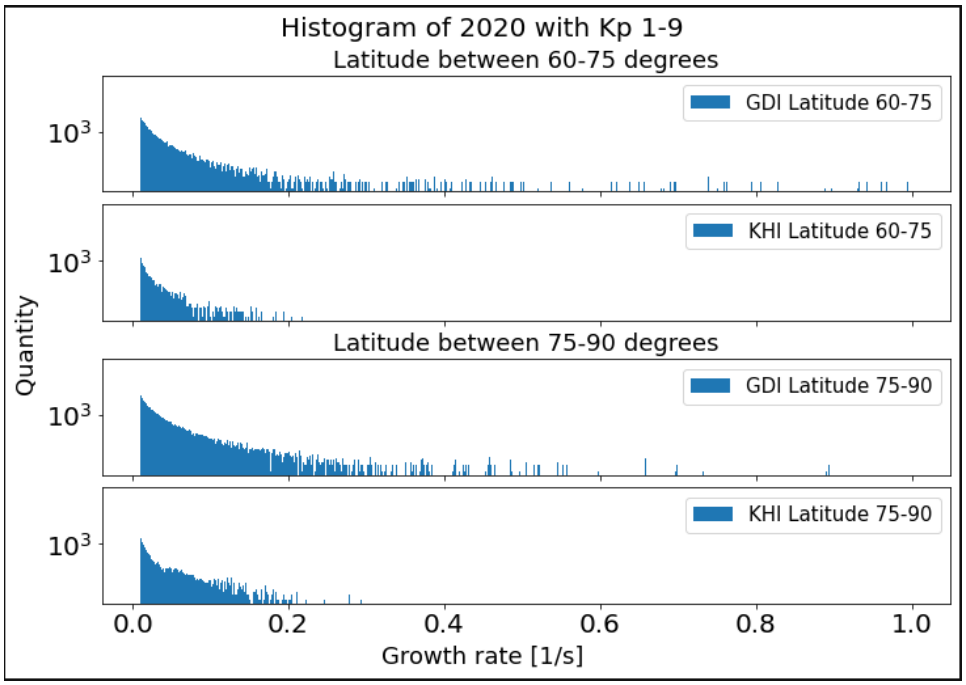

Figure 5.10: A histogram for the GDI and KHI growth rates for different latitudes in 2020

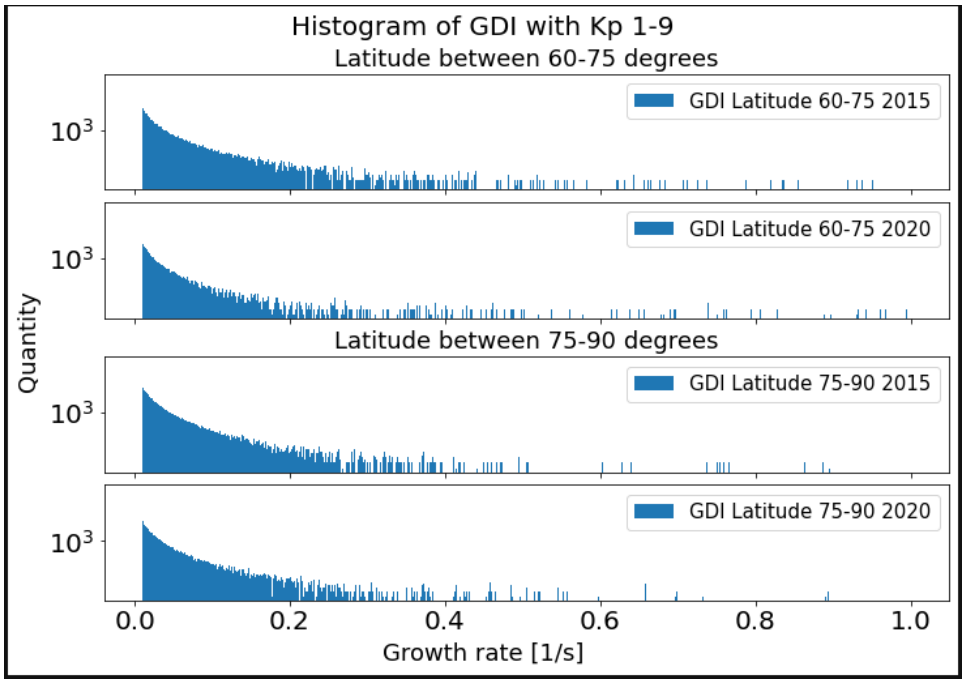

Figure 5.11: A histogram for the GDI growth rates for different latitudes for both 2015 and 2020

In figure 5.12 the two top plots have data from latitude 60-75 degrees, and the two bottom plots have data from latitude 75-90 degrees. This figure is to compare the KHI growth rate between solar maximum and solar minimum. All the data is collected with Kp between 1 and 9. Again can we see that the KHI growth rate from 2015 is larger than the KHI growth rate for 2020. This is for both intervals of latitude. It can also be seen that the KHI growth rate for 2020 is larger in the interval 75 to 90 degrees than from 60 to 75 degrees.

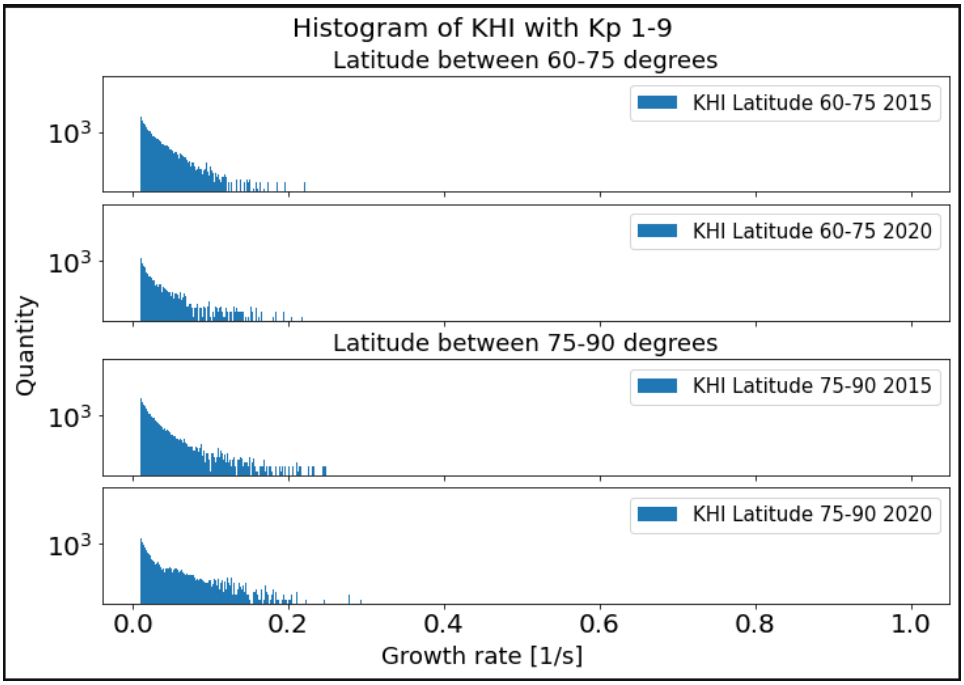

Figure 5.12: A histogram for the KHI growth rates for different latitudes for both 2015 and 2020

In figure 5.13 the two top plots have data with Kp from 1 to 3, and the two bottom plots have data with Kp from 3 to 6. All the data is collected for a latitude from 60 degrees. Here we can see that the GDI growth rate is larger in 2015 for both intervals of Kp. The GDI growth rate also has larger quantities for Kp 1 to 3 for both years. I have chosen to not use Kp 6 to 9 in this plot, as I only have data from 2015 the solar maximum, and would therefore be unable to compare it with a solar minimum.

In figure **??** the two top plots have data with Kp from 1 to 3, and the two bottom plots have data with Kp from 3 to 6. All the data is collected with a latitude from 60 degrees. The KHI growth rate is larger in 2015 for both intervals of Kp. The KHI growth rate also has larger quantities for Kp 1 to 3 for both years.

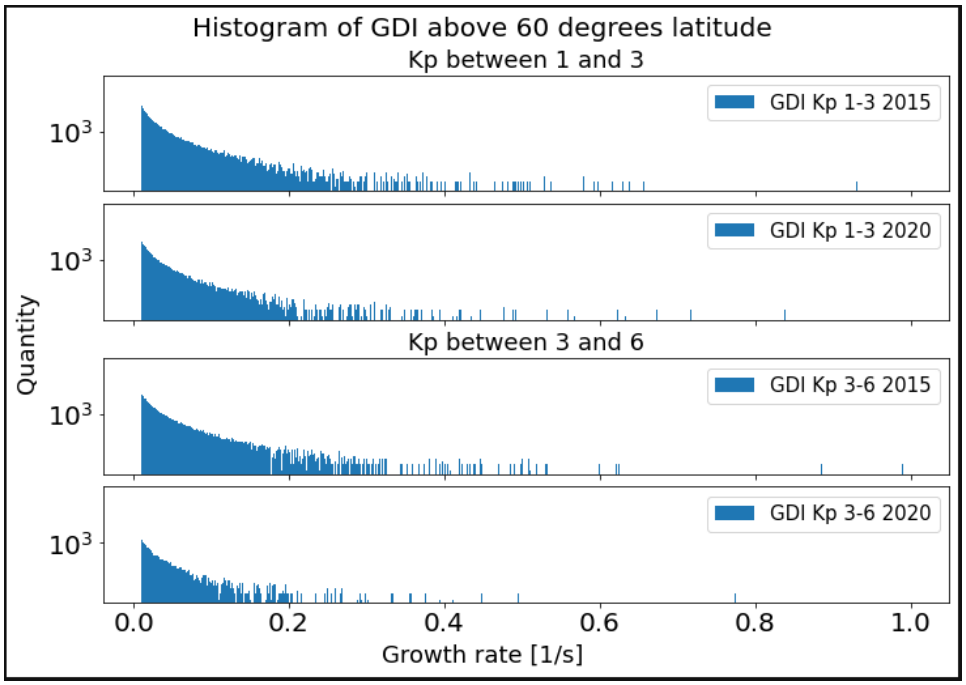

Figure 5.13: A histogram for the GDI growth rates for different Kp values for both 2015 and 2020. Since there are no data for Kp 6-9 in 2020, all the values for Kp 6-9 was omitted.

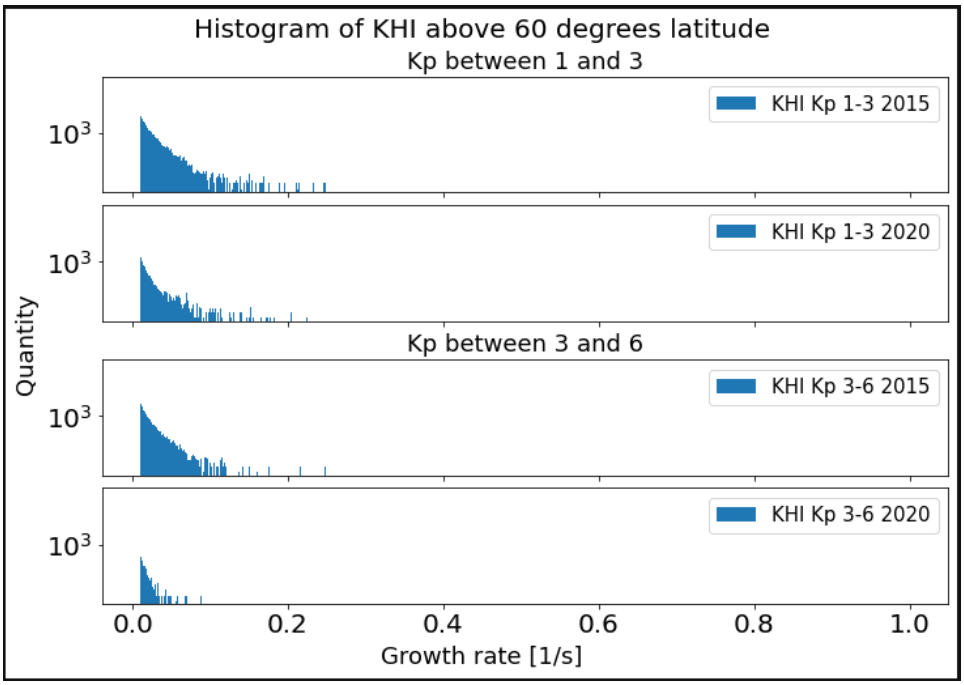

Figure 5.14: A histogram for the KHI growth rates for different Kp values for both 2015 and 2020. Since there is no data for Kp 6-9 in 2020, all values for Kp 6-9 was omitted.

6

Discussion

In this section I will discuss the results previously shown, and use the information from the instrumentation and background sections to try and put the results into context with other literature. The goal of the thesis is to look at the GDI and KHI linear growth rate to try and determine if the GDI or KHI is more dominant. I have looked at two different regions in the northern polar area, the auroral region from 60 to 75 degrees and the polar cap from 75 to 90 degrees. I have also looked at different geomagnetic activity, where I have used the Kp index, as well as looked at 2015 and 2020 to be able to compare solar maximum with solar minimum.

From figures 5.3, 5.4, 5.5 and 5.6 we can see that the GDI growth rate is larger than the KHI growth for both 2015 and 2020, which suggest that GDI is more common. For both the KHI and GDI growth rates there are seasonal variations. However the change in the GDI growth rate is larger than the change in the KHI growth rate which is rather consistent throughout the year. This indicates that in the ionosphere above 60 degrees latitude, GDI grows faster than KHI, and they are more active in the winter months. As an approximation we can look at a common value for the GDI growth rate which is 0.2/s which corresponds to a growth in 5 seconds. For KHI a common value is 0.1/s which corresponds to a growth in 10 seconds. Most of the values are still smaller than this for both the growth rates, which can be seen in the histograms. However the GDI also have some larger values as well. The KHI growth rates being smaller than the GDI growth rates corresponds well with Oksavik et al. (2012)

and Burston et al. (2016).
The reason for the seasonal variations is that in the summertime the whole polar cap has sunlight at all times. One of the criteria to have density gradients is to have areas with different density plasma. However in the summer the ionosphere in the northern hemisphere the density of ions and electrons are already so high, that it is much rarer to have plasma islands than in the winter, when the polar cap is turned away from the sun (Spicher et al., 2017). The solar EUV ionizes the ionospheric E-region. This can result in the F-region irregularities decay quickly, and the irregularity amplitude is lower (Jin et al., 2019). This seems to be consistent with earlier work from Oksavik et al. (2012), Jin et al. (2019), Spicher et al. (2017) and Burston et al. (2016).

