UiT The Arctic University of Norway

Faculty of Science and Technology
Department of Physics and Technology

An Investigation of Turbulence and Intermittency in the High-Latitude Ionosphere: A Multi Case Study

Sondre Tovås

FYS-3931 Master's thesis in Space Physics 30sp June 2024

Contents

List of Figures iii

List of Tables ix

Abstract x

1 Introduction 1

2 Theory 5

2.1 High Latitude Ionosphere and Plasma Structuring 5

2.2 Structure Function 6

2.3 Empirical Flatness 9

2.4 Probability Density Function 10

3 Data Acquisition 11

4 Methods 19

4.1 Structure Function 19

4.2 Empirical Flatness 22

4.3 Probability Density Function 23

5 Results 25

5.1 Structure Function 26

5.1.1 Structure Function and Slope 26

5.1.2 Slope Distribution 30

5.1.3 Area of Structure Function 33

5.2 Empirical Flatness 36

5.2.1 Empirical Flatness and Area Distribution 36

5.2.2 Area of Empirical Flatness 39

5.3 Probability Density Function 42

5.3.1 Small Increment 43

5.3.2 Middle Increment 44

5.3.3 Large Increment 45

6 Discussion 47

6.1 Structure Function 48
6.1.1 Slopes 48
6.1.2 Area 49
6.2 Empirical Flatness 50
6.3 Probability Density Function 54

7 Conclusion **55**
7.1 Future Work 56

8 Appendix **59**

Bibliography **61**
8.1 Table of Days Used 64
8.2 Power Spectral Density 65
8.3 Slopes for Structure Functions at Very Small Scales 66
8.4 AE Index During Storm Event 11th May 2024 66
8.5 Code 67
8.5.1 Calculating Storing and Displaying Data 67
8.5.2 Calculating Slopes and Area 86
8.5.3 Loading Data From File 100
8.5.4 Detecting Polar (Sub)Region 101
8.5.5 Data Processing 105
8.5.6 Functions 118
8.5.7 Plotting Basic Figures 120
8.5.8 Choosing Instance 124

List of Figures

2.1 Structure function of the entire polar region during high geomagnetic activity (blue figure) and low geomagnetic activity (orange figure) 31st and 30th December 2015, respectively. Power law is fitted to each figure (dashed lines). 8
2.2 Corresponding electron density in the polar region in (a) a period of high geomagnetic activity and (b) low geomagnetic activity. 8
2.3 Empirical Flatness of the entire polar region during high geomagnetic activity (blue figure) and low geomagnetic activity (orange figure) 31st and 30th December 2015, respectively. The dashed line represents $F(\tau) = 3$. 9
2.4 PDF of the entire polar region during high geomagnetic activity for three time lags. $\tau = 1.0s$ (dotted blue), $\tau = 10.0s$ (dotted orange) and $\tau = 100.0s$ (dotted green). A Gaussian (solid black) is fitted to the figure. 10

3.1 Figure 3.1a) shows high geomagnetic activity measured 4th November 2014, with the most active period being from approximately 06:00 to 18:00. Figure 3.1b) shows low geomagnetic activity measured 3rd November 2014 [WDC, b]. . . . 13
3.2 The purple line represents the trajectory of the satellite during the high activity day 4th November 2014 while measuring the polar region. The grey and black lines are the trajectories for each pass through the polar region of the satellite during the low activity day. The black line is the closest of these passes to the high activity day. 14
3.3 Measured data from Swarm satellite A during 4th November 2014. Data shown from 06:00 UTC to 16:00 UTC during the period with the highest geomagnetic activity. At approximately 11:00 UTC we highlight a high field-aligned current and plasma density, which corresponds to the measurements seen in Figure 3.1a). The marked data was chosen as instance 1 during this day. 16

3.4 Instance 1 during 4th November 2014. The blue plots are of the plasma density and field-aligned current in Region A, while the orange and green are in Region B and C respectively. 17

3.5 Low activity day on 3rd November 2014. The blue plots are of the plasma density and field-aligned current in Region A, while the orange and green are in Region B and C respectively. 17

4.1 600 seconds example dataset. The blue line represents method A, and the orange line represents method B. The y-axis represents the amount of data points averaged. The x-axis represents the time lag in seconds. 20

5.1 Plasma Density in the polar region in (a) a period of high geomagnetic activity 4th November 2014 and (b) low geomagnetic activity 3rd of November 2014. Each figure is divided into the auroral regions A (blue) and C (green), Region B the polar cap (orange). Both figures share tha same y-axis. . . . 26

5.2 Structure function of order $m = 2$ in the polar region during a day of high geomagnetic activity (blue figure) vs a day of low geomagnetic activity (orange figure). The dashed lines represent the slopes. 27

5.3 Structure function of order $m = 2$ in the polar cap region during a day of high geomagnetic activity (blue figure) vs a day of low geomagnetic activity (orange figure). The dashed lines represent the slopes. 28

5.4 Structure function of order $m = 2$ in region A during a day of high geomagnetic activity (blue figure) vs a day of low geomagnetic activity (orange figure). The dashed lines represent the slopes. 29

5.5 Structure function of order $m = 2$ in region C during a day of high geomagnetic activity (blue figure) vs a day of low geomagnetic activity (orange figure). The dashed lines represent the slopes. 29

5.6 Slopes in polar region. (a) $\tau = [1s, 10s]$ (blue) $\tau = [10s, 100s]$ (orange) during active periods. (b) Active (blue) and quiet (orange) periods for $\tau = [1s, 10s]$. (c) $\tau = [1s, 10s]$ (blue) $\tau = [10s, 100s]$ (orange) during quiet periods. (d) Active (blue) and quiet (orange) periods for $\tau = [10s, 100s]$. 31

5.7 Slopes in polar cap region. (a) $\tau = [1s, 10s]$ (blue) $\tau = [10s, 100s]$ (orange) during active periods. (b) Active (blue) and quiet (orange) periods for $\tau = [1s, 10s]$. (c) $\tau = [1s, 10s]$ (blue) $\tau = [10s, 100s]$ (orange) during quiet periods. (d) Active (blue) and quiet (orange) periods for $\tau = [10s, 100s]$. 31

5.8 Slopes in Region A. (a) $\tau = [1s, 10s]$ (blue) $\tau = [10s, 100s]$ (orange) during active periods. (b) Active (blue) and quiet (orange) periods for $\tau = [1s, 10s]$. (c) $\tau = [1s, 10s]$ (blue) $\tau = [10s, 100s]$ (orange) during quiet periods. (d) Active (blue) and quiet (orange) periods for $\tau = [10s, 100s]$. 32
5.9 Slopes in Region C. (a) $\tau = [1s, 10s]$ (blue) $\tau = [10s, 100s]$ (orange) during active periods. (b) Active (blue) and quiet (orange) periods for $\tau = [1s, 10s]$. (c) $\tau = [1s, 10s]$ (blue) $\tau = [10s, 100s]$ (orange) during quiet periods. (d) Active (blue) and quiet (orange) periods for $\tau = [10s, 100s]$. 33
5.10 Area of structure function in the entire polar region. (a) Area for smaller structures $\tau = [1s, 10s]$ calculated for all instances. (b) Area for larger structures $\tau = [10s, 100s]$ calculated for all instances. 34
5.11 Area of structure function in the polar cap region. (a) Area for smaller structures $\tau = [1s, 10s]$ calculated for all instances. (b) Area for larger structures $\tau = [10s, 100s]$ calculated for all instances. 34
5.12 Area of structure function in region A. (a) Area for smaller structures $\tau = [1s, 10s]$ calculated for all instances. (b) Area for larger structures $\tau = [10s, 100s]$ calculated for all instances. 35
5.13 Area of structure function in region C. (a) Area for smaller structures $\tau = [1s, 10s]$ calculated for all instances. (b) Area for larger structures $\tau = [10s, 100s]$ calculated for all instances. 35
5.14 Dashed line represents $F(\tau) = 3$. (a) active days (blue) and quiet days (orange) for $\tau = [1s, 10s]$. (b) active days (blue) and quiet days (orange) for $\tau = [10s, 100s]$. (c) Area distribution of active days (blue) and quiet days (orange) for $\tau = [1s, 10s]$. (d) Area Distribution for active days (blue) and quiet days (orange) for $\tau = [10s, 100s]$. 37
5.15 Dashed line represents $F(\tau) = 3$. (a) active days (blue) and quiet days (orange) for $\tau = [1s, 10s]$. (b) active days (blue) and quiet days (orange) for $\tau = [10s, 100s]$. (c) Area distribution of active days (blue) and quiet days (orange) for $\tau = [1s, 10s]$. (d) Area distribution of active days (blue) and quiet days (orange) for $\tau = [10s, 100s]$. 37
5.16 Dashed line represents $F(\tau) = 3$. (a) active days (blue) and quiet days (orange) for $\tau = [1s, 10s]$. (b) active days (blue) and quiet days (orange) for $\tau = [10s, 100s]$. (c) Area distribution of active days (blue) and quiet days (orange) for $\tau = [1s, 10s]$. (d) Area distribution of active days (blue) and quiet days (orange) for $\tau = [10s, 100s]$. 38

5.17 Dashed line represents $F(\tau) = 3$. (a) active days (blue) and quiet days (orange) for $\tau = [1s, 10s]$. (b) active days (blue) and quiet days (orange) for $\tau = [10s, 100s]$. (c) Area distribution of active days (blue) and quiet days (orange) for $\tau = [1s, 10s]$. (d) Area distribution of active days (blue) and quiet days (orange) for $\tau = [10s, 100s]$. 39
5.18 Area of empirical flatness in the entire polar region. (a) Area for smaller structures $\tau = [1s, 10s]$ calculated for all instances. (b) Area for larger structures $\tau = [10s, 100s]$ calculated for all instances. 40
5.19 Area of empirical flatness in the polar cap region. (a) Area for smaller structures $\tau = [1s, 10s]$ calculated for all instances. (b) Area for larger structures $\tau = [10s, 100s]$ calculated for all instances. 40
5.20 Area of empirical flatness in region A. (a) Area for smaller structures $\tau = [1s, 10s]$ calculated for all instances. (b) Area for larger structures $\tau = [10s, 100s]$ calculated for all instances. 41
5.21 Area of empirical flatness in region C. (a) Area for smaller structures $\tau = [1s, 10s]$ calculated for all instances. (b) Area for larger structures $\tau = [10s, 100s]$ calculated for all instances. 42
5.22 PDF for five time lags. $\tau = 0.0625s$ (dotted blue), $\tau = 0.125s$ (dotted orange), $\tau = 0.25s$, $\tau = 0.50s$ (dotted red) and $\tau = 1.0s$ (dotted purple). A Gaussian (solid black) is superposed. (a) high activity period on 4th November 2014 (b) low activity period on 3rd November 2014. 43
5.23 PDF for three time lags. $\tau = 1.0s$ (dotted blue), $\tau = 5.0$ (dotted orange) and $\tau = 10.0$. A Gaussian (solid black) is superposed. (a) high activity period on 4th November 2014 (b) low activity period on 3rd November 2014. 44
5.24 PDF for three time lags. $\tau = 1.0s$ (dotted blue), $\tau = 5.0$ (dotted orange) and $\tau = 10.0$. A Gaussian (solid black) is superposed. (a) high activity period on 4th November 2014 (b) low activity period on 3rd November 2014. 45