We can see from figures 5.3, 5.4, 5.5, 5.6 and figures 5.7, 5.8, 5.9, 5.10, 5.11, 5.12, 5.13, 5.14, that there are more quantity of data, and that reaches higher values in 2015 than in 2020. The quantity is probably because the Swarm satellites "EFI" instruments was fully active in 2015, while they were used more sporadically in 2020, due to degradation (Kramer, 2023). However the higher values in GDI growth rate especially is not explained by this. As we can see from figure 5.1 in 2015 we are close to a solar maximum, while in 2020 we are at a solar minimum in the solar cycle. Perhaps a greater influx of solar particles to the ionosphere could be a reason for the larger values in GDI growth rate. From Oksavik et al. (2012) we know that there are generally more patches during the solar cycle maximum, both for winter and summer conditions (Oksavik et al., 2012). If I compare my results with those in Jin et al. (2019) it seems like they are consistent with their findings that there is a solar cycle dependency for the irregularities.

From figure 5.3 we can see that the largest value of growth rates are upwards to 1/s. This is for the GDI growth rate with Kp between 1 and 3, and Kp between 3 and 6. It looks like the GDI growth rate for Kp between 3 and 6 might have one or two values which are larger than the GDI growth rate in Kp between 1 and 3. As seen in figure 5.4 the largest GDI growth rate values happens for Kp between 3 and 6 where it is nearly up to 4 at the most. This fits with others finds that there are more patches with a Kp>4 (Weber et al., 1984).
From figures 5.5 and 5.6 we can see that the GDI growth rate has larger values for latitude between 60 and 75 degrees. This area of the ionosphere corresponds to where the aurora is generally located. There is typically a higher density of electrons and ions in this area (Carlson, 2012), and therefore you would expect the plasma patches to travel from the aurora oval and to the polar cap dayside, and therefore have a higher value of GDI and KHI growth rates farther north (Carlson, 2012). However the GDI only needs density

gradients to be created, and as there is higher density in the auroral region this could suggest a higher number of GDI. It may be that the large number of GDI growth rate that is shown in figures 5.5 and 5.6 are outliers that shouldn't be counted in. However in figure 5.11 we can see that the quantity of growth rate is approximately the same. In figure 5.12 we can see that the values for the KHI growth rates at latitude 75-90 degrees are larger than for latitude 60-75 degrees.

In this thesis I have used the equations for linear growth rate for both instabilities. There might however be uncertainties whit this method of deciding which instability is more dominant in the polar northern hemisphere. There is also an equation for growth rate in collisionless plasma that is not used (Oksavik et al., 2012), and looking at the different parameters separately might also give more accurate results. In figure 5.5 we can see a large value of GDI growth rate which does not match what we see in figure 5.3. To try and figure out the reason for the difference I have looked at if the amount of values match between the latitude and Kp results, and found that the Kp has a larger amount of datapoints. This was done for a few intervals of ten days each for both 2015 and 2020. This is as expected since there are some overlap in the way I have sectioned up the different Kp values. Then I tried to plot the GDI growth rate without a median filter. The results indicated that this might be the reason for the difference in growth rates between figure 5.3 and 5.5. There is also used a median filter on the electron density which is shown in figure 4.2, and this could also have an impact on the datapoints. What I also observed was that the general trend for the years and seasons were still the same.
The chosen scale length of 8000 meter gives us a resolution of 1 Hz. This will have an impact on the datapoints for KHI, but again will do nothing for the large scale trends seen in this thesis. The GDI growth rate was also consistently larger than the KHI growth rate.
I have used in situ measurements from the Swarm satellite A. These measurements and these calculations do not say anything about what is actually happening in the ionosphere, with respect to GDI and KHI growth rates. The calculations only tells us that if there are irregularities, they would with these conditions in the ionosphere have these growth rates for the GDI and KHI. From Tsunoda (1988) we know that the GDI is most active on trailing edges of the plasma patches. With my data it is not possible to differentiate between leading edges, and trailing edges. This means that the data shown in this thesis, says nothing of where in the polar region the measurements are taken. Since we don't know if there even are irregularities or plasma patches at the times the measurements are taken, there are uncertainties in how accurate the results of this thesis are.

/7 Conclusion

In this thesis I have tried to determine which of the main mechanisms for instability is more dominant in the ionosphere, above 60 degrees in latitude. I have looked at both the KHI and GDI growth rates for different latitudes and Kp values, to see if that has any impact on the instabilities. This is all looked at for 2015 and 2020, which is the solar cycle maximum and minimum respectively. It is clear that under my previously stated assumption that the GDI growth rate is larger than the KHI growth rate, and therefore more dominant. The GDI growth rate also appears more prone to seasonal change than the KHI growth rate is. I also noticed that during the solar maximum the values reached for the instabilities are larger than during solar minimum.
For further research a larger dataset could be used so it would be possible to see the whole change from 2015 to 2020. It could also be expanded to consider the southern hemisphere as well as the northern. The research could also include the equation for linear growth rate were collisions are taken into consideration, as well as look at the different parameters in the equations separately. As discussed it could also be an advantage to try and filter the data in a different way than I have done, as this clearly had an impact on the data. This does not mean the results are wrong, as the general trend stays the same no matter if there is a median filter or not.

Bibliography

Bahmer. Magnetosphere simple. 2024. URL https://commons.wikimedia.org/wiki/File:Magnetosphere_simple.jpg.

A. Brekke. *Physics of the Upper Polar Atmosphere*. Springer-Verlag Berlin and Heidelberg GmbH Co. K, 2012.

R. Burston, C. Mitchell, and I. Astin. Polar cap plasma patch primary linear instability growth rates compared. *Journal of Geophysical Research: Space Physics*, 121(4):3439–3451, 2016. doi: https://doi.org/10.1002/2015JA021895. URL https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1002/2015JA021895.

H. C. Carlson. Sharpening our thinking about polar cap ionospheric patch morphology, research, and mitigation techniques. *Radio Science*, 47(4), 2012. doi: https://doi.org/10.1029/2011RS004946. URL https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2011RS004946.

O. K. Carlson, H. C. and J. Moen. On a new process for cusp irregularity production. *Ann. Geophys.*, 26(2871–2885), 2008. doi: https://doi.org/10.5194/angeo-26-2871-2008. URL https://angeo.copernicus.org/articles/26/2871/2008/angeo-26-2871-2008.html.

ESA. About swarm, 2023a. URL https://earth.esa.int/eogateway/missions/swarm.

ESA. Efi overview, 2023b. URL https://earth.esa.int/eogateway/instruments/efi/description.

ESA. Vires for swarm, 2023c. URL https://vires.services.

R. A. Heelis. Studies of ionospheric plasma and electrodynamics and their application to ionosphere-magnetosphere coupling. *Reviews of Geophysics*, 26(2):317–328, 1988. doi: https://doi.org/10.1029/RG026i002p00317. URL https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/RG026i002p00317.

Y. Jin, A. Spicher, C. Xiong, L. B. N. Clausen, G. Kervalishvili, C. Stolle, and W. J. Miloch. Ionospheric plasma irregularities characterized by the swarm satellites: Statistics at high latitudes. *Journal of Geophysical Research: Space Physics*, 124(2):1262–1282, 2019. doi: https://doi.org/10.1029/2018JA026063. URL https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2018JA026063.

Kataoka, Ryuho and Nakano, Shin'ya. Auroral zone over the last 3000 years. *J. Space Weather Space Clim.*, 11:46, 2021. doi: 10.1051/swsc/2021030. URL https://doi.org/10.1051/swsc/2021030.

M. J. Keskinen, H. G. Mitchell, J. A. Fedder, P. Satyanarayana, S. T. Zalesak, and J. D. Huba. Nonlinear evolution of the kelvin-helmholtz instability in the high-latitude ionosphere. *Journal of Geophysical Research: Space Physics*, 93(A1): 137–152, 1988. doi: https://doi.org/10.1029/JA093iA01p00137. URL https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/JA093iA01p00137.

H. J. Kramer. Swarm (geomagnetic leo constellation), 2023. URL https://www.eoportal.org/satellite-missions/swarm#mission-status.

M. Lockwood, S. W. H. Cowley, and M. P. Freeman. The excitation of plasma convection in the high-latitude ionosphere. *Journal of Geophysical Research: Space Physics*, 95(A6):7961–7972, 1990. doi: https://doi.org/10.1029/JA095iA06p07961. URL https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/JA095iA06p07961.

J. Matzka, C. Stolle, Y. Yamazaki, O. Bronkalla, and A. Morschhauser. The geomagnetic kp index and derived indices of geomagnetic activity. *Space Weather*, 19(5):e2020SW002641, 2021. doi: https://doi.org/10.1029/2020SW002641. URL https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2020SW002641. e2020SW002641 2020SW002641.

P. T. Newell, R. A. Greenwald, and J. M. Ruohoniemi. The role of the ionosphere in aurora and space weather. *Reviews of Geophysics*, 39(2):137–149, 2001. doi: https://doi.org/10.1029/1999RG000077. URL https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/1999RG000077.

K. Oksavik, J. Moen, M. Lester, T. A. Bekkeng, and J. K. Bekkeng. In situ measurements of plasma irregularity growth in the cusp ionosphere. *Journal of Geophysical Research: Space Physics*, 117(A11), 2012. doi: https://doi.org/10.1029/2012JA017835. URL https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2012JA017835.

A. Piel. *Definition of the Plasma State*, pages 29–44. Springer International

Publishing, Cham, 2017. ISBN 978-3-319-63427-2. doi: 10.1007/978-3-319-63427-2_2. URL https://doi.org/10.1007/978-3-319-63427-2_2.

C. T. Russell, J. G. Luhmann, and R. J. Strangeway. *Space Physics An Introduction*. Cambridge University Press, 2016.

A. Spicher, T. Cameron, E. M. Grono, K. N. Yakymenko, S. C. Buchert, L. B. N. Clausen, D. J. Knudsen, K. A. McWilliams, and J. I. Moen. Observation of polar cap patches and calculation of gradient drift instability growth times: A swarm case study. *Geophysical Research Letters*, 42(2):201–206, 2015. doi: https://doi.org/10.1002/2014GL062590. URL https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1002/2014GL062590.

A. Spicher, L. B. N. Clausen, W. J. Miloch, V. Lofstad, Y. Jin, and J. I. Moen. Interhemispheric study of polar cap patch occurrence based on swarm in situ data. *Journal of Geophysical Research: Space Physics*, 122(3):3837–3851, 2017. doi: https://doi.org/10.1002/2016JA023750. URL https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1002/2016JA023750.

SunPy. Plotting a solar cycle index, 2023. URL https://docs.sunpy.org/en/stable/generated/gallery/plotting/solar_cycle_example.html.

S. F. D. Team. Mission analysis status, 2023. URL https://earth.esa.int/eogateway/documents/20142/37627/Orbit-Evolution-of-the-Swarm-Mission.pdf/0d858e4c-a774-5a2e-dfc1-32373e1aefbd.

R. T. Tsunoda. High-latitude f region irregularities: A review and synthesis. *Reviews of Geophysics*, 26(4):719–760, 1988. doi: https://doi.org/10.1029/RG026i004p00719. URL https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/RG026i004p00719.

E. J. Weber, J. Buchau, J. G. Moore, J. R. Sharber, R. C. Livingston, J. D. Winningham, and B. W. Reinisch. F layer ionization patches in the polar cap. *Journal of Geophysical Research: Space Physics*, 89(A3):1683–1694, 1984. doi: https://doi.org/10.1029/JA089iA03p01683. URL https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/JA089iA03p01683.

8

A

Main Code

[ ]:
```
SERVER_URL = 'https://vires.services/ows'
```

[ ]:
```
import matplotlib as mpl
import matplotlib.pyplot as plt
import matplotlib.lines as lines
import cartopy.crs as ccrs
from tempfile import TemporaryFile
import sys

import numpy as np
import xarray as xr
import datetime as dt
from matplotlib.dates import DateFormatter
import scipy.signal as sc
import pandas as pd
import time, datetime
from datetime import datetime
# Control the HTML display of the datasets
xr.set_options(display_expand_attrs=False,␣
 ↪display_expand_coords=True, display_expand_data=True)

from viresclient import SwarmRequest
import warnings; warnings.simplefilter('ignore')
```

[ ]:
```
request = SwarmRequest(SERVER_URL)
```

[ ]:
```
request.available_collections("EFI_TCT02", details=False)
```

[ ]:
```
request.available_collections("EFI_IDM", details=False)
```

[ ]:
```
request.available_collections("EFI", details=False)
```

[ ]:
```
tct_vars = [
    # Satellite velocity in NEC frame
    "VsatC", "VsatE", "VsatN",
    # Ion drifts along-track from vertical (..v) and␣
 ↪horizontal (..h) TII sensor
    "Vixv", "Vixh",
    # Ion drifts cross-track (y from horizontal sensor, z␣
 ↪from vertical sensor)
    #  (in satellite-track coordinates)
    "Viy", "Viz",
    # Random error estimates for the above
    #  (Negative value indicates no estimate available)
    "Viy_error", "Viz_error",
]
```

[ ]:
```
idm_vars = [
    # Satellite velocity in NEC frame
    "V_sat_nec",
    # Along-track ion drift velocity, uncertainty, validity␣
 ↪flags, and velocity without detrending
    "V_i", "V_i_err", "V_i_Flags", "V_i_raw",
    # Ion density, uncertainty, and validity flags
    "N_i", "N_i_err", "N_i_Flags",
]
```

[ ]:
```
start = "2015-06-28T21:00:00"
end = "2015-06-28T22:00:00"
L_start = 60
L_end = 90
K_start = 1
K_end = 9
```

[ ]:
```
request = SwarmRequest(SERVER_URL)
request.set_collection("SW_EXPT_EFIA_TCT02")
request.set_products(measurements=tct_vars,
                     auxiliaries=["Kp", "SyncStatus"])
request.set_range_filter("Latitude", L_start, L_end,␣
 ↪negate=False)
request.set_range_filter("Kp", K_start, K_end, negate=False)
request.set_range_filter("SyncStatus", negate=False)
data = request.get_between(start, end, nrecords_limit=None)
```

[ ]:
```
df1 = data.as_dataframe()
df1.head()
```

[ ]:
```
ds1 = data.as_xarray()

ds1
```

[ ]:
```
request = SwarmRequest(SERVER_URL)
request.set_collection("SW_PREL_EFICIDM_2_")
request.set_products(measurements=idm_vars,
    auxiliaries=["Kp"])


request.set_range_filter("Latitude", L_start, L_end,␣
 ↪negate=False)
request.set_range_filter("Kp", K_start, K_end, negate=False)
data = request.get_between(start, end, nrecords_limit=None)
```

[ ]:
```
df2 = data.as_dataframe()
df2.head()
```

[ ]:
```
ds2 = data.as_xarray()
ds2
```

[ ]:
```
request = SwarmRequest(SERVER_URL)
request.set_collection("SW_OPER_EFIA_LP_1B")
request.set_products(measurements=['Ne'], auxiliaries=["Kp"])
request.set_range_filter("Latitude", L_start, L_end,␣
 ↪negate=False)
request.set_range_filter("Kp", K_start, K_end, negate=False)
data = request.get_between(start, end, nrecords_limit=None)
```

[ ]:
```
df3 = data.as_dataframe()
df3.head()
```

[ ]:
```
ds3 = data.as_xarray()
ds3
```

[ ]:
```
d1 = np.ndarray(len(ds1.Timestamp.values))

out = pd.to_datetime(ds1.Timestamp.values)
#print(out[0:2])
d = out.to_pydatetime()
for i in range(len(d1)):
    d1[i] = d[i].timestamp()
#print(d[0:2])
#print(d1[0:2])
d2 = np.ndarray(len(ds2.Timestamp.values))

out2 = pd.to_datetime(ds2.Timestamp.values)
#print(type(out.to_pydatetime()))
d_2 = out2.to_pydatetime()
for i in range(len(d2)):
    d2[i] = d_2[i].timestamp()
#print(d_2[0:2])
#print(d2[0:2])

d3 = np.ndarray(len(ds3.Timestamp.values))

out3 = pd.to_datetime(ds3.Timestamp.values)
#print(type(out.to_pydatetime()))
d_3 = out3.to_pydatetime()
for i in range(len(d3)):
    d3[i] = d_3[i].timestamp()
```

[ ]:
```
if len(d1) > len(d2):
    #print("hei")
    N = np.empty((len(d2)))
    #Vi = np.empty((len(d3)))
    Vr = np.empty((len(d2)))
    V = np.empty((len(d2)))
    Vy = np.empty((len(d2)))
    tid = np.empty((len(d2)))
    Kp_1 = np.empty((len(d2)))
    #Kp_2 = np.empty((len(d3)))
    Ne = np.empty((len(d2)))
    Vixh = np.empty((len(d2)))
    Vixv = np.empty((len(d2)))
    N[:] = np.nan
    #Vi[:] = np.nan
    Vr[:] = np.nan
    V[:] = np.nan
    Vy[:] = np.nan
    tid[:] = np.nan
    Kp_1[:] = np.nan
    #Kp_2[:] = np.nan
    Ne[:] = np.nan
    Vixh[:] = np.nan
    Vixv[:] = np.nan
    #print(np.shape(ds2.V_sat_nec))
```

[ ]:
```
    for i in range(0, len(d2)):
        #print(i)
        num2 = d2[i]
        ind = np.argmin(np.abs(num2-d1))
        #print(ind)
        num1 = d1[ind]
        #print(abs(num2 - num1))
        #print(num1, num2)
        if abs(num2 - num1) > 0.03:

            #print(abs(num2 - num1))
            tid[i] = d2[i]

        elif abs(num2 - num1) <= 0.03:

            tid[i] = d2[i]
            N[i] = ds2.N_i[i]
            #Vi[i] = ds2.V_i[i]
            Vy[i] = ds1.Viy[i]
            Vr[i] = ds1.VsatE[i]  #E og N
            V[i] = ds1.VsatN[i]
            Kp_1[i] = ds1.Kp[i]
            #Kp_2[i] = ds2.Kp[i]
            #Ne[i] = ds3.Ne[i]
            Vixh[i] = ds1.Vixh[i]
            Vixv[i] = ds1.Vixv[i]
```

[ ]:
```
        #print(num1, num2)
        #print(Vi)
elif len(d2) > len(d1):
    #print("lol")
    N = np.empty((len(d1)))
    Vi = np.empty((len(d1)))
    Vr = np.empty((len(d1)))
    V = np.empty((len(d1)))
    Vy = np.empty((len(d1)))
    tid = np.empty((len(d1)))
    Kp_1 = np.empty((len(d1)))
    Kp_2 = np.empty((len(d1)))
    Ne = np.empty((len(d1)))
    Vixh = np.empty((len(d1)))
    Vixv = np.empty((len(d1)))
    N[:] = np.nan
    Vi[:] = np.nan
    Vr[:] = np.nan
    V[:] = np.nan
    Vy[:] = np.nan
    tid[:] = np.nan
    Kp_1[:] = np.nan
    Kp_2[:] = np.nan
    Ne[:] = np.nan
    Vixh[:] = np.nan
    Vixv[:] = np.nan

    for i in range(0, len(d1)):
        #print(i)
        num1 = d1[i]
        ind = np.argmin(np.abs(num1-d2))
        #print(ind)
        num2 = d2[ind]
```

[ ]:
```
        if abs(num1 - num2) > 0.03:

            tid[i] = d1[i]

        elif abs(num1 - num2) <= 0.03:

            tid[i] = d1[i]
            N[i] = ds2.N_i[ind]
            Vi[i] = ds2.V_i[ind]
            Vy[i] = ds1.Viy[i]
            Vr[i] = ds2.V_sat_nec[ind, 0]
            V[i] = ds2.V_sat_nec[ind, 1]
            Kp_1[i] = ds1.Kp[i]
            Kp_2[i] = ds2.Kp[ind]
            Ne[i] = ds3.Ne[i]
            Vixh[i] = ds1.Vixh[i]
            Vixv[i] = ds1.Vixv[i]
            #print(Vi)
            #print()
            #print(Vi)
            #print(Vy)'
```

[ ]:
```
"""elif len(d3) > len(d1):
    #print("lol")
    #N = np.empty((len(d1)))
    #Vi = np.empty((len(d1)))
    Vr = np.empty((len(d1)))
    V = np.empty((len(d1)))
    Vy = np.empty((len(d1)))
    tid = np.empty((len(d1)))
    Kp_1 = np.empty((len(d1)))
    #Kp_2 = np.empty((len(d1)))
    Ne = np.empty((len(d1)))
    Vixh = np.empty((len(d1)))
    Vixv = np.empty((len(d1)))
    #N[:] = np.nan
    #Vi[:] = np.nan
    Vr[:] = np.nan
    V[:] = np.nan
    Vy[:] = np.nan
    tid[:] = np.nan
    Kp_1[:] = np.nan
    #Kp_2[:] = np.nan
    Ne[:] = np.nan
    Vixh[:] = np.nan
    Vixv[:] = np.nan
```

[ ]:
```
    for i in range(0, len(d1)):
        #print(i)
        num1 = d1[i]
        ind = np.argmin(np.abs(num1-d3))
        #print(ind)
        num2 = d3[ind]

        if abs(num1 - num2) > 0.03:

            tid[i] = d1[i]

        elif abs(num1 - num2) <= 0.03:

            tid[i] = d1[i]
            #N[i] = ds2.N_i[i]
            #Vi[i] = ds2.V_i[i]
            Vy[i] = ds1.Viy[i]
            Vr[i] = ds1.VsatE[i]
            V[i] = ds1.VsatN[i]
            Kp_1[i] = ds1.Kp[i]
            #Kp_2[i] = ds2.Kp[i]
            Ne[i] = ds3.Ne[i]
            Vixh[i] = ds1.Vixh[i]
            Vixv[i] = ds1.Vixv[i]
            #print(Vi)
            #print()
            #print(Vi)
            #print(Vy)'"""
```

[ ]:
```
# Sample DataFrame with float column representing Unix␣
↪timestamps
data = {'timestamp': tid}
time = pd.DataFrame(data)

# Convert float to datetime using Pandas' to_datetime()␣
↪function
time['timestamp'] = pd.to_datetime(time['timestamp'],␣
↪unit='s')

# Print the updated DataFrame
#print(time)

#plt.scatter(tid, Vi)
#plt.show()
```

[ ]:
```
#Vi_1 = [np.nan if S >= 10000 or S <= -10000 else S for S in␣
↪Vi]
Vy_1 = [np.nan if T >= 10000 or T <= -10000 else T for T in␣
↪Vy]
Vr_1 = [np.nan if U >= 10000 or U <= -10000 else U for U in␣
↪Vr]
V_1 = [np.nan if X >= 10000 or X <= -10000 else X for X in V]
N_1 = [np.nan if Y >= 1e8 or Y <= 0 else Y for Y in N]
Vixh_1 = [np.nan if H >= 10000 or H <= -10000 else H for H␣
↪in Vixh]
Vixv_1 = [np.nan if L >= 10000 or L <= -10000 else L for L␣
↪in Vixv]
```

[ ]:
```
#Vi_2 = np.array(Vi_1)
Vy_2 = np.array(Vy_1)
Vr_2 = np.array(Vr_1)
V_2 = np.array(V_1)
N_2 = np.array(N_1)
N_3 = N_2*1e6
Vixh_2 = np.array(Vixh_1)
Vixv_2 = np.array(Vixv_1)

#plt.scatter(tid, Vi_2)
#plt.show()

#print(Vi_2)
```

[ ]:
```
#plt.plot(tid, N_3)
#plt.ylim(0, 1e6)
#plt.show()
```

[ ]:
```
delta_t = 1
V_r = Vr_2
V = V_2
V_perp = np.sqrt(V_r**2+V**2)
#V_exb = Vi_2
#delta_x = delta_t * abs(V_exb - V_perp)
```

[ ]:
```
N_i0 = N_3
N_i3 = sc.medfilt(N_3, kernel_size=None) # To remove spikes
N_i5 = sc.medfilt(N_3, kernel_size=5)

"""delta_n1s = np.empty((len(N_3)))
N_background = np.empty((len(N_3)))

delta_n1s[:] = np.nan
N_background[:] = np.nan

for t in range(len(N_3)-2):
    delta_n1s[t] = (N_i[t + 2] - N_i[t])
    N_background[t] = min(N_i[t: t + 2])"""
```

[ ]:
```
#N_01s = np.percentile(V_exb, 30)
#N_background = min(N_i) # background density, using the 30␣
 ↪percentile of all the density values
plt.rcParams.update({'font.size': 15})
plt.figure(figsize=(10, 7))
#print(N_i[155:165])
plt.scatter(time, N_i0, label='N_i 0 point gradient')
plt.scatter(time, N_i3, label='N_i 3 point gradient')
```

[ ]:
```
plt.xticks(fontsize=20, rotation = -45)
plt.yticks(fontsize=20)
plt.xlabel('Time', fontsize=20)
plt.ylabel('Ion density [m^-3]', fontsize=20)
plt.legend()
plt.show()
```

[ ]:
```
#Growth_GD = (V_exb * delta_n1s)/(N_background * delta_x)
#plt.scatter(time, Growth_GD)
#plt.show()
"""print(Growth_GD)
print(V_exb[158])
print(delta_n1s[158])
print(N_background[158])
print(delta_x[158])
np.where(Growth_GD < -50)"""
```

[ ]:
```
"""fig, (ax1, ax2, ax3, ax4, ax5) = plt.subplots(5,␣
 ↪sharex=True, figsize=(10, 7))
ax1.scatter(time, delta_n1s, label="delta_n1s")
ax2.scatter(time, V_exb, label="Velocity")
ax3.scatter(time, N_background, label="N_background")
ax4.scatter(time, delta_x, label='delta x')
ax5.scatter(time, Growth_GD, label='Growth_GD')
ax1.legend()
ax2.legend()
ax3.legend()
ax4.legend()
ax5.legend()
plt.rc('font', size=22)
plt.show()"""
```

[ ]:
```
L_1 = 8000
```

[ ]:
```
V_iy = Vy_2

delta_v1s = np.empty(len(Vy_2))
delta_v1s[:] = np.nan

for t in range(len(Vy_2)-2):
    delta_v1s[t] = (V_iy[t + 2] - V_iy[t])
```

[ ]:
```
Growth_KHy = 0.2*(delta_v1s/L_1)
```

[ ]:
```
N_e0 = Ne
N_e3 = sc.medfilt(Ne, kernel_size=3) # To remove spikes
N_e5 = sc.medfilt(Ne, kernel_size=5)

"""delta_ne1s = np.empty((len(Ne)))
Ne_background = np.empty((len(Ne)))

delta_ne1s[:] = np.nan
Ne_background[:] = np.nan

for t in range(len(Ne)-2):
    delta_ne1s[t] = (N_e[t + 2] - N_e[t])
    Ne_background[t] = min(N_e[t: t + 2])"""
```

[ ]:
```
plt.scatter(time, N_e0, label="N_e 0 point gradient")
plt.scatter(time, N_e3, label="N_e 3 point gradient")
plt.scatter(time, N_e5, label="N_e 5 point gradient")
plt.xticks(rotation=45)
plt.xlabel('Time')
plt.ylabel('Electron density [m^-3]')
plt.show()
```

[ ]:
```
V_plasma = np.sqrt(Vixv_2**2 + Vixh_2**2)

N_rel = delta_ne1s/Ne_background

GD = (V_plasma * delta_ne1s)/(Ne_background * delta_t *␣
↪V_perp)
```