6.1 Empirical flatness for day of high geomagnetic activity (blue) and day of low geomagnetic activity (orange) 51
6.2 Structure function for $m = 2$ and $m = 4$ during active day 5th December 2015, and quiet day 3rd December 2015. (a) Active $m = 2$ (blue) and quiet $m = 2$ (orange). (b) Active $m = 4$ (green) and quiet $m = 4$ (red). 51
6.3 Average of change in electron density between time t and $\tau = 1s$ of order $m = 2$ 52
6.4 Average of change in electron density between time t and $\tau = 1s$ of order $m = 4$ 53

8.1 Power spectral density during 31st December 2015 over the entire polar region. 65
8.2 Slopes for all structure functions calculated over the entire polar region. (a) Slopes for $\tau = [0.0625s, 1s]$ vs $\tau = [1s, 10s]$ during days of high geomagnetic activity. (b) Slopes during high geomagnetic activity vs low geomagnetic activity for $\tau = [0.0625s, 1s]$. (c) Slopes for $\tau = [0.0625ss, 1s]$ vs $\tau = [1s, 10s]$ during days of low geomagnetic activity. (d) Slopes during high geomagnetic activity vs low geomagnetic activity for $\tau = [1s, 10s]$. 66
8.3 AE index 11th May 2024. [WDC, a] 67

List of Tables

8.1 Dates of Active and Quiet periods 65

Abstract

High-latitude electron density irregularities can affect technology such as the Global Navigation Satellite Services (GNSS) by degrading radio signals. These irregularities are not fully understood. In this work we investigate turbulence and intermittency in varying regions of the high-latitude ionosphere using the structure function, empirical flatness and probability density function. Our aim is to investigate whether these turbulence data analysis tools can reveal additional information about the nature of these phenomenons in the selected regions. We apply these methods on 16 Hz electron density Swarm data across the polar cap and auroral regions in the northern hemisphere. We observed different slopes at large and small scales for the structure function in the polar cap and across the entire polar region. We also noticed a clear difference between the results during days of high and low geomagnetic activity. However, we could not observe any significant differences in the auroral region using this method. Furthermore, the results obtained by measuring the empirical flatness and probability density function remain inconclusive as we did not notice any substantial differences in either region or across the different geomagnetic levels. Our results suggest at least some different scaling behaviour across the entire polar region and polar cap which is affected by geomagnetic storms, but further investigation is required to establish any additional details about turbulence and intermittency.

/1

Introduction

The various mechanisms between Earth's atmosphere, magnetosphere and the solar wind give rise to a layer of plasma in Earth's upper atmosphere. The plasma in this layer, called the ionosphere, exhibits a turbulent nature often due to to ionospheric convection caused by factors such as coupling between the solar wind and magnetosphere [Lester, 2003] [De Michelis et al., 2020]. This turbulent behaviour is characterised by the chaotic fluctuations in the plasma. Energy in the system cascades down to smaller scales from larger through eddies [Frisch, 1995]. The study of turbulence is important as it can affect radio signals in multiple frequencies, like those from Global Navigation Satellite Systems (GNSS) degrading their performance [Moen, Jøran et al., 2013]. Another space weather phenomenon is intermittency. Intermittency is an occurrence characterised by an uneven distribution of the plasma in the turbulent flow. As the energy cascades down to smaller scales, localized structures form, i.e. we do not have a constant energy transfer rate and local scaling exponents exists at different scales [Bruno et al., 2001]. The main characteristic of intermittency is the localized sporadic structures in the plasma turbulence. Cases of large intermittent structures in the polar cap, where the magnetic field lines are open, are called polar cap patches. This region is highly affected by the interplanetary magnetic field (IMF) conditions which controls the convection pattern [Moen, Jøran et al., 2013][Wernik et al., 2003]. In the polar cap region, these so-called polar cap patches can be at 100 km scales. In short, the ionosphere can have plasma structures from tens to thousands of km and 10 to 1 km [Basu et al., 1990] down to irregularities at the scale of the ion gyroradius [Tsunoda, 1988]. These structures are of great interest in the context of space weather and its affect on technology.

We use tools such as the structure function which provides information about the scaling behaviour of the irregularities at different time lags. A property of this scaling behaviour such as scale-invariance is important in the studies of turbulence [Spicher et al., 2015]. Other tools used are the Empirical flatness and the probability density function (PDF), which can detect large fluctuations at smaller timescales. We also mention the use of power spectral density (PSD) which offers insights into the energy in the system. Due to time constraints and many studies [Chian et al., 2008][Phelps and Sagalyn, 1976] [Mandeep et al., 2014] being done using the PSD, we do not use this tool and will not cover further. We add a figure representing the PSD for one of our dataset in the appendix, however. Of the aforementioned tools, the structure function have been the main focus. This is primarily due to the amount of information we can obtain from this tool. It is also a process required to obtain the empirical flatness.

We build on the foundation of several works that comprise both case studies and statistical studies. [Spicher et al., 2015] investigated growth rates associated with polar cap patches using the ICI-2 sounding rocket in the trailing edge of a polar cap patch and a region of electron density enhancements associated with particle precipitation. The ICI-2 sounding rocket provides data at very high resolutions (5787 Hz) but provides limited geographical coverage. The results indicated regions associated with particle precipitation being more random in nature, as opposed to the polar cap where structures are more intermittent. [Consolini et al., 2020] did an analysis of the intermittency in the high-latitude ionospheric region using density measurements captured at 2 Hz by one of the satellites in ESA's swarm constellation during two periods of increased geomagnetic activity. One of the main findings was the signature of intermittency in electron density fluctuations in the auroral oval. [De Michelis et al., 2020] analyzed the scaling features of the electron density fluctuations in the high-latitude ionosphere during the 2015 St. Patrick's geomagnetic storm using two of ESA's swarm satellites obtaining density measurements captured at 1 Hz. The measurements were made throughout the phases of the storm from 16 March to 22 March 2015, with varying geomagnetic activity. Furthermore this work investigated the occurrence of high values of the Rate of change Of the electron Density Index (RODI). The results suggested that the geomagnetic activity level, latitude and magnetic local time affected the different scaling features of the structure function. Furthermore [Tozzi et al., 2023] studied polar cap patches during days of high and low geomagnetic activity levels using Swarm data from a 3.5 year period starting on 16th July 2014. Their findings indicated values of RODI, 1st and 2nd order scaling exponents and intermittency being reduced outside the polar cap patches. These differences were more prominent at higher geomagnetic activity levels. Moreover, approximately 57.4% of GNSS loss of lock in the northern hemisphere during periods of elevated geomagnetic activity coincided with polar cap patches, while the amount was 51.4% during quiet periods. [Lovati et al., 2023] used 1 Hz Swarm data collected between 15th July 2015 and 31st December 2021 to investigate the dependence on season, solar activity and geomagnetic activity. They concluded that loss of lock events and electron density fluctuations increased during heightened solar activity and geomagnetic levels. These cases were associated with plasma density irregularities.

Previous literature suggest a difference between the polar cap and auroral region e.g. [Spicher et al., 2015]. Thus we decided to investigate both of these regions, hoping to gain some insights into more of their differences and similarities. Furthermore, we separate the days into two categories: days of high geomagnetic activity and days of low geomagnetic activity. This is due to the high latitude ionospheric irregularities being influenced by the geomagnetic activity [Jin et al., 2019] and often studied in the context of storm events [De Michelis et al., 2020][[Mitchell et al., 2005] and refs therein].
Consequently we want to investigate whether there are any changes between these days, as [De Michelis et al., 2020] observed. Since we want to investigate days of high geomagnetic activity, we chose periods during solar maximum i.e. the years 2014 and 2015. We chose the winter months during these two years due polar cap patches being a winter phenomenon [[Noja et al.,] and refs therein].

We present this thesis in 7 chapters and an appendix. Chapter 2 presents the essential theory required for the project. Chapter 3 outlines the data collection, and how we prepare it for further processing. Chapter 4 describes how each tool presented in Chapter 2 is applied. In Chapter 5 we cover the results of the work, while Chapter 6 discusses these findings in the context of the existing literature. Finally, Chapter 7 summarizes the findings, suggesting future work and mentioning acknowledgments.

2

Theory

We introduce some essential theoretical aspects in this chapter. We begin by giving an explanation of the high latitude ionosphere and its plasma structuring followed by a section dedicated to each of the data analysis tools used.

2.1 High Latitude Ionosphere and Plasma Structuring

The ionosphere is the region in Earth's atmosphere where ionised particles are created when high energy radiation from the sun interacts with the atmosphere. These particles form a layer of plasma at which is divided into the D Region, E Region and F Region. The E Region is located at altitudes in the range 90 - 150 km while the D - and F Regions are located below 90 km and above 150 km, respectively [Anderson, 1999].

As the solar wind interacts with the magnetosphere, high energy particles precipitate down along the magnetic field lines into the atmosphere, producing ionization. In the high latitude region the magnetic field lines in the polar cap are open, while they are closed in the auroral regions.

The plasma density is affected by the recombination rate, which is the rate at which ions and free electrons combine to form neutral particles. This factor varies with altitude, due to the distribution of different ions in the individual regions. The density is highest in the F Region where atomic oxygen O_+ dominates, while molecular oxygen O_2 and nitrogen N_2 are more common at lower altitudes [Kelley, 2009].