[ ]:
```
#Vi_1 = [np.nan if S >= 10000 or S <= -10000 else S for S in␣
↪Vi]
#x = np.argwhere(np.isnan(Growth_GD))
#y = np.argwhere(np.isnan(Growth_KHy))

#print(x)
#print(y)

#Growth_GD1 = [np.nan if G >= 1 or G <= -1 else G for G in␣
↪Growth_GD]
#Growth_GD2 = np.array(Growth_GD1)
```

[ ]:
```
for i in range(len(Growth_KHy)):
    if np.isnan(Growth_KHy[i]):
        #Growth_GD[i] = np.nan
        GD[i] = np.nan
    #if np.isnan(Growth_GD[i]):
        #Growth_KHy[i] = np.nan
        #GD[i] = np.nan
    if np.isnan(GD[i]):
        #Growth_GD[i] = np.nan
        Growth_KHy[i] = np.nan

#print(np.argwhere(np.isnan(Growth_KHy)))
#print(np.argwhere(np.isnan(Growth_GD2)))
#print(np.argwhere(np.isnan(GD)))
```

[ ]:
```
"""fig, (ax1, ax2, ax3) = plt.subplots(3, sharex=True,␣
↪figsize=(10, 7))
ax1.scatter(time, V_iy, label="Velocity_y_TCT")
ax2.scatter(time, V_exb, label="Velocity")
ax3.scatter(time, N_i, label="Ion_density")
ax1.legend()
ax2.legend()
ax3.legend()
plt.xticks(rotation=45)
plt.rc('font', size=22)
plt.show()"""
```

[ ]:
```
"""fig, axes = plt.subplots(nrows=3, sharex=True,␣
↪figsize=(10, 7))
ds1.plot.scatter(x="Timestamp", y="Viy", ax=axes[0],␣
↪label="Velocity_y_TCT")
ds2.plot.scatter(x="Timestamp", y="V_i", ax=axes[1],␣
↪label="Velocity")
ds2.plot.scatter(x="Timestamp", y="N_i", ax=axes[2],␣
↪label="Ion_density")
axes[0].legend()
axes[1].legend()
axes[2].legend()
plt.xticks(rotation=45)
plt.rc('font', size=22)
plt.show()"""
```

[ ]:
```
#plt.figure(figsize=(8,5))

#Growth_GDI = sc.medfilt(Growth_GD, kernel_size=None)
Growth_KHIy = sc.medfilt(Growth_KHy, kernel_size=None)
GDI = sc.medfilt(GD, kernel_size=None)
```

[ ]:
```
fig = plt.figure(figsize=(10,7))
plt.scatter(time, abs(GDI), label="GDI_Ne", marker='o')
plt.scatter(time, abs(Growth_KHIy), label="KHI", marker='.')
#plt.scatter(time, abs(Growth_GDI), label="GDI_Ni", marker='.
↪')

plt.xticks(rotation=45)
plt.ylabel("Growth rate [1/s]")
plt.xlabel("Time [Date]")
#plt.ylim(0, 0.5)
```

[ ]:
```
plt.rc('font', size=22)
plt.legend()
plt.show()
```

[ ]:
```
#Growth_KHIy_1 = Growth_KHy[~pd.isnull(Growth_KHy)]
#GDI_1 = GD[~pd.isnull(GD)]
#print(len(Growth_KHIy))
#print(len(GDI))

"""GDI_1[0:len(GD_2)]

if GD_1 > GD_2:
    GD_1_short = GD_1[0:len(GD_2)]
else:
    GD_1_short = GD_1"""

fig, ax = plt.subplots(figsize=(10,7))
#print(((polyline)))
```

[ ]:
```
diag_line, = ax.plot(ax.get_xlim(), ax.get_ylim(), ls="--",␣
 ↪c=".3")
def on_change(axes):
    # When this function is called it checks the current
    # values of xlim and ylim and modifies diag_line
    # accordingly.
    x_lims = ax.get_xlim()
    y_lims = ax.get_ylim()
    diag_line.set_data(x_lims, y_lims)

# Connect two callbacks to your axis instance.
# These will call the function "on_change" whenever
# xlim or ylim is changed.
ax.callbacks.connect('xlim_changed', on_change)
ax.callbacks.connect('ylim_changed', on_change)
```

[ ]:
```
plt.axis('square')
plt.xlim([0,0.3])
plt.ylim([0,0.3])
plt.ylabel("GDI growth rate [1/s]")
plt.xlabel("KHI growth rate [1/s]")
plt.rc('font', size=22)
plt.scatter(abs(Growth_KHIy), abs(GDI))
plt.legend()
plt.show()
#Legge inn best fit curve
```

[ ]:
```
"""geod = ccrs.Geodetic()
proj = ccrs.LambertAzimuthalEqualArea(central_latitude=90.0)

plt.figure(figsize=(16,9))

ease_extent = [-9000000., 9000000., -9000000., 9000000.]
ax = plt.axes(projection=proj)
ax.set_extent(ease_extent, crs=proj)
ax.gridlines(color='gray', linestyle='--')
ax.coastlines()

plt.tight_layout()
```

[ ]:
```
start_lon, start_lat = ds2.Longitude[0], ds2.Latitude[0]
stop_lon, stop_lat = ds2.Longitude[-1], ds2.Latitude[-1]

plt.plot([start_lon, stop_lon], [start_lat, stop_lat],
         color='blue', linewidth=2, marker='o',
         transform=ccrs.Geodetic(),
         )

plt.show()"""
```

[ ]:
```
"""np.savetxt('Jan_2015_GDI_1_3', GDI)
np.savetxt('Jan_2015_KHI_1_3', Growth_KHIy)
np.savetxt('Jan_2015_Kp_1_1_3', Kp_1)
#np.savetxt('Jnn_2015_Kp_2_75', Kp_2)
np.savetxt('Jan_2015_time_1_3', time)
#np.savetxt('Jan_2015_N_75', N_3)
#np.savetxt('Jan_2015_Vi_75', Vi_2)
np.savetxt('Jan_2015_Vr_1_3', Vr_2)
np.savetxt('Jan_2015_V_1_3', V_2)
np.savetxt('Jan_2015_Vy_1_3', Vy_2)
np.savetxt('Jan_2015_deltaV_1_3', delta_v1s)
np.savetxt('Jan_2015_N_rel_1_3', N_rel)"""
```

[ ]:
```
"""fig, (ax1, ax2, ax3, ax4) = plt.subplots(4, sharex=True,␣
↪figsize=(10, 7))
ax1.scatter(time, Kp_2, label="Kp_index_2")
ax2.scatter(time, abs(Growth_GDI), label="GDI")
ax3.scatter(time, Kp_1, label="Kp_index_1")
ax4.scatter(time, abs(Growth_KHIy), label="KHI")
ax1.legend()
ax2.legend()
ax3.legend()
ax4.legend()
plt.rc('font', size=22)
plt.xticks(rotation = -45)
plt.show()"""
```

One example on how I plotted the data saved from the main code:

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

[ ]:
```
G_Jan75 = open('Jan_2020_GDI_75', 'r')
G_Jan1 = G_Jan75.readlines()
G1_Jan = [float(i) for i in G_Jan1]
GDI_Jan75 = np.asarray(G1_Jan)

K_Jan75 = open('Jan_2020_KHI_75', 'r')
K_Jan1 = K_Jan75.readlines()
K1_Jan = [float(i) for i in K_Jan1]
KHI_Jan75 = np.asarray(K1_Jan)
```

[ ]:
```
T_Jan75 = open('Jan_2020_time_75', 'r')
T_Jan1 = T_Jan75.readlines()
T1_Jan = [float(i) for i in T_Jan1]
Time_Jan75 = np.asarray(T1_Jan)

# Sample DataFrame with float column representing Unix␣
 ↪timestamps
data_jan75 = {'timestamp': Time_Jan75}
time_jan75 = pd.DataFrame(data_jan75)

# Convert float to datetime using Pandas' to_datetime()␣
 ↪function
time_jan75['timestamp'] = pd.
 ↪to_datetime(time_jan75['timestamp'], unit='ns')
```

[ ]:
```

G_Feb75 = open('Feb_2020_GDI_75', 'r')
G_Feb1 = G_Feb75.readlines()
G1_Feb = [float(i) for i in G_Feb1]
GDI_Feb75 = np.asarray(G1_Feb)

K_Feb75 = open('Feb_2020_KHI_75', 'r')
K_Feb1 = K_Feb75.readlines()
K1_Feb = [float(i) for i in K_Feb1]
KHI_Feb75 = np.asarray(K1_Feb)
```

[ ]:
```
T_Feb75 = open('Feb_2020_time_75', 'r')
T_Feb1 = T_Feb75.readlines()
T1_Feb = [float(i) for i in T_Feb1]
Time_Feb75 = np.asarray(T1_Feb)

# Sample DataFrame with float column representing Unix␣
 ↪timestamps
data_feb75 = {'timestamp': Time_Feb75}
time_feb75 = pd.DataFrame(data_feb75)

# Convert float to datetime using Pandas' to_datetime()␣
 ↪function
time_feb75['timestamp'] = pd.
 ↪to_datetime(time_feb75['timestamp'], unit='ns')
```

[ ]:
```
G_Mar75 = open('Mar_2020_GDI_75', 'r')
G_Mar1 = G_Mar75.readlines()
G1_Mar = [float(i) for i in G_Mar1]
GDI_Mar75 = np.asarray(G1_Mar)

K_Mar75 = open('Mar_2020_KHI_75', 'r')
K_Mar1 = K_Mar75.readlines()
K1_Mar = [float(i) for i in K_Mar1]
KHI_Mar75 = np.asarray(K1_Mar)
```

[ ]:
```
T_Mar75 = open('Mar_2020_time_75', 'r')
T_Mar1 = T_Mar75.readlines()
T1_Mar = [float(i) for i in T_Mar1]
Time_Mar75 = np.asarray(T1_Mar)

# Sample DataFrame with float column representing Unix␣
 ↪timestamps
data_mar75 = {'timestamp': Time_Mar75}
time_mar75 = pd.DataFrame(data_mar75)

# Convert float to datetime using Pandas' to_datetime()␣
 ↪function
time_mar75['timestamp'] = pd.
 ↪to_datetime(time_mar75['timestamp'], unit='ns')
```

[ ]:
```


G_Apr75 = open('Apr_2020_GDI_75', 'r')
G_Apr1 = G_Apr75.readlines()
G1_Apr = [float(i) for i in G_Apr1]
GDI_Apr75 = np.asarray(G1_Apr)

K_Apr75 = open('Apr_2020_KHI_75', 'r')
K_Apr1 = K_Apr75.readlines()
K1_Apr = [float(i) for i in K_Apr1]
KHI_Apr75 = np.asarray(K1_Apr)

T_Apr75 = open('Apr_2020_time_75', 'r')
T_Apr1 = T_Apr75.readlines()
T1_Apr = [float(i) for i in T_Apr1]
Time_Apr75 = np.asarray(T1_Apr)
```

[ ]:
```
# Sample DataFrame with float column representing Unix␣
 ↪timestamps
data_apr75 = {'timestamp': Time_Apr75}
time_apr75 = pd.DataFrame(data_apr75)

# Convert float to datetime using Pandas' to_datetime()␣
 ↪function
time_apr75['timestamp'] = pd.
 ↪to_datetime(time_apr75['timestamp'], unit='ns')



G_Mai75 = open('Mai_2020_GDI_75', 'r')
G_Mai1 = G_Mai75.readlines()
G1_Mai = [float(i) for i in G_Mai1]
GDI_Mai75 = np.asarray(G1_Mai)
```

[ ]:
```
K_Mai75 = open('Mai_2020_KHI_75', 'r')
K_Mai1 = K_Mai75.readlines()
K1_Mai = [float(i) for i in K_Mai1]
KHI_Mai75 = np.asarray(K1_Mai)

T_Mai75 = open('Mai_2020_time_75', 'r')
T_Mai1 = T_Mai75.readlines()
T1_Mai = [float(i) for i in T_Mai1]
Time_Mai75 = np.asarray(T1_Mai)

# Sample DataFrame with float column representing Unix␣
 ↪timestamps
data_mai75 = {'timestamp': Time_Mai75}
time_mai75 = pd.DataFrame(data_mai75)
```

[ ]:
```
# Convert float to datetime using Pandas' to_datetime()␣
 ↪function
time_mai75['timestamp'] = pd.
 ↪to_datetime(time_mai75['timestamp'], unit='ns')



G_Jun75 = open('Jun_2020_GDI_75', 'r')
G_Jun1 = G_Jun75.readlines()
G1_Jun = [float(i) for i in G_Jun1]
GDI_Jun75 = np.asarray(G1_Jun)

K_Jun75 = open('Jun_2020_KHI_75', 'r')
K_Jun1 = K_Jun75.readlines()
K1_Jun = [float(i) for i in K_Jun1]
KHI_Jun75 = np.asarray(K1_Jun)
```

[ ]:
```
T_Jun75 = open('Jun_2020_time_75', 'r')
T_Jun1 = T_Jun75.readlines()
T1_Jun = [float(i) for i in T_Jun1]
Time_Jun75 = np.asarray(T1_Jun)

# Sample DataFrame with float column representing Unix␣
 ↪timestamps
data_jun75 = {'timestamp': Time_Jun75}
time_jun75 = pd.DataFrame(data_jun75)

# Convert float to datetime using Pandas' to_datetime()␣
 ↪function
time_jun75['timestamp'] = pd.
 ↪to_datetime(time_jun75['timestamp'], unit='ns')
```

[ ]:
```
G_Jul75 = open('Jul_2020_GDI_75', 'r')
G_Jul1 = G_Jul75.readlines()
G1_Jul = [float(i) for i in G_Jul1]
GDI_Jul75 = np.asarray(G1_Jul)

K_Jul75 = open('Jul_2020_KHI_75', 'r')
K_Jul1 = K_Jul75.readlines()
K1_Jul = [float(i) for i in K_Jul1]
KHI_Jul75 = np.asarray(K1_Jul)

T_Jul75 = open('Jul_2020_time_75', 'r')
T_Jul1 = T_Jul75.readlines()
T1_Jul = [float(i) for i in T_Jul1]
Time_Jul75 = np.asarray(T1_Jul)
```