The plasma's position is not static. It moves due to the $\mathbf{E} \times \mathbf{B}$ drift perpendicular to the electric and magnetic fields. This drift, wave wave interaction and cascading [[Wernik et al., 2003] and refs therein] can cause plasma structures of varying sizes in the ionosphere. The F-region can have structures from scale lengths of 100 km to a couple of meters [Kintner and Seyler, 1985].

One of the main aspects of the high-latitude ionospheric region is its perpendicular magnetic field lines. In the polar cap, where the field lines are open, the interplanetary magnetic field have a strong influence [Moen, Jøran et al., 2013] [Wernik et al., 2003]. If the IMF have a southward component, we get a two-cell convection pattern which produces an anti-sun ward flow in this region [Kelley, 2009]. In other words, structures in the polar cap drifts from day to night [Moen, Jøran et al., 2013]. Large structures in the polar cap are called polar cap patches and can have densities up to 10 times larger than the background density[Moen, Jøran et al., 2013]. The electric fields of magnetospheric origin play an important role in the structure of plasma at high latitudes [Kelley, 2009]. Due to these field being perpendicular in the high-latitude ionosphere, plasma is transported to regions such as the polar cap during winter months. [Kelley, 2009]. In contrast, regions with closed magnetic field lines are susceptible to an increase in plasma production due to precipitation.

2.2 Structure Function

We introduce the structure function in this section, which is a method used to detect if the scaling behaviour at particular time scales change [Spicher et al., 2015], i.e. if the structure function is scale invariant, which may be a sign of intermittency [Bruno et al., 2001]. We begin by explaining the structure function, before moving on to why it can be used to characterise intermittency. Considering a set of ionospheric plasma density data $y(t)$, we can observe the scaling behaviour, and in turn intermittency by using an mth order structure function [Monin and Yaglom, 1975]

$$S(m,\tau) = \langle |y(t+\tau) - y(t)|^m \rangle \tag{2.1}$$

as defined in [Spicher et al., 2015]. Here y(t) represents the plasma density at time t while τ represents a time delay corresponding to $t + \tau$. The structure function is calculated for n amount of t for each τ where $\langle ... \rangle$ represents the average of all t for one unique τ. We use the average to make a distinction from the time average [Frisch, 1995]. Due to the non-negative nature of the structure function and the assumption that a point have a greater displacement from its initial position the further in time we go, the values of the structure function rises as τ increases.

We can analyse scale invariance either globally, or at different scales. In a case without any intermittency we expect the structure function be scale invariant, following a power law

$$S(m,\tau) \sim \tau^{\frac{m}{3}} \tag{2.2}$$

deviation from this can be a sign of intermittency, where we expect stronger intermittency for larger deviations [Bruno et al., 2001]. Therefore we predict that measurements taken during conditions where we expect high intermittency to diverge further from this power law than measurements where we expect lower intermittency.

An example of the structure functions and their corresponding power laws during two days of high and low geomagnetic activity is presented in Figure 2.1 in a log-log plot. The high activity day is on 31st December 2015 and the low activity day is on 30th December 2015. The figure illustrates equation 2.1 for $m = 2$ calculated in the entire polar region. The power law in equation 2.2 for m=2 is also fitted to the structure functions as dashed lines. The y-axis displays order of magnitude. We see both structure functions deviating from the power law, and a decrease in output values as τ increases. Larger τ leads to a smaller amount of data points used when calculating the average.

We also present the corresponding electron density measurements in Figure 2.2. We observe the electron density in the polar region during a high geomagnetic activity event in Figure a), while figure b) exhibits the corresponding low activity event.

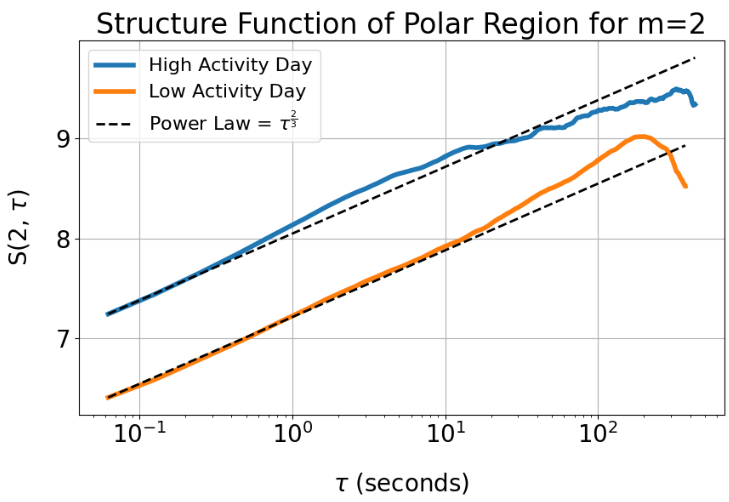


Figure 2.1: Structure function of the entire polar region during high geomagnetic activity (blue figure) and low geomagnetic activity (orange figure) 31st and 30th December 2015, respectively. Power law is fitted to each figure (dashed lines).

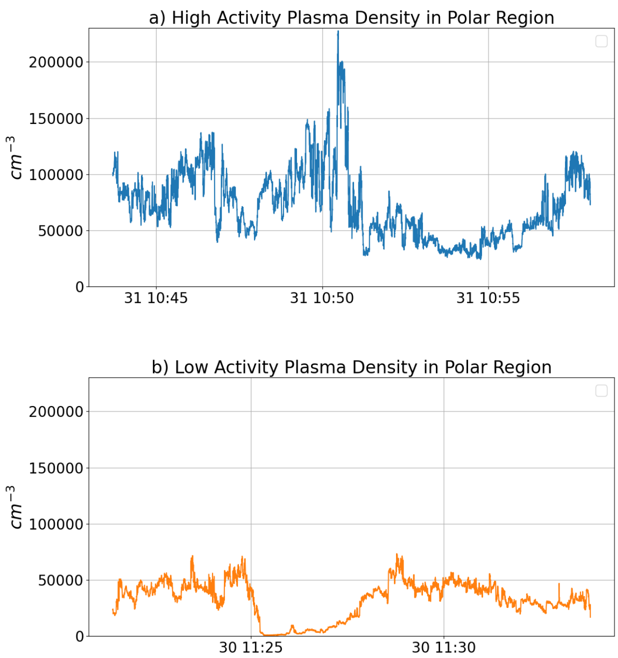


Figure 2.2: Corresponding electron density in the polar region in (a) a period of high geomagnetic activity and (b) low geomagnetic activity.

2.3 Empirical Flatness

We introduce another way of detecting intermittency, which is by calculating the empirical flatness ([Spicher et al., 2015], [Sahraoui, 2008], [Frisch, 1995])

$$F(\tau) = \frac{S(4,\tau)}{S^2(2,\tau)} \tag{2.3}$$

which is the ratio between the structure function for $m = 4$ and $m = 2$ squared. The empirical flatness, also called kurtosis, is expected to increase in cases of intermittency, especially for smaller scales [Chian et al., 2008]. Consequently we expect $F(\tau)$ to have a smaller value close to 3 when intermittency is low [Spicher et al., 2015]. We refer to the different τ intervals as "scales" on account of their representation of different time resolutions. These resolutions are used to examine the spatial variability of plasma structures.

Figure 2.3 shows the empirical flatness for the same cases as the structure function in the previous section. High activity day 31st December 2015 and low activity day on 30th December 2015. The dashed line represents $F(\tau) = 3$. We can see how the flatness increases for smaller τ, while larger scales have corresponding values close to 3. The high activity day has larger flatness than the low activity day, which is not always the case. We will elaborate further in chapters 5 and 6.

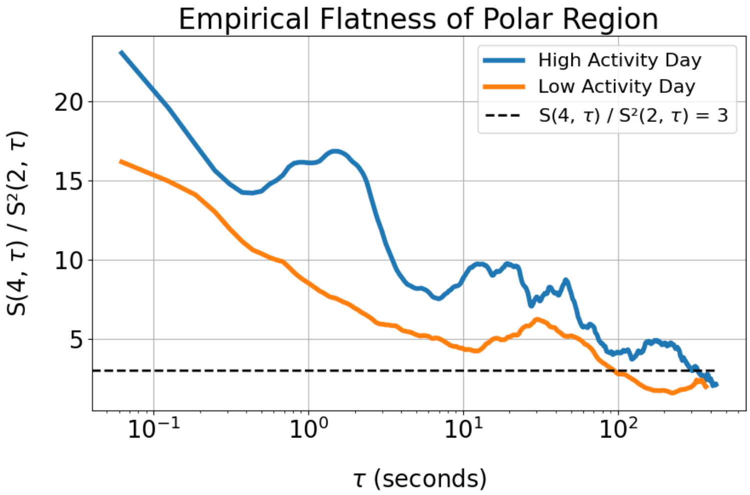


Figure 2.3: Empirical Flatness of the entire polar region during high geomagnetic activity (blue figure) and low geomagnetic activity (orange figure) 31st and 30th December 2015, respectively. The dashed line represents $F(\tau) = 3$.

2.4 Probability Density Function

Here we present another method of identifying intermittency at smaller scales, the probability density function (PDF). The method examines the density variations in time which provides information on the dynamics within the system at different scales. The variations at these scales are observed by investigating Non-Gaussianity of the signal's probability density function [Consolini et al., 2020], which we define as

$$PDF = y(t + \tau) - y(t) \tag{2.4}$$

where y(t) represents the plasma density at time t and τ represents Δt. A departure from a Gaussian curve could be signs of intermittent structures. Higher kurtosis or heavy tails and sharper peaks correspond to greater irregularities [Chian et al., 2008]. We may see larger fluctuations at smaller scales due to the intermittency similar to the empirical flatness.

We show an example of the probability density functions for a case on 31st December 2015 in Figure 2.4, where we normalized to the standard deviation as in [Consolini et al., 2020]. The PDFs displays the functions for $\tau = [1s, 10s, 100s]$ and a Gaussian fitted to the total PDF. The PDFs are calculated throughout the entire polar region. The PDF moves closer to the Gaussian as τ increases, with smoother peaks and tails not extending as far out to the side. Consequently as the scale decreases, the peaks become sharper and the tails extend further out, deviating from the superposed Gaussian. This implicates a dependence of time scale, that is PDFs are not scale invariant [Consolini et al., 2020].