[ ]:
```
# Sample DataFrame with float column representing Unix␣
 ↪timestamps
data_jul75 = {'timestamp': Time_Jul75}
time_jul75 = pd.DataFrame(data_jul75)

# Convert float to datetime using Pandas' to_datetime()␣
 ↪function
time_jul75['timestamp'] = pd.
 ↪to_datetime(time_jul75['timestamp'], unit='ns')
```

[ ]:
```
G_Aug75 = open('Aug_2020_GDI_75', 'r')
G_Aug1 = G_Aug75.readlines()
G1_Aug = [float(i) for i in G_Aug1]
GDI_Aug75 = np.asarray(G1_Aug)

K_Aug75 = open('Aug_2020_KHI_75', 'r')
K_Aug1 = K_Aug75.readlines()
K1_Aug = [float(i) for i in K_Aug1]
KHI_Aug75 = np.asarray(K1_Aug)

T_Aug75 = open('Aug_2020_time_75', 'r')
T_Aug1 = T_Aug75.readlines()
T1_Aug = [float(i) for i in T_Aug1]
Time_Aug75 = np.asarray(T1_Aug)
```

[ ]:
```
# Sample DataFrame with float column representing Unix␣
 ↪timestamps
data_aug75 = {'timestamp': Time_Aug75}
time_aug75 = pd.DataFrame(data_aug75)

# Convert float to datetime using Pandas' to_datetime()␣
 ↪function
time_aug75['timestamp'] = pd.
 ↪to_datetime(time_aug75['timestamp'], unit='ns')



G_Sep75 = open('Sep_2020_GDI_75', 'r')
G_Sep1 = G_Sep75.readlines()
G1_Sep = [float(i) for i in G_Sep1]
GDI_Sep75 = np.asarray(G1_Sep)

K_Sep75 = open('Sep_2020_KHI_75', 'r')
K_Sep1 = K_Sep75.readlines()
K1_Sep = [float(i) for i in K_Sep1]
KHI_Sep75 = np.asarray(K1_Sep)
```

[ ]:
```
T_Sep75 = open('Sep_2020_time_75', 'r')
T_Sep1 = T_Sep75.readlines()
T1_Sep = [float(i) for i in T_Sep1]
Time_Sep75 = np.asarray(T1_Sep)

# Sample DataFrame with float column representing Unix␣
 ↪timestamps
data_sep75 = {'timestamp': Time_Sep75}
time_sep75 = pd.DataFrame(data_sep75)

# Convert float to datetime using Pandas' to_datetime()␣
 ↪function
time_sep75['timestamp'] = pd.
 ↪to_datetime(time_sep75['timestamp'], unit='ns')
```

[ ]:
```
G_Okt75 = open('Okt_2020_GDI_75', 'r')
G_Okt1 = G_Okt75.readlines()
G1_Okt = [float(i) for i in G_Okt1]
GDI_Okt75 = np.asarray(G1_Okt)

K_Okt75 = open('Okt_2020_KHI_75', 'r')
K_Okt1 = K_Okt75.readlines()
K1_Okt = [float(i) for i in K_Okt1]
KHI_Okt75 = np.asarray(K1_Okt)

T_Okt75 = open('Okt_2020_time_75', 'r')
T_Okt1 = T_Okt75.readlines()
T1_Okt = [float(i) for i in T_Okt1]
Time_Okt75 = np.asarray(T1_Okt)
```

[ ]:
```
# Sample DataFrame with float column representing Unix␣
 ↪timestamps
data_okt75 = {'timestamp': Time_Okt75}
time_okt75 = pd.DataFrame(data_okt75)

# Convert float to datetime using Pandas' to_datetime()␣
 ↪function
time_okt75['timestamp'] = pd.
 ↪to_datetime(time_okt75['timestamp'], unit='ns')



G_Nov75 = open('Nov_2020_GDI_75', 'r')
G_Nov1 = G_Nov75.readlines()
G1_Nov = [float(i) for i in G_Nov1]
GDI_Nov75 = np.asarray(G1_Nov)
```

[ ]:
```
K_Nov75 = open('Nov_2020_KHI_75', 'r')
K_Nov1 = K_Nov75.readlines()
K1_Nov = [float(i) for i in K_Nov1]
KHI_Nov75 = np.asarray(K1_Nov)

T_Nov75 = open('Nov_2020_time_75', 'r')
T_Nov1 = T_Nov75.readlines()
T1_Nov = [float(i) for i in T_Nov1]
Time_Nov75 = np.asarray(T1_Nov)
```

[ ]:
```
# Sample DataFrame with float column representing Unix␣
↪timestamps
data_nov75 = {'timestamp': Time_Nov75}
time_nov75 = pd.DataFrame(data_nov75)

# Convert float to datetime using Pandas' to_datetime()␣
↪function
time_nov75['timestamp'] = pd.
↪to_datetime(time_nov75['timestamp'], unit='ns')
```

[ ]:
```
G_Des75 = open('Des_2020_GDI_75', 'r')
G_Des1 = G_Des75.readlines()
G1_Des = [float(i) for i in G_Des1]
GDI_Des75 = np.asarray(G1_Des)

K_Des75 = open('Des_2020_KHI_75', 'r')
K_Des1 = K_Des75.readlines()
K1_Des = [float(i) for i in K_Des1]
KHI_Des75 = np.asarray(K1_Des)
```

[ ]:
```
T_Des75 = open('Des_2020_time_75', 'r')
T_Des1 = T_Des75.readlines()
T1_Des = [float(i) for i in T_Des1]
Time_Des75 = np.asarray(T1_Des)

# Sample DataFrame with float column representing Unix␣
↪timestamps
data_des75 = {'timestamp': Time_Des75}
time_des75 = pd.DataFrame(data_des75)

# Convert float to datetime using Pandas' to_datetime()␣
↪function
time_des75['timestamp'] = pd.
↪to_datetime(time_des75['timestamp'], unit='ns')
```

[ ]:
```
GDI_75_20 = np.concatenate([abs(GDI_Jan75), abs(GDI_Feb75),␣
↪abs(GDI_Mar75), abs(GDI_Apr75), abs(GDI_Mai75),
                            abs(GDI_Jun75), abs(GDI_Jul75),␣
↪abs(GDI_Aug75), abs(GDI_Sep75), abs(GDI_Okt75),
                            abs(GDI_Nov75), abs(GDI_Des75)])
```

[ ]:
```
KHI_75_20 = np.concatenate([abs(KHI_Jan75), abs(KHI_Feb75),␣
↪abs(KHI_Mar75), abs(KHI_Apr75), abs(KHI_Mai75),
                        abs(KHI_Jun75), abs(KHI_Jul75),␣
↪abs(KHI_Aug75), abs(KHI_Sep75), abs(KHI_Okt75),
                        abs(KHI_Nov75), abs(KHI_Des75)])

time_75_20 = np.concatenate([time_jan75, time_feb75,␣
↪time_mar75, time_apr75, time_mai75, time_jun75,␣
↪time_jul75,
                         time_aug75, time_sep75,␣
↪time_okt75, time_nov75, time_des75])
```

[ ]:
```
G_Jan90 = open('Jan_2020_GDI_90', 'r')
G_Jan2 = G_Jan90.readlines()
G2_Jan = [float(i) for i in G_Jan2]
GDI_Jan90 = np.asarray(G2_Jan)

K_Jan90 = open('Jan_2020_KHI_90', 'r')
K_Jan2 = K_Jan90.readlines()
K2_Jan = [float(i) for i in K_Jan2]
KHI_Jan90 = np.asarray(K2_Jan)
```

[ ]:
```
T_Jan90 = open('Jan_2020_time_90', 'r')
T_Jan2 = T_Jan90.readlines()
T2_Jan = [float(i) for i in T_Jan2]
Time_Jan90 = np.asarray(T2_Jan)

# Sample DataFrame with float column representing Unix␣
↪timestamps
data_jan90 = {'timestamp': Time_Jan90}
time_jan90 = pd.DataFrame(data_jan90)

# Convert float to datetime using Pandas' to_datetime()␣
↪function
time_jan90['timestamp'] = pd.
↪to_datetime(time_jan90['timestamp'], unit='ns')
```

[ ]:
```
G_Feb90 = open('Feb_2020_GDI_90', 'r')
G_Feb2 = G_Feb90.readlines()
G2_Feb = [float(i) for i in G_Feb2]
GDI_Feb90 = np.asarray(G2_Feb)

K_Feb90 = open('Feb_2020_KHI_90', 'r')
K_Feb2 = K_Feb90.readlines()
K2_Feb = [float(i) for i in K_Feb2]
KHI_Feb90 = np.asarray(K2_Feb)
```

[ ]:
```
T_Feb90 = open('Feb_2020_time_90', 'r')
T_Feb2 = T_Feb90.readlines()
T2_Feb = [float(i) for i in T_Feb2]
Time_Feb90 = np.asarray(T2_Feb)

# Sample DataFrame with float column representing Unix␣
 ↪timestamps
data_feb90 = {'timestamp': Time_Feb90}
time_feb90 = pd.DataFrame(data_feb90)

# Convert float to datetime using Pandas' to_datetime()␣
 ↪function
time_feb90['timestamp'] = pd.
 ↪to_datetime(time_feb90['timestamp'], unit='ns')
```

[ ]:
```
G_Mar90 = open('Mar_2020_GDI_90', 'r')
G_Mar2 = G_Mar90.readlines()
G2_Mar = [float(i) for i in G_Mar2]
GDI_Mar90 = np.asarray(G2_Mar)

K_Mar90 = open('Mar_2020_KHI_90', 'r')
K_Mar2 = K_Mar90.readlines()
K2_Mar = [float(i) for i in K_Mar2]
KHI_Mar90 = np.asarray(K2_Mar)
```

[ ]:
```
T_Mar90 = open('Mar_2020_time_90', 'r')
T_Mar2 = T_Mar90.readlines()
T2_Mar = [float(i) for i in T_Mar2]
Time_Mar90 = np.asarray(T2_Mar)

# Sample DataFrame with float column representing Unix␣
 ↪timestamps
data_mar90 = {'timestamp': Time_Mar90}
time_mar90 = pd.DataFrame(data_mar90)

# Convert float to datetime using Pandas' to_datetime()␣
 ↪function
time_mar90['timestamp'] = pd.
 ↪to_datetime(time_mar90['timestamp'], unit='ns')
```

[ ]:
```


G_Apr90 = open('Apr_2020_GDI_90', 'r')
G_Apr2 = G_Apr90.readlines()
G2_Apr = [float(i) for i in G_Apr2]
GDI_Apr90 = np.asarray(G2_Apr)

K_Apr90 = open('Apr_2020_KHI_90', 'r')
K_Apr2 = K_Apr90.readlines()
K2_Apr = [float(i) for i in K_Apr2]
KHI_Apr90 = np.asarray(K2_Apr)
```

[ ]:
```
T_Apr90 = open('Apr_2020_time_90', 'r')
T_Apr2 = T_Apr90.readlines()
T2_Apr = [float(i) for i in T_Apr2]
Time_Apr90 = np.asarray(T2_Apr)

# Sample DataFrame with float column representing Unix␣
 ↪timestamps
data_apr90 = {'timestamp': Time_Apr90}
time_apr90 = pd.DataFrame(data_apr90)

# Convert float to datetime using Pandas' to_datetime()␣
 ↪function
time_apr90['timestamp'] = pd.
 ↪to_datetime(time_apr90['timestamp'], unit='ns')
```

[ ]:
```
G_Mai90 = open('Mai_2020_GDI_90', 'r')
G_Mai2 = G_Mai90.readlines()
G2_Mai = [float(i) for i in G_Mai2]
GDI_Mai90 = np.asarray(G2_Mai)

K_Mai90 = open('Mai_2020_KHI_90', 'r')
K_Mai2 = K_Mai90.readlines()
K2_Mai = [float(i) for i in K_Mai2]
KHI_Mai90 = np.asarray(K2_Mai)

T_Mai90 = open('Mai_2020_time_90', 'r')
T_Mai2 = T_Mai90.readlines()
T2_Mai = [float(i) for i in T_Mai2]
Time_Mai90 = np.asarray(T2_Mai)
```

[ ]:
```
# Sample DataFrame with float column representing Unix␣
 ↪timestamps
data_mai90 = {'timestamp': Time_Mai90}
time_mai90 = pd.DataFrame(data_mai90)

# Convert float to datetime using Pandas' to_datetime()␣
 ↪function
time_mai90['timestamp'] = pd.
 ↪to_datetime(time_mai90['timestamp'], unit='ns')



G_Jun90 = open('Jun_2020_GDI_90', 'r')
G_Jun2 = G_Jun90.readlines()
G2_Jun = [float(i) for i in G_Jun2]
GDI_Jun90 = np.asarray(G2_Jun)

K_Jun90 = open('Jun_2020_KHI_90', 'r')
K_Jun2 = K_Jun90.readlines()
K2_Jun = [float(i) for i in K_Jun2]
KHI_Jun90 = np.asarray(K2_Jun)
```

[ ]:
```
T_Jun90 = open('Jun_2020_time_90', 'r')
T_Jun2 = T_Jun90.readlines()
T2_Jun = [float(i) for i in T_Jun2]
Time_Jun90 = np.asarray(T2_Jun)

# Sample DataFrame with float column representing Unix␣
 ↪timestamps
data_jun90 = {'timestamp': Time_Jun90}
time_jun90 = pd.DataFrame(data_jun90)

# Convert float to datetime using Pandas' to_datetime()␣
 ↪function
time_jun90['timestamp'] = pd.
 ↪to_datetime(time_jun90['timestamp'], unit='ns')
```