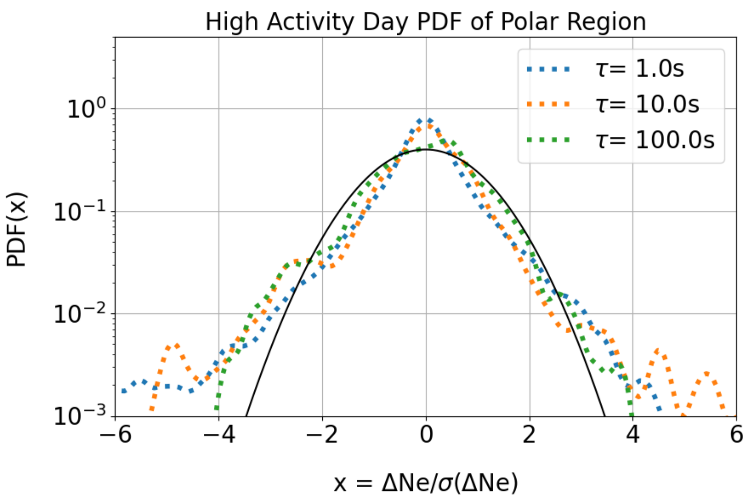


Figure 2.4: PDF of the entire polar region during high geomagnetic activity for three time lags. $\tau = 1.0s$ (dotted blue), $\tau = 10.0s$ (dotted orange) and $\tau = 100.0s$ (dotted green). A Gaussian (solid black) is fitted to the figure.

3

Data Acquisition

In the following section, we introduce the methods used for obtaining and pre processing the data used. We also elaborate on the sources used for acquiring our data, and present some background information on the instruments used for the data collection.

The data used have been obtained from the European Space Agency's Swarm Mission [ESA, a]. There are three identical satellites named Alpha, Bravo, and Charlie, that make up the Swarm constellation. The constellation follows a near-polar orbit with Bravo orbiting at an initial altitude of 511 km while Alpha and Charlie orbits at a lower initial altitude of 462 km, side by side. The mission's primary goal is to study Earth's magnetic field along with the electric currents in the magnetosphere and ionosphere [ESA, a] [ESA, b].

Measurements made from each satellite can be acquired through ESA's Swarm Data Access website [The European Space Agency,], granting access to both level 1b and level 2 packages. Level 1b contain calibrated data closely related to the direct measurements made by the satellites while level 2 contain additional derived data. One can obtain specific data such as different plasma measurements or magnetic measurements through the advanced section in the database in either in 2 Hz or 16 Hz resolution. The data is provided under regulation by ESA's Data Policy and terms and conditions [ESA, c].

The mission was first launched on 22nd November 2013 from Plesetsk Cosmodrome in northern Russia and initially established constellation on 17th April 2014. At the writing of this thesis, the Swarm mission is planned to last until 2025 [ESA, a].

The measurements were chosen from Swarm A. The data was downloaded from the website during spring 2024. Of the accessible data we used the measurements of the field-aligned currents at 1 Hz and the plasma density at 16 Hz sampling rate. Previous work such as [De Michelis et al., 2020] and [Consolini et al., 2020] have used density measurements from the Swarm constellation at 1 Hz and 2 Hz. By using 16 Hz we can investigate the structure function at smaller scales and compare with the results from previously published literature. The field-aligned current data was found under the level2daily section, and the plasma data was found under the advanced section. While there is not as much data at 16 Hz as there is at 2 Hz, we went with the higher sampling frequency to see if there were any important differences. We could not find any data for the field-aligned current at 16 Hz however. Both files are acquired as .cdf-files with the field-aligned current and plasma density given in units of $\mu A/m^2$ and cm^{-3} respectively in addition to other relevant data data such as latitude, longitude and timestamp.

We chose days in the winter months during the years 2014 and 2015. This was due to plasma convection playing an important role during the winter months [Wernik et al., 2003], and prevalence of polar cap patches [Dandekar and Bullett, 1999] which relate to high latitude ionospheric polar cap scintillations [Moen, Jøran et al., 2013]. The years were chosen due to the solar maximum, resulting in more days of high geomagnetic activity. When deciding which days to select within the target period, we initially wanted to investigate whether any papers had done any research on particular days with high geomagnetic activity which stood out, like St. Patrick's day on 17th March 2015 [Consolini et al., 2020][De Michelis et al., 2020]. We would be able to compare our results to what others have already found using the same examples by deciding on data using this method. However, this proved challenging as we were unable to find any 16 Hz data for these days. Instead we utilized data from the World Data Center for Geomagnetism, Kyoto's provisional AE index [WDC, b] which follows the International Council for Science - World Data System (ICSU-WDS) data sharing principles [World Data System,]. We used the service from the provisional AE index to find days with high geomagnetic activity based on the provided data from the website. The data used was accessed spring 2024.

We also obtain days of low geomagnetic activity. The days of low geomagnetic activity were found using the same method. They were chosen during the same month as the high activity days, preferably the day before or after. Due to many high activity days in a row, this was not always possible.

Figure 3.1 shows the auroral electrojet (AE) activity during an active day and a quiet day. The strength of the auroral electrojet is associated with geomagnetic activity [Davis and Sugiura, 1966] [Akasofu et al., 1965]. By tracking changes in the AE index, we can detect geomagnetic disturbances, like geomagnetic storms. Figure 3.1a shows the AE activity on 4th November 2014 and Figure 3.1b on 3rd November 2014 as displayed on the World Data Center for Geomagnetism website. In Figure 3.1a the activity is extremely low early in the day up to 07:00, indicated by the flat activity level. From 07:00 and up to 18:00 the AE activity rises drastically, with the largest spikes at 11:00 and 14:30 surpassing 1000 nT. These are the types of activity levels we look for when deciding days of high geomagnetic activity. In contrast, Figure 3.1b displays no signs of an elevated AE intensity. When searching for days of high geomagnetic activity, we look for events surpassing approximately 800 nT. In the end, our high activity data had events in the range from 800 nT to 2000 nT. All days used are listed in the Appendix.

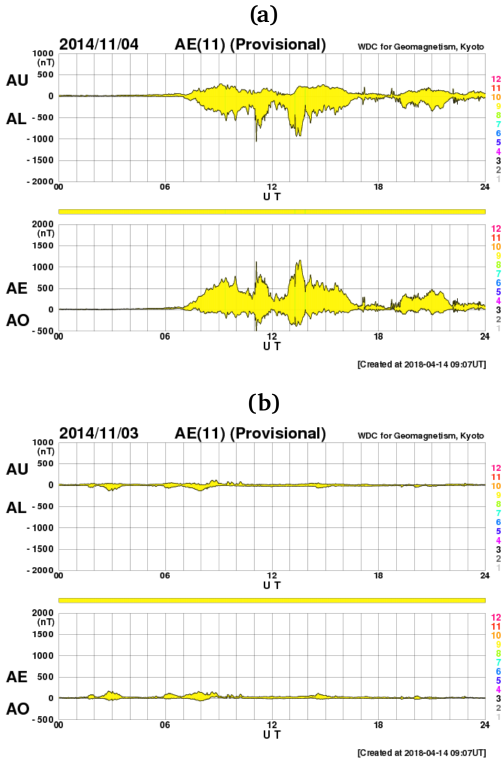


Figure 3.1: Figure 3.1a) shows high geomagnetic activity measured 4th November 2014, with the most active period being from approximately 06:00 to 18:00. Figure 3.1b) shows low geomagnetic activity measured 3rd November 2014 [WDC, b].

We compare data obtained during periods of different geomagnetic activity. When investigating the high activity and the low activity days we chose the time during the quiet day based on the satellite's position rather than its closeness in time to the active day. This is due to the satellite not passing over the same region at the same time each day. We want the trajectories where the satellite is as close as possible to minimize the influence of other factors, and ensure similar environmental conditions.

The nearest location was found using the latitude and longitude of the Swarm A satellite when it passed over the high activity region, comparing this to the closest latitudes and longitudes during the low activity day. This is displayed in Figure 3.2. where the purple line show the satellites trajectory during the day of high activity. The grey lines are the satellite's trajectory for each pass over the north pole during days of low activity. The black line is the closest of the trajectories during the this period, calculated by the script. We can see it is indeed the closest trajectory to the high activity trajectory.

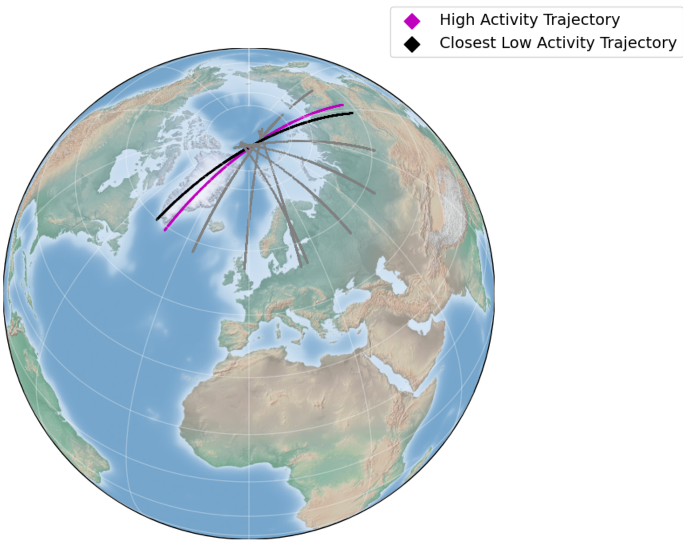


Figure 3.2: The purple line represents the trajectory of the satellite during the high activity day 4th November 2014 while measuring the polar region. The grey and black lines are the trajectories for each pass through the polar region of the satellite during the low activity day. The black line is the closest of these passes to the high activity day.

Swarm A makes multiple passes through the polar region each day but each pass does not contain equally interesting data. Thus we need to choose instances during each day which we want to investigate further. We started by observing when the geomagnetic activity was highest during the period in which the satellite's 16 Hz faceplate plasma density measuring device was active. Furthermore, we looked at the intensity of the field-aligned current and plasma density during this period when deciding passes. Ultimately, we decided to focus on three different times each day, which we have labeled as instances 1, 2, and 3 and each occurring in the northern polar region. In the early stages, we also analysed the southern pole, but due to time constraints, we left this out for future work.