[ ]:
```
G_Jul90 = open('Jul_2020_GDI_90', 'r')
G_Jul2 = G_Jul90.readlines()
G2_Jul = [float(i) for i in G_Jul2]
GDI_Jul90 = np.asarray(G2_Jul)

K_Jul90 = open('Jul_2020_KHI_90', 'r')
K_Jul2 = K_Jul90.readlines()
K2_Jul = [float(i) for i in K_Jul2]
KHI_Jul90 = np.asarray(K2_Jul)

T_Jul90 = open('Jul_2020_time_90', 'r')
T_Jul2 = T_Jul90.readlines()
T2_Jul = [float(i) for i in T_Jul2]
Time_Jul90 = np.asarray(T2_Jul)

# Sample DataFrame with float column representing Unix␣
 ↪timestamps
data_jul90 = {'timestamp': Time_Jul90}
time_jul90 = pd.DataFrame(data_jul90)
```

[ ]:
```
# Convert float to datetime using Pandas' to_datetime()␣
 ↪function
time_jul90['timestamp'] = pd.
 ↪to_datetime(time_jul90['timestamp'], unit='ns')



G_Aug90 = open('Aug_2020_GDI_90', 'r')
G_Aug2 = G_Aug90.readlines()
G2_Aug = [float(i) for i in G_Aug2]
GDI_Aug90 = np.asarray(G2_Aug)

K_Aug90 = open('Aug_2020_KHI_90', 'r')
K_Aug2= K_Aug90.readlines()
K2_Aug = [float(i) for i in K_Aug2]
KHI_Aug90 = np.asarray(K2_Aug)
```

[ ]:
```
T_Aug90 = open('Aug_2020_time_90', 'r')
T_Aug2 = T_Aug90.readlines()
T2_Aug = [float(i) for i in T_Aug2]
Time_Aug90 = np.asarray(T2_Aug)

# Sample DataFrame with float column representing Unix␣
 ↪timestamps
data_aug90 = {'timestamp': Time_Aug90}
time_aug90 = pd.DataFrame(data_aug90)

# Convert float to datetime using Pandas' to_datetime()␣
 ↪function
time_aug90['timestamp'] = pd.
 ↪to_datetime(time_aug90['timestamp'], unit='ns')
```

[ ]:
```
G_Sep90 = open('Sep_2020_GDI_90', 'r')
G_Sep2 = G_Sep90.readlines()
G2_Sep = [float(i) for i in G_Sep2]
GDI_Sep90 = np.asarray(G2_Sep)

K_Sep90 = open('Sep_2020_KHI_90', 'r')
K_Sep2 = K_Sep90.readlines()
K2_Sep = [float(i) for i in K_Sep2]
KHI_Sep90 = np.asarray(K2_Sep)

T_Sep90 = open('Sep_2020_time_90', 'r')
T_Sep2 = T_Sep90.readlines()
T2_Sep = [float(i) for i in T_Sep2]
Time_Sep90 = np.asarray(T2_Sep)
```

[ ]:
```
# Sample DataFrame with float column representing Unix␣
 ↪timestamps
data_sep90 = {'timestamp': Time_Sep90}
time_sep90 = pd.DataFrame(data_sep90)

# Convert float to datetime using Pandas' to_datetime()␣
 ↪function
time_sep90['timestamp'] = pd.
 ↪to_datetime(time_sep90['timestamp'], unit='ns')
```

[ ]:
```
G_Okt90 = open('Okt_2020_GDI_90', 'r')
G_Okt2 = G_Okt90.readlines()
G2_Okt = [float(i) for i in G_Okt2]
GDI_Okt90 = np.asarray(G2_Okt)

K_Okt90 = open('Okt_2020_KHI_90', 'r')
K_Okt2 = K_Okt90.readlines()
K2_Okt = [float(i) for i in K_Okt2]
KHI_Okt90 = np.asarray(K2_Okt)
```

[ ]:
```
T_Okt90 = open('Okt_2020_time_90', 'r')
T_Okt2 = T_Okt90.readlines()
T2_Okt = [float(i) for i in T_Okt2]
Time_Okt90 = np.asarray(T2_Okt)

# Sample DataFrame with float column representing Unix␣
 ↪timestamps
data_okt90 = {'timestamp': Time_Okt90}
time_okt90 = pd.DataFrame(data_okt90)

# Convert float to datetime using Pandas' to_datetime()␣
 ↪function
time_okt90['timestamp'] = pd.
 ↪to_datetime(time_okt90['timestamp'], unit='ns')
```

[ ]:
```
G_Nov90 = open('Nov_2020_GDI_90', 'r')
G_Nov2 = G_Nov90.readlines()
G2_Nov = [float(i) for i in G_Nov2]
GDI_Nov90 = np.asarray(G2_Nov)

K_Nov90 = open('Nov_2020_KHI_90', 'r')
K_Nov2 = K_Nov90.readlines()
K2_Nov = [float(i) for i in K_Nov2]
KHI_Nov90 = np.asarray(K2_Nov)
```

[ ]:
```
T_Nov90 = open('Nov_2020_time_90', 'r')
T_Nov2 = T_Nov90.readlines()
T2_Nov = [float(i) for i in T_Nov2]
Time_Nov90 = np.asarray(T2_Nov)

# Sample DataFrame with float column representing Unix␣
 ↪timestamps
data_nov90 = {'timestamp': Time_Nov90}
time_nov90 = pd.DataFrame(data_nov90)

# Convert float to datetime using Pandas' to_datetime()␣
 ↪function
time_nov90['timestamp'] = pd.
 ↪to_datetime(time_nov90['timestamp'], unit='ns')
```

[ ]:
```
G_Des90 = open('Des_2020_GDI_90', 'r')
G_Des2 = G_Des90.readlines()
G2_Des = [float(i) for i in G_Des2]
GDI_Des90 = np.asarray(G2_Des)

K_Des90 = open('Des_2020_KHI_90', 'r')
K_Des2 = K_Des90.readlines()
K2_Des = [float(i) for i in K_Des2]
KHI_Des90 = np.asarray(K2_Des)

T_Des90 = open('Des_2020_time_90', 'r')
T_Des2 = T_Des90.readlines()
T2_Des = [float(i) for i in T_Des2]
Time_Des90 = np.asarray(T2_Des)
```

[ ]:
```
# Sample DataFrame with float column representing Unix␣
 ↪timestamps
data_des90 = {'timestamp': Time_Des90}
time_des90 = pd.DataFrame(data_des90)

# Convert float to datetime using Pandas' to_datetime()␣
 ↪function
time_des90['timestamp'] = pd.
 ↪to_datetime(time_des90['timestamp'], unit='ns')
```

[ ]:
```
GDI_90_20 = np.concatenate([abs(GDI_Jan90), abs(GDI_Feb90),␣
↪abs(GDI_Mar90), abs(GDI_Apr90), abs(GDI_Mai90),
                            abs(GDI_Jun90), abs(GDI_Jul90),␣
↪abs(GDI_Aug90), abs(GDI_Sep90), abs(GDI_Okt90),
                            abs(GDI_Nov90), abs(GDI_Des90)])

KHI_90_20 = np.concatenate([abs(KHI_Jan90), abs(KHI_Feb90),␣
↪abs(KHI_Mar90), abs(KHI_Apr90), abs(KHI_Mai90),
                            abs(KHI_Jun90), abs(KHI_Jul90),␣
↪abs(KHI_Aug90), abs(KHI_Sep90), abs(KHI_Okt90),
                            abs(KHI_Nov90), abs(KHI_Des90)])

time_90_20 = np.concatenate([time_jan90, time_feb90,␣
↪time_mar90, time_apr90, time_mai90, time_jun90,␣
↪time_jul90,
                             time_aug90, time_sep90,␣
↪time_okt90, time_nov90, time_des90])
```

[ ]:
```
G_Jan75 = open('Jan_2015_GDI_75', 'r')
G_Jan1 = G_Jan75.readlines()
G1_Jan = [float(i) for i in G_Jan1]
GDI_Jan75 = np.asarray(G1_Jan)

K_Jan75 = open('Jan_2015_KHI_75', 'r')
K_Jan1 = K_Jan75.readlines()
K1_Jan = [float(i) for i in K_Jan1]
KHI_Jan75 = np.asarray(K1_Jan)

T_Jan75 = open('Jan_2015_time_75', 'r')
T_Jan1 = T_Jan75.readlines()
T1_Jan = [float(i) for i in T_Jan1]
Time_Jan75 = np.asarray(T1_Jan)
```

[ ]:
```
# Sample DataFrame with float column representing Unix␣
 ↪timestamps
data_jan75 = {'timestamp': Time_Jan75}
time_jan75 = pd.DataFrame(data_jan75)

# Convert float to datetime using Pandas' to_datetime()␣
 ↪function
time_jan75['timestamp'] = pd.
 ↪to_datetime(time_jan75['timestamp'], unit='ns')



G_Feb75 = open('Feb_2015_GDI_75', 'r')
G_Feb1 = G_Feb75.readlines()
G1_Feb = [float(i) for i in G_Feb1]
GDI_Feb75 = np.asarray(G1_Feb)

K_Feb75 = open('Feb_2015_KHI_75', 'r')
K_Feb1 = K_Feb75.readlines()
K1_Feb = [float(i) for i in K_Feb1]
KHI_Feb75 = np.asarray(K1_Feb)
```

[ ]:
```
T_Feb75 = open('Feb_2015_time_75', 'r')
T_Feb1 = T_Feb75.readlines()
T1_Feb = [float(i) for i in T_Feb1]
Time_Feb75 = np.asarray(T1_Feb)

# Sample DataFrame with float column representing Unix␣
 ↪timestamps
data_feb75 = {'timestamp': Time_Feb75}
time_feb75 = pd.DataFrame(data_feb75)

# Convert float to datetime using Pandas' to_datetime()␣
 ↪function
time_feb75['timestamp'] = pd.
 ↪to_datetime(time_feb75['timestamp'], unit='ns')
```

[ ]:
```
G_Mar75 = open('Mar_2015_GDI_75', 'r')
G_Mar1 = G_Mar75.readlines()
G1_Mar = [float(i) for i in G_Mar1]
GDI_Mar75 = np.asarray(G1_Mar)

K_Mar75 = open('Mar_2015_KHI_75', 'r')
K_Mar1 = K_Mar75.readlines()
K1_Mar = [float(i) for i in K_Mar1]
KHI_Mar75 = np.asarray(K1_Mar)

T_Mar75 = open('Mar_2015_time_75', 'r')
T_Mar1 = T_Mar75.readlines()
T1_Mar = [float(i) for i in T_Mar1]
Time_Mar75 = np.asarray(T1_Mar)
```

[ ]:
```
# Sample DataFrame with float column representing Unix␣
 ↪timestamps
data_mar75 = {'timestamp': Time_Mar75}
time_mar75 = pd.DataFrame(data_mar75)

# Convert float to datetime using Pandas' to_datetime()␣
 ↪function
time_mar75['timestamp'] = pd.
 ↪to_datetime(time_mar75['timestamp'], unit='ns')



G_Apr75 = open('Apr_2015_GDI_75', 'r')
G_Apr1 = G_Apr75.readlines()
G1_Apr = [float(i) for i in G_Apr1]
GDI_Apr75 = np.asarray(G1_Apr)
```

[ ]:
```
K_Apr75 = open('Apr_2015_KHI_75', 'r')
K_Apr1 = K_Apr75.readlines()
K1_Apr = [float(i) for i in K_Apr1]
KHI_Apr75 = np.asarray(K1_Apr)

T_Apr75 = open('Apr_2015_time_75', 'r')
T_Apr1 = T_Apr75.readlines()
T1_Apr = [float(i) for i in T_Apr1]
Time_Apr75 = np.asarray(T1_Apr)
```

[ ]:
```
# Sample DataFrame with float column representing Unix␣
↪timestamps
data_apr75 = {'timestamp': Time_Apr75}
time_apr75 = pd.DataFrame(data_apr75)

# Convert float to datetime using Pandas' to_datetime()␣
↪function
time_apr75['timestamp'] = pd.
↪to_datetime(time_apr75['timestamp'], unit='ns')
```

[ ]:
```
G_Mai75 = open('Mai_2015_GDI_75', 'r')
G_Mai1 = G_Mai75.readlines()
G1_Mai = [float(i) for i in G_Mai1]
GDI_Mai75 = np.asarray(G1_Mai)

K_Mai75 = open('Mai_2015_KHI_75', 'r')
K_Mai1 = K_Mai75.readlines()
K1_Mai = [float(i) for i in K_Mai1]
KHI_Mai75 = np.asarray(K1_Mai)

T_Mai75 = open('Mai_2015_time_75', 'r')
T_Mai1 = T_Mai75.readlines()
T1_Mai = [float(i) for i in T_Mai1]
Time_Mai75 = np.asarray(T1_Mai)
```

[ ]:
```
# Sample DataFrame with float column representing Unix␣
↪timestamps
data_mai75 = {'timestamp': Time_Mai75}
time_mai75 = pd.DataFrame(data_mai75)

# Convert float to datetime using Pandas' to_datetime()␣
↪function
time_mai75['timestamp'] = pd.
↪to_datetime(time_mai75['timestamp'], unit='ns')



G_Jun75 = open('Jun_2015_GDI_75', 'r')
G_Jun1 = G_Jun75.readlines()
G1_Jun = [float(i) for i in G_Jun1]
GDI_Jun75 = np.asarray(G1_Jun)

K_Jun75 = open('Jun_2015_KHI_75', 'r')
K_Jun1 = K_Jun75.readlines()
K1_Jun = [float(i) for i in K_Jun1]
KHI_Jun75 = np.asarray(K1_Jun)
```

[ ]:
```
T_Jun75 = open('Jun_2015_time_75', 'r')
T_Jun1 = T_Jun75.readlines()
T1_Jun = [float(i) for i in T_Jun1]
Time_Jun75 = np.asarray(T1_Jun)

# Sample DataFrame with float column representing Unix␣
↪timestamps
data_jun75 = {'timestamp': Time_Jun75}
time_jun75 = pd.DataFrame(data_jun75)

# Convert float to datetime using Pandas' to_datetime()␣
↪function
time_jun75['timestamp'] = pd.
↪to_datetime(time_jun75['timestamp'], unit='ns')
```

[ ]:
```
G_Jul75 = open('Jul_2015_GDI_75', 'r')
G_Jul1 = G_Jul75.readlines()
G1_Jul = [float(i) for i in G_Jul1]
GDI_Jul75 = np.asarray(G1_Jul)

K_Jul75 = open('Jul_2015_KHI_75', 'r')
K_Jul1 = K_Jul75.readlines()
K1_Jul = [float(i) for i in K_Jul1]
KHI_Jul75 = np.asarray(K1_Jul)
```

[ ]:
```
T_Jul75 = open('Jul_2015_time_75', 'r')
T_Jul1 = T_Jul75.readlines()
T1_Jul = [float(i) for i in T_Jul1]
Time_Jul75 = np.asarray(T1_Jul)

# Sample DataFrame with float column representing Unix␣
 ↪timestamps
data_jul75 = {'timestamp': Time_Jul75}
time_jul75 = pd.DataFrame(data_jul75)

# Convert float to datetime using Pandas' to_datetime()␣
 ↪function
time_jul75['timestamp'] = pd.
 ↪to_datetime(time_jul75['timestamp'], unit='ns')
```

[ ]:
```
G_Aug75 = open('Aug_2015_GDI_75', 'r')
G_Aug1 = G_Aug75.readlines()
G1_Aug = [float(i) for i in G_Aug1]
GDI_Aug75 = np.asarray(G1_Aug)

K_Aug75 = open('Aug_2015_KHI_75', 'r')
K_Aug1 = K_Aug75.readlines()
K1_Aug = [float(i) for i in K_Aug1]
KHI_Aug75 = np.asarray(K1_Aug)

T_Aug75 = open('Aug_2015_time_75', 'r')
T_Aug1 = T_Aug75.readlines()
T1_Aug = [float(i) for i in T_Aug1]
Time_Aug75 = np.asarray(T1_Aug)

# Sample DataFrame with float column representing Unix␣
 ↪timestamps
data_aug75 = {'timestamp': Time_Aug75}
time_aug75 = pd.DataFrame(data_aug75)
```

[ ]:
```
# Convert float to datetime using Pandas' to_datetime()␣
 ↪function
time_aug75['timestamp'] = pd.
 ↪to_datetime(time_aug75['timestamp'], unit='ns')



G_Sep75 = open('Sep_2015_GDI_75', 'r')
G_Sep1 = G_Sep75.readlines()
G1_Sep = [float(i) for i in G_Sep1]
GDI_Sep75 = np.asarray(G1_Sep)
```

[ ]:
```
K_Sep75 = open('Sep_2015_KHI_75', 'r')
K_Sep1 = K_Sep75.readlines()
K1_Sep = [float(i) for i in K_Sep1]
KHI_Sep75 = np.asarray(K1_Sep)

T_Sep75 = open('Sep_2015_time_75', 'r')
T_Sep1 = T_Sep75.readlines()
T1_Sep = [float(i) for i in T_Sep1]
Time_Sep75 = np.asarray(T1_Sep)
```

[ ]:
```
# Sample DataFrame with float column representing Unix␣
↪timestamps
data_sep75 = {'timestamp': Time_Sep75}
time_sep75 = pd.DataFrame(data_sep75)

# Convert float to datetime using Pandas' to_datetime()␣
↪function
time_sep75['timestamp'] = pd.
↪to_datetime(time_sep75['timestamp'], unit='ns')
```

[ ]:
```

G_Okt75 = open('Okt_2015_GDI_75', 'r')
G_Okt1 = G_Okt75.readlines()
G1_Okt = [float(i) for i in G_Okt1]
GDI_Okt75 = np.asarray(G1_Okt)

K_Okt75 = open('Okt_2015_KHI_75', 'r')
K_Okt1 = K_Okt75.readlines()
K1_Okt = [float(i) for i in K_Okt1]
KHI_Okt75 = np.asarray(K1_Okt)

T_Okt75 = open('Okt_2015_time_75', 'r')
T_Okt1 = T_Okt75.readlines()
T1_Okt = [float(i) for i in T_Okt1]
Time_Okt75 = np.asarray(T1_Okt)
```

[ ]:
```
# Sample DataFrame with float column representing Unix␣
↪timestamps
data_okt75 = {'timestamp': Time_Okt75}
time_okt75 = pd.DataFrame(data_okt75)

# Convert float to datetime using Pandas' to_datetime()␣
↪function
time_okt75['timestamp'] = pd.
↪to_datetime(time_okt75['timestamp'], unit='ns')
```

[ ]:
```
G_Nov75 = open('Nov_2015_GDI_75', 'r')
G_Nov1 = G_Nov75.readlines()
G1_Nov = [float(i) for i in G_Nov1]
GDI_Nov75 = np.asarray(G1_Nov)

K_Nov75 = open('Nov_2015_KHI_75', 'r')
K_Nov1 = K_Nov75.readlines()
K1_Nov = [float(i) for i in K_Nov1]
KHI_Nov75 = np.asarray(K1_Nov)

T_Nov75 = open('Nov_2015_time_75', 'r')
T_Nov1 = T_Nov75.readlines()
T1_Nov = [float(i) for i in T_Nov1]
Time_Nov75 = np.asarray(T1_Nov)
```

[ ]:
```
# Sample DataFrame with float column representing Unix␣
 ↪timestamps
data_nov75 = {'timestamp': Time_Nov75}
time_nov75 = pd.DataFrame(data_nov75)

# Convert float to datetime using Pandas' to_datetime()␣
 ↪function
time_nov75['timestamp'] = pd.
 ↪to_datetime(time_nov75['timestamp'], unit='ns')



G_Des75 = open('Des_2015_GDI_75', 'r')
G_Des1 = G_Des75.readlines()
G1_Des = [float(i) for i in G_Des1]
GDI_Des75 = np.asarray(G1_Des)

K_Des75 = open('Des_2015_KHI_75', 'r')
K_Des1 = K_Des75.readlines()
K1_Des = [float(i) for i in K_Des1]
KHI_Des75 = np.asarray(K1_Des)
```

[ ]:
```
T_Des75 = open('Des_2015_time_75', 'r')
T_Des1 = T_Des75.readlines()
T1_Des = [float(i) for i in T_Des1]
Time_Des75 = np.asarray(T1_Des)

# Sample DataFrame with float column representing Unix␣
 ↪timestamps
data_des75 = {'timestamp': Time_Des75}
time_des75 = pd.DataFrame(data_des75)

# Convert float to datetime using Pandas' to_datetime()␣
 ↪function
time_des75['timestamp'] = pd.
 ↪to_datetime(time_des75['timestamp'], unit='ns')
```

[ ]:
```
GDI_75_15 = np.concatenate([abs(GDI_Jan75), abs(GDI_Feb75),␣
 ↪abs(GDI_Mar75), abs(GDI_Apr75), abs(GDI_Mai75),
                            abs(GDI_Jun75), abs(GDI_Jul75),␣
 ↪abs(GDI_Aug75), abs(GDI_Sep75), abs(GDI_Okt75),
                            abs(GDI_Nov75), abs(GDI_Des75)])

KHI_75_15 = np.concatenate([abs(KHI_Jan75), abs(KHI_Feb75),␣
 ↪abs(KHI_Mar75), abs(KHI_Apr75), abs(KHI_Mai75),
                            abs(KHI_Jun75), abs(KHI_Jul75),␣
 ↪abs(KHI_Aug75), abs(KHI_Sep75), abs(KHI_Okt75),
                            abs(KHI_Nov75), abs(KHI_Des75)])

time_75_15 = np.concatenate([time_jan75, time_feb75,␣
 ↪time_mar75, time_apr75, time_mai75, time_jun75,␣
 ↪time_jul75,
                             time_aug75, time_sep75,␣
 ↪time_okt75, time_nov75, time_des75])
```

[ ]:
```
G_Jan90 = open('Jan_2015_GDI_90', 'r')
G_Jan2 = G_Jan90.readlines()
G2_Jan = [float(i) for i in G_Jan2]
GDI_Jan90 = np.asarray(G2_Jan)

K_Jan90 = open('Jan_2015_KHI_90', 'r')
K_Jan2 = K_Jan90.readlines()
K2_Jan = [float(i) for i in K_Jan2]
KHI_Jan90 = np.asarray(K2_Jan)

T_Jan90 = open('Jan_2015_time_90', 'r')
T_Jan2 = T_Jan90.readlines()
T2_Jan = [float(i) for i in T_Jan2]
Time_Jan90 = np.asarray(T2_Jan)

# Sample DataFrame with float column representing Unix␣
 ↪timestamps
data_jan90 = {'timestamp': Time_Jan90}
time_jan90 = pd.DataFrame(data_jan90)
```

[ ]:
```
# Convert float to datetime using Pandas' to_datetime()␣
 ↪function
time_jan90['timestamp'] = pd.
 ↪to_datetime(time_jan90['timestamp'], unit='ns')



G_Feb90 = open('Feb_2015_GDI_90', 'r')
G_Feb2 = G_Feb90.readlines()
G2_Feb = [float(i) for i in G_Feb2]
GDI_Feb90 = np.asarray(G2_Feb)

K_Feb90 = open('Feb_2015_KHI_90', 'r')
K_Feb2 = K_Feb90.readlines()
K2_Feb = [float(i) for i in K_Feb2]
KHI_Feb90 = np.asarray(K2_Feb)
```

[ ]:
```
T_Feb90 = open('Feb_2015_time_90', 'r')
T_Feb2 = T_Feb90.readlines()
T2_Feb = [float(i) for i in T_Feb2]
Time_Feb90 = np.asarray(T2_Feb)

# Sample DataFrame with float column representing Unix␣
 ↪timestamps
data_feb90 = {'timestamp': Time_Feb90}
time_feb90 = pd.DataFrame(data_feb90)

# Convert float to datetime using Pandas' to_datetime()␣
 ↪function
time_feb90['timestamp'] = pd.
 ↪to_datetime(time_feb90['timestamp'], unit='ns')
```

[ ]:
```

G_Mar90 = open('Mar_2015_GDI_90', 'r')
G_Mar2 = G_Mar90.readlines()
G2_Mar = [float(i) for i in G_Mar2]
GDI_Mar90 = np.asarray(G2_Mar)

K_Mar90 = open('Mar_2015_KHI_90', 'r')
K_Mar2 = K_Mar90.readlines()
K2_Mar = [float(i) for i in K_Mar2]
KHI_Mar90 = np.asarray(K2_Mar)
```

[ ]:
```
T_Mar90 = open('Mar_2015_time_90', 'r')
T_Mar2 = T_Mar90.readlines()
T2_Mar = [float(i) for i in T_Mar2]
Time_Mar90 = np.asarray(T2_Mar)

# Sample DataFrame with float column representing Unix␣
 ↪timestamps
data_mar90 = {'timestamp': Time_Mar90}
time_mar90 = pd.DataFrame(data_mar90)

# Convert float to datetime using Pandas' to_datetime()␣
 ↪function
time_mar90['timestamp'] = pd.
 ↪to_datetime(time_mar90['timestamp'], unit='ns')
```

[ ]:
```
G_Apr90 = open('Apr_2015_GDI_90', 'r')
G_Apr2 = G_Apr90.readlines()
G2_Apr = [float(i) for i in G_Apr2]
GDI_Apr90 = np.asarray(G2_Apr)

K_Apr90 = open('Apr_2015_KHI_90', 'r')
K_Apr2 = K_Apr90.readlines()
K2_Apr = [float(i) for i in K_Apr2]
KHI_Apr90 = np.asarray(K2_Apr)

T_Apr90 = open('Apr_2015_time_90', 'r')
T_Apr2 = T_Apr90.readlines()
T2_Apr = [float(i) for i in T_Apr2]
Time_Apr90 = np.asarray(T2_Apr)
```

[ ]:
```
# Sample DataFrame with float column representing Unix␣
 ↪timestamps
data_apr90 = {'timestamp': Time_Apr90}
time_apr90 = pd.DataFrame(data_apr90)

# Convert float to datetime using Pandas' to_datetime()␣
 ↪function
time_apr90['timestamp'] = pd.
 ↪to_datetime(time_apr90['timestamp'], unit='ns')
```

[ ]:
```


G_Mai90 = open('Mai_2015_GDI_90', 'r')
G_Mai2 = G_Mai90.readlines()
G2_Mai = [float(i) for i in G_Mai2]
GDI_Mai90 = np.asarray(G2_Mai)

K_Mai90 = open('Mai_2015_KHI_90', 'r')
K_Mai2 = K_Mai90.readlines()
K2_Mai = [float(i) for i in K_Mai2]
KHI_Mai90 = np.asarray(K2_Mai)

T_Mai90 = open('Mai_2015_time_90', 'r')
T_Mai2 = T_Mai90.readlines()
T2_Mai = [float(i) for i in T_Mai2]
Time_Mai90 = np.asarray(T2_Mai)
```

[ ]:
```
# Sample DataFrame with float column representing Unix␣
↪timestamps
data_mai90 = {'timestamp': Time_Mai90}
time_mai90 = pd.DataFrame(data_mai90)

# Convert float to datetime using Pandas' to_datetime()␣
↪function
time_mai90['timestamp'] = pd.
↪to_datetime(time_mai90['timestamp'], unit='ns')
```

[ ]:
```
G_Jun90 = open('Jun_2015_GDI_90', 'r')
G_Jun2 = G_Jun90.readlines()
G2_Jun = [float(i) for i in G_Jun2]
GDI_Jun90 = np.asarray(G2_Jun)

K_Jun90 = open('Jun_2015_KHI_90', 'r')
K_Jun2 = K_Jun90.readlines()
K2_Jun = [float(i) for i in K_Jun2]
KHI_Jun90 = np.asarray(K2_Jun)

T_Jun90 = open('Jun_2015_time_90', 'r')
T_Jun2 = T_Jun90.readlines()
T2_Jun = [float(i) for i in T_Jun2]
Time_Jun90 = np.asarray(T2_Jun)
```