We divide the polar region into 3 different sub regions; A, B, and C. As the satellite passes through the polar region we typically see two consecutive spikes in the field-aligned current density. The location of these spikes are associated with the auroral oval [Iijima and Potemra, 1976]. The first spike is classified as region A and is where the satellite passes through the first part of the auroral region. Region B is the region between the two spikes where the satellite passes through the polar cap. The second spike is where the satellite passes through the opposite site of the auroral region and is classified as region C. We chose to investigate both the polar cap and auroral region to inspect the differences and similarities when applying the same tools to the regions.

We developed our own algorithm for classifying the different regions. The method uses the field-aligned currents to get an estimate of where the auroral region is in the polar region during the plasma density measurements. We detect when the the satellite passes through the aurora by looking for these spikes in the field-aligned currents. During the spikes, we utilize the timestamps to determine the corresponding plasma density. The spikes are detected by setting a threshold value for the field-aligned current, classifying each element higher than the threshold as part of the auroral region. The threshold value was in most cases set as $0.5\mu A/m^2$, but this was either increased or decreased if our algorithm incorrectly identified the FAC. When two spikes are more than 2 minutes apart, they are separated into different regions. This limit can be adjusted if necessary. Each spike is numbered from 1 to n. When the user want to investigate a specific region (Region A or C), the corresponding number in this range is used. If the user want to investigate the entire polar region, a tuple can be used as input, where each of its values correspond to either of two side by side spikes. To specify the polar cap (Region B), an additional boolean is used along with the tuple to convey that the user wants the region between the spikes. This way the algorithm created can detect either all polar regions, one polar region, or a sub region A, B or C. Figure 3.3 presents a polar region divided into the auroral region and polar cap by the algorithm. Figure a) presents the plasma density measured in 16 Hz by Swarm A and Figure 3.3 b) show Swarm A's measurement of the field-aligned currents in 1 Hz. The chosen case is highlighted in orange, red and green for the auroral region (Region A and C) and the polar cap (Region B), respectively. We can see an increase in electron density in a) at the same time as the field-aligned current rises in b).the

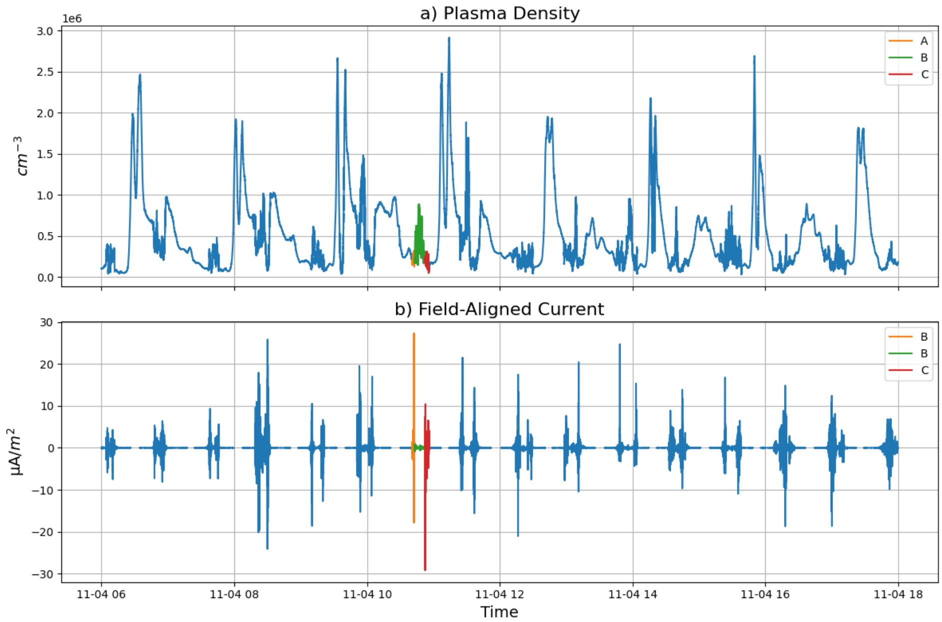


Figure 3.3: Measured data from Swarm satellite A during 4th November 2014. Data shown from 06:00 UTC to 16:00 UTC during the period with the highest geomagnetic activity. At approximately 11:00 UTC we highlight a high field-aligned current and plasma density, which corresponds to the measurements seen in Figure 3.1a). The marked data was chosen as instance 1 during this day.

The data we are left with is shown in Figure 3.4. We have focused on the designated region shown in Figure 3.3. Figure a) and b) show the plasma density and field-aligned currents, respectively. The plasma density increases as Swarm A enters the polar cap, while there are no field-aligned currents due to the open magnetic field lines in this region.

For comparison, we display the plasma density and field-aligned currents for the corresponding low activity day on 3rd November 2014 in Figure 3.5. Both Figure a) and b) uses the same limits for the axes as Figure 3.4 a) and b), respectively.

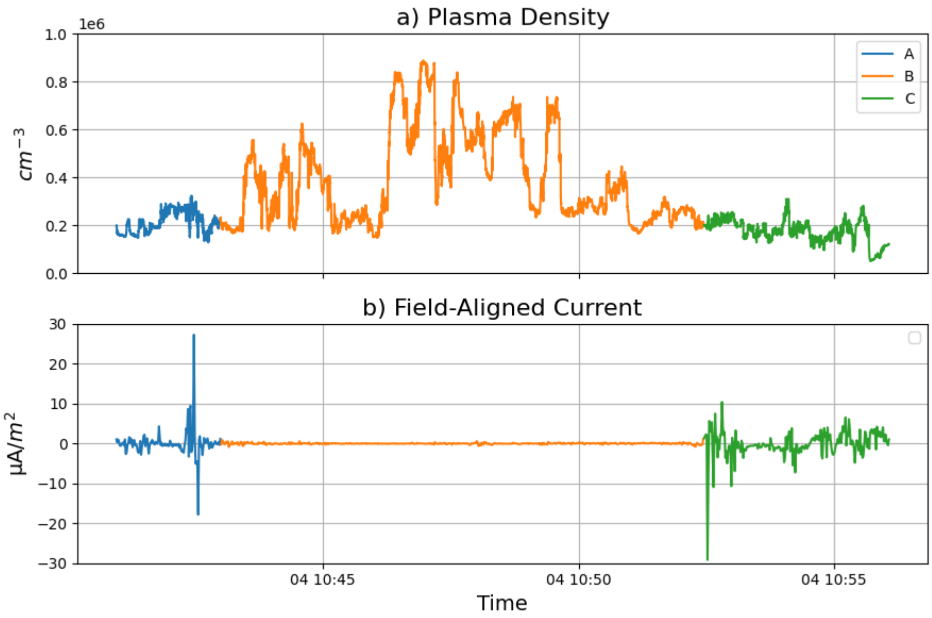


Figure 3.4: Instance 1 during 4th November 2014. The blue plots are of the plasma density and field-aligned current in Region A, while the orange and green are in Region B and C respectively.

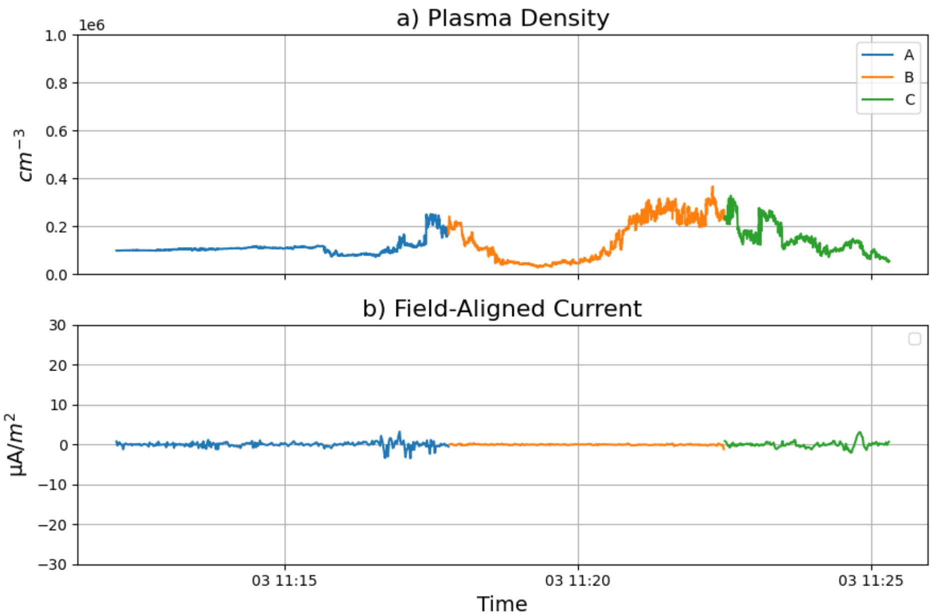


Figure 3.5: Low activity day on 3rd November 2014. The blue plots are of the plasma density and field-aligned current in Region A, while the orange and green are in Region B and C respectively.

In this work we apply different processing methods to try and detect differences between the high geomagnetic activity days and low geomagnetic activity days. We ended up with 13 days of high activity and 8 days of low activity. When we had to compare various high activity days to the same low activity days, we attempted to compare them at different times of the day to prevent duplicating data from low activity periods. In total we ended up with 39 distinct events during high activity periods and 31 distinct events during the low activity period.

4

Methods

In this chapter, each method is described within its dedicated section. We first review the implementation of the structure function followed by the procedures of its further processing in section 4.1. In section 4.2, we explain the methods behind the empirical flatness. Lastly we explain how we implemented the probability density function in section 4.3. Each method is implemented using the programming language Python, or more explicitly Python 3.10. Occurrences where modules have been used for specific calculations are referenced where applicable.

4.1 Structure Function

This section presents how we approached implementing the structure function for our processed data. There are multiple ways of doing this, and we demonstrate the two methods we tested. We also explain the different procedures used to further analyse the results.

Initially a completely vectored version of the structure function was implemented, resulting in a very fast runtime. However, this had some limitations. This implementation of the structure function always averages over the same amount of data points in an interval. This amount is defined as the signal's length subtracted by the maximum time lag τ_{max}. If the largest time lag is chosen at half the signal's length, the average is always be computed for half the data points. This results in the amount of data point used decided by τ_{max}.

The disadvantages of this solution, hereafter referred to as method A, was determined to outweigh the advantages. Instead a second method, referred to as method B, could calculate the structure function for the entire signal's length. This method calculated the mean of as many data points possible for each time delay, resulting in the amount of data points averaged decreasing by 1 for each increase in τ (see Figure 4.1. The drawback to this solution was that the code was significantly slower, using approximately 24 hours to calculate the structure function for the 39 active and 39 quiet cases.

We illustrate the differences in Figure 4.1, which shows how many data points the average can be calculated over for each of the two methods. The figure represents the amount of data points (y-axis) used in the structure function for each time lag τ (x-axis). The figure presents a dataset consisting of 600 data points. Method A denoted by the blue line, is shown for a time delay interval $\tau = [0s, 300s]$ while method B denoted by the orange line is shown for $\tau = [0s, 600s]$. Method A have a constant number of averaged data points, regardless of τ, while method B is more dynamic, calculating the maximum number of available data points for each τ. At time lag 600s, i.e. the length of the signal, no amount of data points are used as we cannot use data at time 601s.

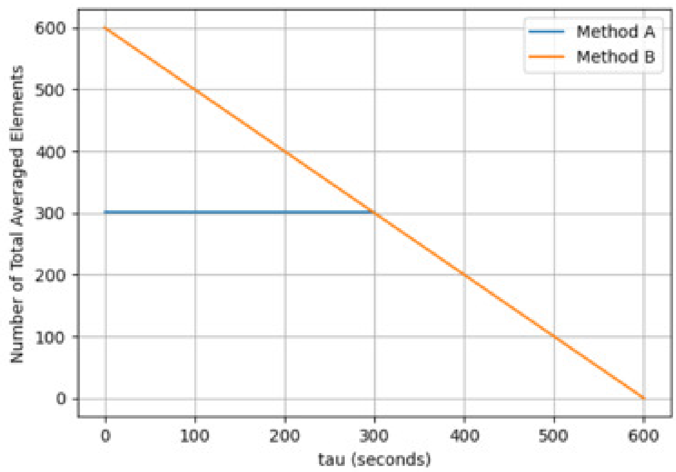


Figure 4.1: 600 seconds example dataset. The blue line represents method A, and the orange line represents method B. The y-axis represents the amount of data points averaged. The x-axis represents the time lag in seconds.

We decided to use method B due to the increased amount of data points for smaller τ. Due to the accuracy of the calculations decreasing for larger τ, we chose to use a maximum time delay of half the signal's maximum length. This made sure we always calculated the average of at least half of the data points.

We calculated the structure function for each instance each day with a time delay interval [1 / 16 s, n] where n is half the signal's length in seconds. We start at 1/16 seconds due to the sampling rate of the signal being 16 Hz. The structure functions are stored in multiple .csv-files as data frames for faster acquisition. The structure function was calculated for both $m = 2$ and $m = 4$ (see equation 2.1). At higher orders than 4, the structure results become unreliable [Horbury and Balogh, 1997]. We chose to only calculate the structure function for orders $m = 2$ and $m = 4$ due to our large amount of data and the computational time required to generate the result. We also need $m = 2$ and $m = 4$ when calculating the empirical flatness.

We opted for a maximum tau of 100 seconds when examining the structure functions despite the signal being longer in many instances. This is due to a reduced accuracy as τ increases due to less averaged points, as well as the limited duration of some structure functions which do not extend much beyond 100 seconds. When we divide the polar region into Region A, B and C we end up with cases where regions do not last long enough to calculate the structure function for $\tau = 100s$. This is most common for Region A and C. We wanted to avoid restricting the maximum time lag to less than $100s$ in sufficiently long datasets. For shorter datasets, we computed the structure function up to the maximum attainable time lag. As an example, in a dataset lasting $160s$, we set the maximum time lag τ to $80s$. We justify this due to the final result not getting significantly affected the shorter signals. In addition, the average length for the structure function in the two shortest regions are approximately 133s and 119s for Region A and C, respectively. For completeness we mention that the average length in region B is 205s and the entire polar region is 455s. Note that the sum of the lengths of Region A, B and C is 457s, we assume the 2 second difference is due to rounding errors in Python.

Furthermore, we wanted to determine which of the two structure functions had the highest output values at the intervals [1s, 10s] and [10s, 100s] when comparing the high activity and low activity days. This was done to check if we could observe a trend, i.e. higher values for days of high geomagnetic activity. To do this we used the Simpson integration rule [SciPy, a] to estimate the area the curves.

Additionally we calculated the slope of the structure function using scipy's linear regression [SciPy, b]. This was done for τ in the ranges $[1s, 10s]$ and

$[10s, 100s]$ resulting in 4 different data sets. We investigated the range $\tau = [0.0625s, 1s]$ to observe if there are any differences at smaller scales by utilizing our 16 Hz data. However, our primary focus was on $\tau = [1s, 10s]$ and $\tau = [10s, 100s]$ due to difficulties interpreting the data, as we have very few data points at such small timescales. A figure covering the slopes at $\tau = [0.0625s, 1s]$ can be found in the appendix. These data sets were categorized into two groups: one for high activity days and one for low activity days, for each of the τ intervals. Initially both the high and low activity data sets had 39 values each, for the 39 different cases. Due to not having as many days of low geomagnetic activity as of high geomagnetic activity, some duplicate data was present. To avoid using the same data more than once we removed the duplicates in the data sets for the low activity days. This resulted in 31 unique slopes.

At first we used a kernel density estimation on the histograms which yields better results for large data sets, hoping to gain more details about our data. In the end we decided that due to not having more than at maximum 39 data points for the high activity day and 31 for the low activity day, we would not gain any more information than by displaying the results in histograms.

We also plotted the ratio of the structure functions in each region, that is: Region B/A, Region B/C and Region A/C. We did not have time to analyse these results, however.

4.2 Empirical Flatness

We describe the method behind the implementation of the empirical flatness, and how we further used these computations in the following section.

Since both the structure function for $m = 2$ and $m = 4$ is calculated at this stage, we applied these calculations when determining empirical flatness to save on time and computational power. This method results in a time series of equal length to the aforementioned structure function e.g. an empirical flatness in the interval [1/16s, n] where n is half the length of the signal. The empirical flatness was calculated for the entire polar region, both auroral regions (Region A and C) and the polar cap (Region B) in between these two auroral regions.

Moreover, we divide the empirical flatness into two intervals $\tau = [1s, 10s]$ and $\tau = [10s, 100s]$ instead of three as we did for the structure function. We omit the smallest interval as we could not observe any meaningful difference between those using $\tau = [1s, 10s]$. Because large values often occur when τ is small, we calculate the area of each flatness curve using the same method as for the structure function possibly observing intermittent characteristics.

4.3 Probability Density Function

The implementation of the probability density function is described in this segment. We also highlight the tools used to get our result.

The probability density function $n_e(t + \Delta t) - n_e(t)$ was calculated for multiple increments of τ. $\tau = [.0625s, .125s, .25s, .5s, 1s]$, at extremely small scales $\tau = [1s, 5s, 10s]$ at small scales and $\tau = [10s, 50s, 100s]$ at larger scales. We decided to use extremely small scales to make better use of our 16 Hz sampling rate. The results were normalized to the standard deviation following the same methods as in [Consolini et al., 2020]. We divided the increments into three intervals. The first from 0.0625s to 1s to see if we can notice any difference in very small structures due to our high resolution data. The two latter time intervals following the same reasoning as our structure functions and empirical flatness. Especially the empirical flatness due to the correlation to kurtosis.

The probability density functions were calculated in the interval separately for the high and low activity days. We computed the PDFs in the entire polar region, and Region A, B and C. We fitted a Gaussian to the probability density functions. Its mean and standard deviation was calculated from the average of the PDFs mean and standard deviations for all increments to make the Non-Gaussianity more discernible.

We used a kernel density estimate (KDE) to produce the figures related to the PDFs. A KDE is comparable to a histogram, but in our case it produced a clearer visualisation of the PDFs, similar to results found in [Consolini et al., 2020]. The KDE was performed using the seaborn module's kdeplot [Seaborn,].

5

Results

We present our observations for each of the methods used. Each section consists of a specific method used which we presented in chapter 4. We follow the same structure as in the aforementioned chapter. Each section is divided into subsections. We first present the results covering the calculation throughout the entire polar region, before moving on to the polar cap, referred to as Region B. Region A and C are discussed lastly, due to both regions being located in the auroral region. Unless otherwise specified, each plot corresponding to different regions (e.g., structure function for each region) follows the same formatting and annotations.

Firstly in section 5.1, we introduce our results obtained by use of the structure function. We display some examples of the structure functions and slopes before moving on to the corresponding histograms where we display all our data for the slopes. We then show the histograms for the areas of the structure function curves and the structure functions' divergence from the power law.

We then move on to the results obtained from the empirical flatness in section 5.2 where we present figures for the empirical flatnesses and histograms corresponding to the areas under the curves.

Due to the preceding method's result sharing some similarities with the next one, section 5.3 display the results gained by calculating the PDFs. Due to the similarities between all figures, we only display examples of one instance for one day, which we will further elaborate in chapter 6.

5.1 Structure Function

For all regions we first present the structure functions calculated for $\tau = [\frac{1}{16}s, \frac{L}{2}s]$ where L is the length of the data set for a high and low geomagnetic activity day. We also add three dashed lines for each day representing the slopes at $\tau = [0.0625s, 1s]$, $\tau = [1s, 10s]$ and $\tau = [10s, 100s]$. This is to illustrate a typical structure function and slope for each of the cases. We then move on to the distribution of the slopes displayed in histograms. Furthermore we show the areas of each structure function before displaying the corresponding distributions in histograms.

5.1.1 Structure Function and Slope

Firstly we present the plasma density during days of high and low geomagnetic activity in Figure 5.1. Figure a) exhibits the plasma density for a period on 4th November 2014, the same instance that was used as an example in chapter 3. Figure b) shows the quiet day 3rd of November 2014 in a period where the satellite is as close in latitude and longitude as possible to our active period.