[ ]:
```
# Sample DataFrame with float column representing Unix␣
↪timestamps
data_jun90 = {'timestamp': Time_Jun90}
time_jun90 = pd.DataFrame(data_jun90)

# Convert float to datetime using Pandas' to_datetime()␣
↪function
time_jun90['timestamp'] = pd.
↪to_datetime(time_jun90['timestamp'], unit='ns')
```

[ ]:
```
G_Jul90 = open('Jul_2015_GDI_90', 'r')
G_Jul2 = G_Jul90.readlines()
G2_Jul = [float(i) for i in G_Jul2]
GDI_Jul90 = np.asarray(G2_Jul)

K_Jul90 = open('Jul_2015_KHI_90', 'r')
K_Jul2 = K_Jul90.readlines()
K2_Jul = [float(i) for i in K_Jul2]
KHI_Jul90 = np.asarray(K2_Jul)

T_Jul90 = open('Jul_2015_time_90', 'r')
T_Jul2 = T_Jul90.readlines()
T2_Jul = [float(i) for i in T_Jul2]
Time_Jul90 = np.asarray(T2_Jul)
```

[ ]:
```
# Sample DataFrame with float column representing Unix␣
 ↪timestamps
data_jul90 = {'timestamp': Time_Jul90}
time_jul90 = pd.DataFrame(data_jul90)

# Convert float to datetime using Pandas' to_datetime()␣
 ↪function
time_jul90['timestamp'] = pd.
 ↪to_datetime(time_jul90['timestamp'], unit='ns')



G_Aug90 = open('Aug_2015_GDI_90', 'r')
G_Aug2 = G_Aug90.readlines()
G2_Aug = [float(i) for i in G_Aug2]
GDI_Aug90 = np.asarray(G2_Aug)
```

[ ]:
```
K_Aug90 = open('Aug_2015_KHI_90', 'r')
K_Aug2= K_Aug90.readlines()
K2_Aug = [float(i) for i in K_Aug2]
KHI_Aug90 = np.asarray(K2_Aug)

T_Aug90 = open('Aug_2015_time_90', 'r')
T_Aug2 = T_Aug90.readlines()
T2_Aug = [float(i) for i in T_Aug2]
Time_Aug90 = np.asarray(T2_Aug)

# Sample DataFrame with float column representing Unix␣
 ↪timestamps
data_aug90 = {'timestamp': Time_Aug90}
time_aug90 = pd.DataFrame(data_aug90)
```

[ ]:
```
# Convert float to datetime using Pandas' to_datetime()␣
 ↪function
time_aug90['timestamp'] = pd.
 ↪to_datetime(time_aug90['timestamp'], unit='ns')



G_Sep90 = open('Sep_2015_GDI_90', 'r')
G_Sep2 = G_Sep90.readlines()
G2_Sep = [float(i) for i in G_Sep2]
GDI_Sep90 = np.asarray(G2_Sep)

K_Sep90 = open('Sep_2015_KHI_90', 'r')
K_Sep2 = K_Sep90.readlines()
K2_Sep = [float(i) for i in K_Sep2]
KHI_Sep90 = np.asarray(K2_Sep)

T_Sep90 = open('Sep_2015_time_90', 'r')
T_Sep2 = T_Sep90.readlines()
T2_Sep = [float(i) for i in T_Sep2]
Time_Sep90 = np.asarray(T2_Sep)
```

[ ]:
```
# Sample DataFrame with float column representing Unix␣
 ↪timestamps
data_sep90 = {'timestamp': Time_Sep90}
time_sep90 = pd.DataFrame(data_sep90)

# Convert float to datetime using Pandas' to_datetime()␣
 ↪function
time_sep90['timestamp'] = pd.
 ↪to_datetime(time_sep90['timestamp'], unit='ns')



G_Okt90 = open('Okt_2015_GDI_90', 'r')
G_Okt2 = G_Okt90.readlines()
G2_Okt = [float(i) for i in G_Okt2]
GDI_Okt90 = np.asarray(G2_Okt)

K_Okt90 = open('Okt_2015_KHI_90', 'r')
K_Okt2 = K_Okt90.readlines()
K2_Okt = [float(i) for i in K_Okt2]
KHI_Okt90 = np.asarray(K2_Okt)
```

[ ]:
```
T_Okt90 = open('Okt_2015_time_90', 'r')
T_Okt2 = T_Okt90.readlines()
T2_Okt = [float(i) for i in T_Okt2]
Time_Okt90 = np.asarray(T2_Okt)

# Sample DataFrame with float column representing Unix␣
 ↪timestamps
data_okt90 = {'timestamp': Time_Okt90}
time_okt90 = pd.DataFrame(data_okt90)

# Convert float to datetime using Pandas' to_datetime()␣
 ↪function
time_okt90['timestamp'] = pd.
 ↪to_datetime(time_okt90['timestamp'], unit='ns')
```

[ ]:
```
G_Nov90 = open('Nov_2015_GDI_90', 'r')
G_Nov2 = G_Nov90.readlines()
G2_Nov = [float(i) for i in G_Nov2]
GDI_Nov90 = np.asarray(G2_Nov)

K_Nov90 = open('Nov_2015_KHI_90', 'r')
K_Nov2 = K_Nov90.readlines()
K2_Nov = [float(i) for i in K_Nov2]
KHI_Nov90 = np.asarray(K2_Nov)

T_Nov90 = open('Nov_2015_time_90', 'r')
T_Nov2 = T_Nov90.readlines()
T2_Nov = [float(i) for i in T_Nov2]
Time_Nov90 = np.asarray(T2_Nov)

# Sample DataFrame with float column representing Unix␣
 ↪timestamps
data_nov90 = {'timestamp': Time_Nov90}
time_nov90 = pd.DataFrame(data_nov90)
```

[ ]:
```
# Convert float to datetime using Pandas' to_datetime()␣
 ↪function
time_nov90['timestamp'] = pd.
 ↪to_datetime(time_nov90['timestamp'], unit='ns')



G_Des90 = open('Des_2015_GDI_90', 'r')
G_Des2 = G_Des90.readlines()
G2_Des = [float(i) for i in G_Des2]
GDI_Des90 = np.asarray(G2_Des)

K_Des90 = open('Des_2015_KHI_90', 'r')
K_Des2 = K_Des90.readlines()
K2_Des = [float(i) for i in K_Des2]
KHI_Des90 = np.asarray(K2_Des)

T_Des90 = open('Des_2015_time_90', 'r')
T_Des2 = T_Des90.readlines()
T2_Des = [float(i) for i in T_Des2]
Time_Des90 = np.asarray(T2_Des)
```

[ ]:
```
# Sample DataFrame with float column representing Unix␣
↪timestamps
data_des90 = {'timestamp': Time_Des90}
time_des90 = pd.DataFrame(data_des90)

# Convert float to datetime using Pandas' to_datetime()␣
↪function
time_des90['timestamp'] = pd.
↪to_datetime(time_des90['timestamp'], unit='ns')
```

[ ]:
```
GDI_90_15 = np.concatenate([abs(GDI_Jan90), abs(GDI_Feb90),␣
↪abs(GDI_Mar90), abs(GDI_Apr90), abs(GDI_Mai90),
                            abs(GDI_Jun90), abs(GDI_Jul90),␣
↪abs(GDI_Aug90), abs(GDI_Sep90), abs(GDI_Okt90),
                            abs(GDI_Nov90), abs(GDI_Des90)])

KHI_90_15 = np.concatenate([abs(KHI_Jan90), abs(KHI_Feb90),␣
↪abs(KHI_Mar90), abs(KHI_Apr90), abs(KHI_Mai90),
                            abs(KHI_Jun90), abs(KHI_Jul90),␣
↪abs(KHI_Aug90), abs(KHI_Sep90), abs(KHI_Okt90),
                            abs(KHI_Nov90), abs(KHI_Des90)])

time_90_15 = np.concatenate([time_jan90, time_feb90,␣
↪time_mar90, time_apr90, time_mai90, time_jun90,␣
↪time_jul90,
                             time_aug90, time_sep90,␣
↪time_okt90, time_nov90, time_des90])
```

[ ]:
```
fig, (ax1, ax2, ax3, ax4) = plt.subplots(4, sharex=True,␣
 ↪figsize=(10, 7), constrained_layout = True)

plt.rcParams.update({'font.size': 15})

ax1.hist(GDI_75_15, bins=1000, range=(0.01, 1), label='GDI␣
 ↪Latitude 60-75 2015')
ax2.hist(GDI_75_20, bins=1000, range=(0.01, 1), label='GDI␣
 ↪Latitude 60-75 2020')
ax3.hist(GDI_90_15, bins=1000, range=(0.01, 1), label='GDI␣
 ↪Latitude 75-90 2015')
ax4.hist(GDI_90_20, bins=1000, range=(0.01, 1), label='GDI␣
 ↪Latitude 75-90 2020')

ax1.legend()
ax2.legend()
ax3.legend()
ax4.legend()
```

[ ]:
```
ax1.set_title('Latitude between 60-75 degrees')
ax3.set_title('Latitude between 75-90 degrees')

ax1.set_yscale('log')
ax2.set_yscale('log')
ax3.set_yscale('log')
ax4.set_yscale('log')

ax1.set_ylim(top=10**6)
ax2.set_ylim(top=10**6)
ax3.set_ylim(top=10**6)
ax4.set_ylim(top=10**6)

ax1.tick_params(axis='both', which='major', labelsize=20)
ax2.tick_params(axis='both', which='major', labelsize=20)
ax3.tick_params(axis='both', which='major', labelsize=20)
ax4.tick_params(axis='both', which='major', labelsize=20)
```

[ ]:
```
fig.supxlabel('Growth rate [1/s]')
fig.supylabel('Quantity')
fig.suptitle('Histogram of GDI with Kp 1-9', fontsize=20)

plt.show()
```

[ ]:
```
fig, (ax1, ax2, ax3, ax4) = plt.subplots(4, sharex=True,␣
↪figsize=(10, 7), constrained_layout = True)

plt.rcParams.update({'font.size': 15})
```

[ ]:
```
ax1.hist(KHI_75_15, bins=1000, range=(0.01, 1),␣
↪density=True, stacked=True, label='KHI Latitude 60-75␣
↪2015')
ax2.hist(KHI_75_20, bins=1000, range=(0.01, 1),␣
↪density=True, stacked=True, label='KHI Latitude 60-75␣
↪2020')
ax3.hist(KHI_90_15, bins=1000, range=(0.01, 1),␣
↪density=True, stacked=True, label='KHI Latitude 75-90␣
↪2015')
ax4.hist(KHI_90_20, bins=1000, range=(0.01, 1),␣
↪density=True, stacked=True, label='KHI Latitude 75-90␣
↪2020')

ax1.legend()
ax2.legend()
ax3.legend()
ax4.legend()

ax1.set_title('Latitude between 60-75 degrees')
ax3.set_title('Latitude between 75-90 degrees')
```

[ ]:
```
ax1.set_yscale('log')
ax2.set_yscale('log')
ax3.set_yscale('log')
ax4.set_yscale('log')

ax1.set_ylim(top=10**6)
ax2.set_ylim(top=10**6)
ax3.set_ylim(top=10**6)
ax4.set_ylim(top=10**6)

ax1.tick_params(axis='both', which='major', labelsize=20)
ax2.tick_params(axis='both', which='major', labelsize=20)
ax3.tick_params(axis='both', which='major', labelsize=20)
ax4.tick_params(axis='both', which='major', labelsize=20)
```

[ ]:
```
fig.supxlabel('Growth rate [1/s]')
fig.supylabel('Quantity')
fig.suptitle('Histogram of KHI with Kp 1-9', fontsize=20)

plt.show()
```

[ ]:
```
fig, (ax1, ax2) = plt.subplots(2, sharex=True, figsize=(10,␣
↪7), constrained_layout = True)

plt.rcParams.update({'font.size': 15})

ax1.scatter(time_75_15, GDI_75_15, label="GDI latitude 60-75␣
↪2015", marker='o')
ax1.scatter(time_75_20, GDI_75_20, label="GDI latitude 60-75␣
↪2020", marker='o')
ax2.scatter(time_90_15, GDI_90_15, label="GDI latitude 75-90␣
↪2015", marker='o')
ax2.scatter(time_90_20, GDI_90_20, label="GDI latitude 75-90␣
↪2020", marker='o')

ax1.legend()
ax2.legend()
```

[ ]:
```
ax1.set_title('Latitude between 60 and 75 degrees')
ax2.set_title('Latitude between 75 and 90 degrees')

ax1.set_ylim(top=6)
ax2.set_ylim(top=6)

ax1.tick_params(axis='both', which='major', labelsize=20)
ax2.tick_params(axis='both', which='major', labelsize=20)
```

[ ]:
```
fig.supylabel('Growth rate [1/s]')
fig.supxlabel('Time [Date]')
fig.suptitle('GDI with Kp between 1-9', fontsize=20)
plt.xticks(rotation=30)

plt.show()
```

[ ]:
```
fig, (ax1, ax2) = plt.subplots(2, sharex=True, figsize=(10,␣
↪7), constrained_layout = True)

plt.rcParams.update({'font.size': 15})

ax1.scatter(time_75_15, KHI_75_15, label="KHI latitude 60-75␣
↪2015", marker='o')
ax1.scatter(time_75_20, KHI_75_20, label="KHI latitude 60-75␣
↪2020", marker='o')
ax2.scatter(time_90_15, KHI_90_15, label="KHI latitude 75-90␣
↪2015", marker='o')
ax2.scatter(time_90_20, KHI_90_20, label="KHI latitude 75-90␣
↪2020", marker='o')
```

[ ]:
```
ax1.legend()
ax2.legend()

ax1.set_title('Latitude between 60 and 75 degrees')
ax2.set_title('Latitude between 75 and 90 degrees')

ax1.set_ylim(top=0.5)
ax2.set_ylim(top=0.5)

ax1.tick_params(axis='both', which='major', labelsize=20)
ax2.tick_params(axis='both', which='major', labelsize=20)
```

[ ]:
```
fig.supylabel('Growth rate [1/s]')
fig.supxlabel('Time [Date]')
fig.suptitle('KHI with Kp between 1-9', fontsize=20)
plt.xticks(rotation=30)

plt.show()
```