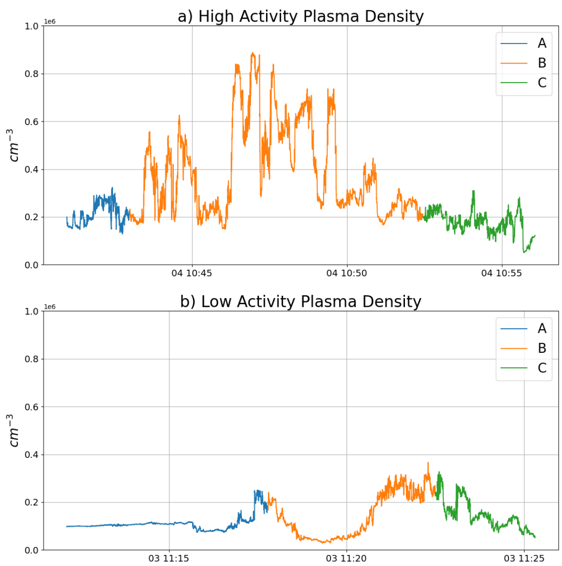


Figure 5.1: Plasma Density in the polar region in (a) a period of high geomagnetic activity 4th November 2014 and (b) low geomagnetic activity 3rd of November 2014. Each figure is divided into the auroral regions A (blue) and C (green), Region B the polar cap (orange). Both figures share tha same y-axis.

Figures 5.2, 5.3, 5.4 and 5.5 show the structure function in a log-log plot calculated for our active and quiet day. The dashed lines show the regression analysis for $\tau = [0.0625, 1s]$, $\tau = [1s, 10s]$ and $[10s, 100s]$ for both cases. The Y-axis displays order of magnitude.

In Figure 5.2 the structure function is calculated for the entire polar region. The structure function values during the high activity day are quite larger than during the low activity day. We also see a decrease in slope for this day, while it is actually slightly increasing for the low activity day as the scale increases. Beyond $\tau = 100s$ the structure functions become more uneven, and we have fewer data points when calculating the average leading to a less precise result.

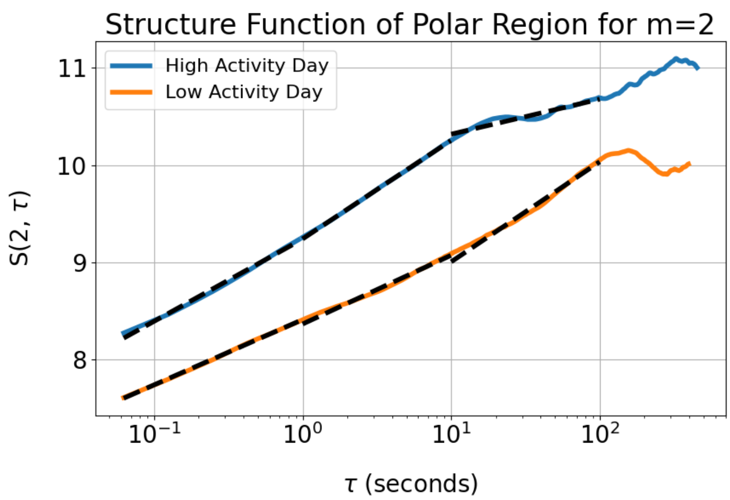


Figure 5.2: Structure function of order $m = 2$ in the polar region during a day of high geomagnetic activity (blue figure) vs a day of low geomagnetic activity (orange figure). The dashed lines represent the slopes.

Figure 5.3 shows the structure function calculated for the polar cap region. The structure function values during the high activity day are larger than during the low activity day, similar to the entire polar region. The main difference is the structure function during the high activity day being slightly higher in the polar cap. Furthermore, structure function for the low activity day is approximately an order of magnitude lower. The slopes also share a likeness to the slopes in the complete polar region. The structure function does not extend as far past $\tau = 100s$ due to the polar cap only being a component of the polar region.

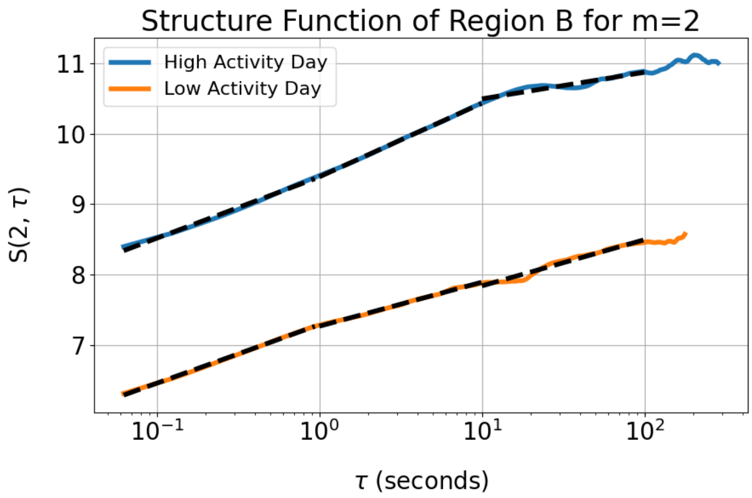


Figure 5.3: Structure function of order $m = 2$ in the polar cap region during a day of high geomagnetic activity (blue figure) vs a day of low geomagnetic activity (orange figure). The dashed lines represent the slopes.

Figure 5.4 shows the structure function calculated for region A, as Swarm A measures the auroral region before the polar cap. The structure function values during the high activity day are again larger than during the low activity day, but not quite as large as in the entire polar region nor the polar cap. We can see that other than the magnitudes, the structure functions seem to exhibit a behaviour opposite of what we see in the previous two examples. The slope during the high activity day is steeper than the slope during the quiet day at larger scales. The steepness' of the slopes are harder to distinguish, however. An important remark is that this will not always be the case, which becomes clearer when observing the histograms for the slopes. It should be noted that the structure function does not span the entire interval when $\tau = [10s, 100s]$, as the auroral region regions often are smaller than the polar cap.

We see the structure function calculated for region C in Figure 5.5 as Swarm A measures the auroral region after the polar cap. The structure function values during the high activity day are similar to those in region A. The values for the low activity day are a bit smaller than in region A however. The structure functions behave similar to the results in Figures 5.2 and 5.3, with a very flat slope from $10s$ to $100s$ during the high activity day. We make an important remark here as well, which is that this will not always be the case. It should rather be noted that the results can behave similar to what we see in region A. As in the aforementioned region, we do not always have measurements which are long enough to span the entire τ interval.

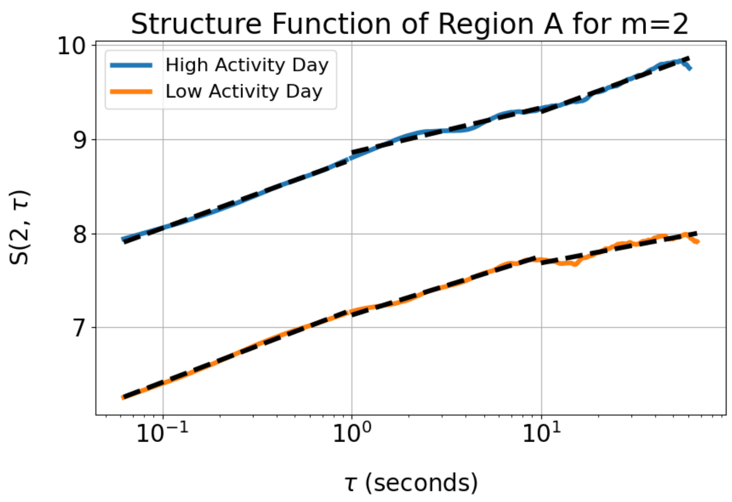


Figure 5.4: Structure function of order $m = 2$ in region A during a day of high geomagnetic activity (blue figure) vs a day of low geomagnetic activity (orange figure). The dashed lines represent the slopes.

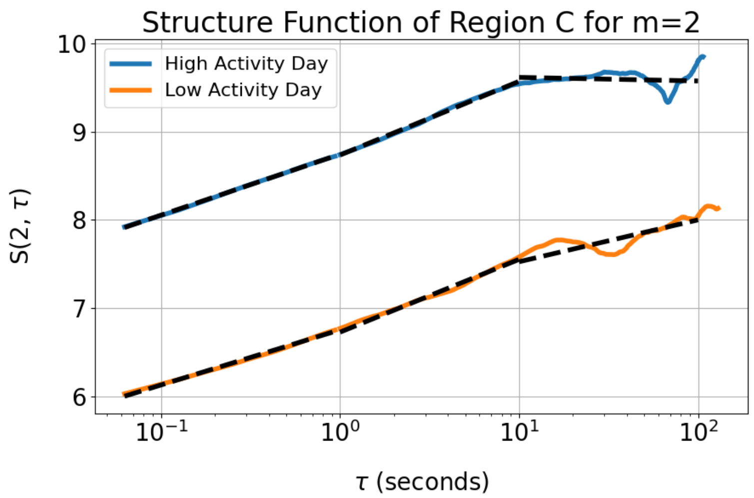


Figure 5.5: Structure function of order $m = 2$ in region C during a day of high geomagnetic activity (blue figure) vs a day of low geomagnetic activity (orange figure). The dashed lines represent the slopes.

5.1.2 Slope Distribution

Instead of showing 39 figures similar to those in the figures in the previous subsection, we instead add each result to histograms as shown in Figures 5.6, 5.7, 5.8 and 5.9. Here we can see the slope of all structure functions calculated in the entire polar region. Note that duplicate slopes produced for the same instances during days of low geomagnetic activity are removed, resulting in 31 slopes for these days in comparison to 39 slopes for days of high geomagnetic activity. Due to the similarity of the results for order $m = 2$ and $m = 4$ we decided to only show the figures for $m = 2$.

The slopes are displayed for $\tau = [1s, 10s]$ and $\tau = [10s, 100s]$. Figure a) shows $[1s, 10s]$ and $[10s, 100s]$ for days of high geomagnetic activity. Figure b) shows high and low activity days for the interval $[1s, 10s]$. Figure c) shows $[1s, 10s]$ and $[10s, 100s]$ for days of low geomagnetic activity. Lastly, Figure d) shows high and low activity days for the interval $[10s, 100s]$.

We present the slopes in the entire polar region in Figure 5.6. If we first take a look at Figure a) and c) we notice a clear difference. During days of high geomagnetic activity there is a trend of slopes decreasing in value at larger scales. During days of low geomagnetic activity a difference in slopes is much harder to notice. Moreover, the distinction between active and quiet days become more pronounced when inspecting Figures b) and d). The slopes during active days are generally steeper than quiet days at smaller scales and shallower at larger scales.

Figure 5.7 shows the slopes of days during high and low geomagnetic activity in the polar cap region. Figure a) and c) are quite similar to the results covering the complete polar region. The values of the slopes are spread over a wider range in this case, however. Figures b) and d) are alike the previous result, though the high and low activity days are more distinct, especially for $\tau = [10s, 100s]$.

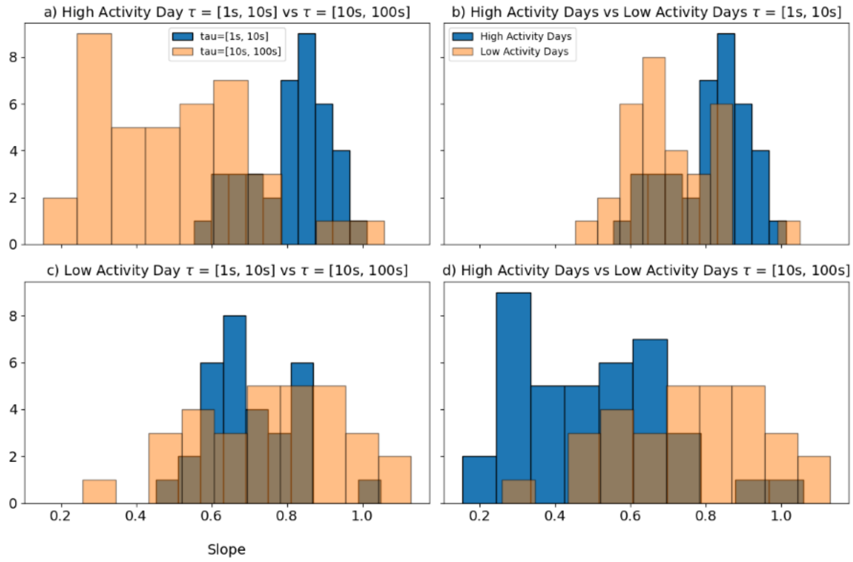


Figure 5.6: Slopes in polar region. (a) $\tau = [1s, 10s]$ (blue) $\tau = [10s, 100s]$ (orange) during active periods. (b) Active (blue) and quiet (orange) periods for $\tau = [1s, 10s]$. (c) $\tau = [1s, 10s]$ (blue) $\tau = [10s, 100s]$ (orange) during quiet periods. (d) Active (blue) and quiet (orange) periods for $\tau = [10s, 100s]$.

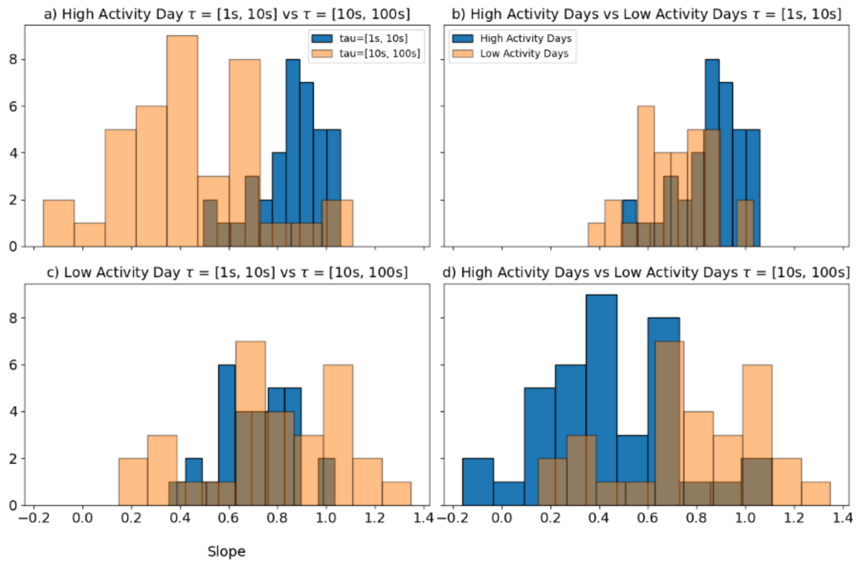


Figure 5.7: Slopes in polar cap region. (a) $\tau = [1s, 10s]$ (blue) $\tau = [10s, 100s]$ (orange) during active periods. (b) Active (blue) and quiet (orange) periods for $\tau = [1s, 10s]$. (c) $\tau = [1s, 10s]$ (blue) $\tau = [10s, 100s]$ (orange) during quiet periods. (d) Active (blue) and quiet (orange) periods for $\tau = [10s, 100s]$.

Figure 5.8 exhibit the slopes of days during high and low geomagnetic activity in the first section of the auroral region, region A. Noticing any distinguishing features is a bit more challenging than in the previous two results. In Figure a) the slopes are slightly smaller at higher scales. In Figure c), the slopes at large scales are spread more than the two previous regions, ranging from -0.5 to 1.2. It is challenging to notice any substantial difference between the days of high and low activity in Figures b) and d).

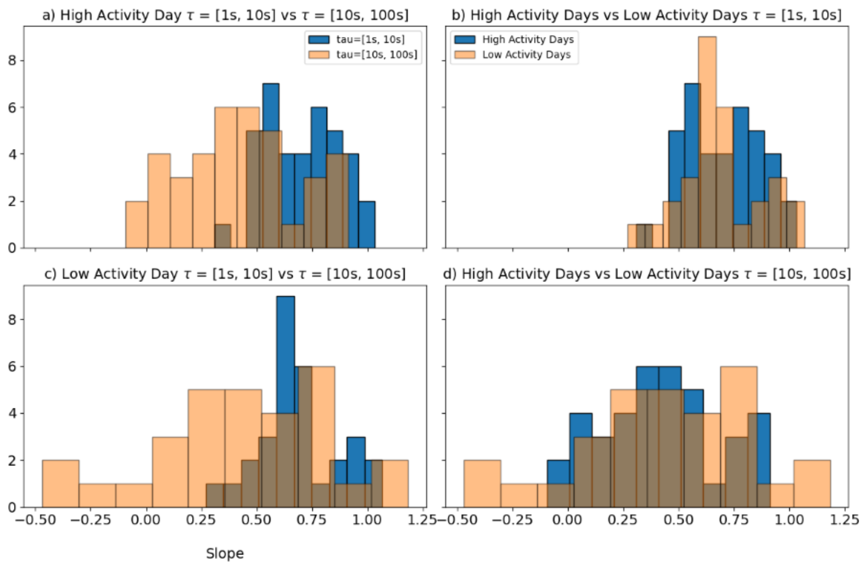


Figure 5.8: Slopes in Region A. (a) $\tau = [1s, 10s]$ (blue) $\tau = [10s, 100s]$ (orange) during active periods. (b) Active (blue) and quiet (orange) periods for $\tau = [1s, 10s]$. (c) $\tau = [1s, 10s]$ (blue) $\tau = [10s, 100s]$ (orange) during quiet periods. (d) Active (blue) and quiet (orange) periods for $\tau = [10s, 100s]$.

Lastly in Figure 5.9 we present the slopes of days during high and low geomagnetic activity in Region C, the second part of the auroral region as the satellite leaves the polar region. Noticing any differences in each histogram is even more challenging for these cases. In Figure a) and c) the slopes at large scales have wider distribution than at small scales. Otherwise, it is quite hard to distinguish the high and low activity days.

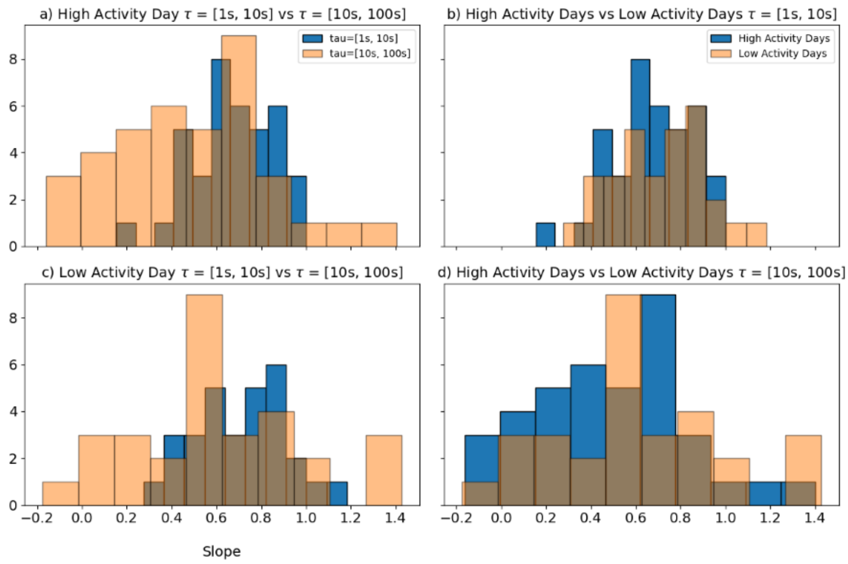


Figure 5.9: Slopes in Region C. (a) $\tau = [1s, 10s]$ (blue) $\tau = [10s, 100s]$ (orange) during active periods. (b) Active (blue) and quiet (orange) periods for $\tau = [1s, 10s]$. (c) $\tau = [1s, 10s]$ (blue) $\tau = [10s, 100s]$ (orange) during quiet periods. (d) Active (blue) and quiet (orange) periods for $\tau = [10s, 100s]$.

5.1.3 Area of Structure Function

Figures 5.10, 5.11, 5.12 and 5.13 display the area covered by the structure functions of order $m = 2$ for each instance, totaling 39. We do not remove the duplicates in these figures as we want to compare the results on a case-by-case basis. Figure a) and b) show $\tau = [1s, 10s]$ and $\tau = [10s, 100s]$ respectively when calculating the structure function. The blue plots are areas calculated during days of high geomagnetic activity while the orange plots are calculated ruing days of low activity. We do not present similar plots for $\tau = [0.0625s, 1s]$.

In the entire polar region exhibited in Figure 5.10, we can see that for most cases the high activity days have larger output values. While the area increases for all cases at larger scales, the relative difference between active and quiet cases do not have any noticeable change.

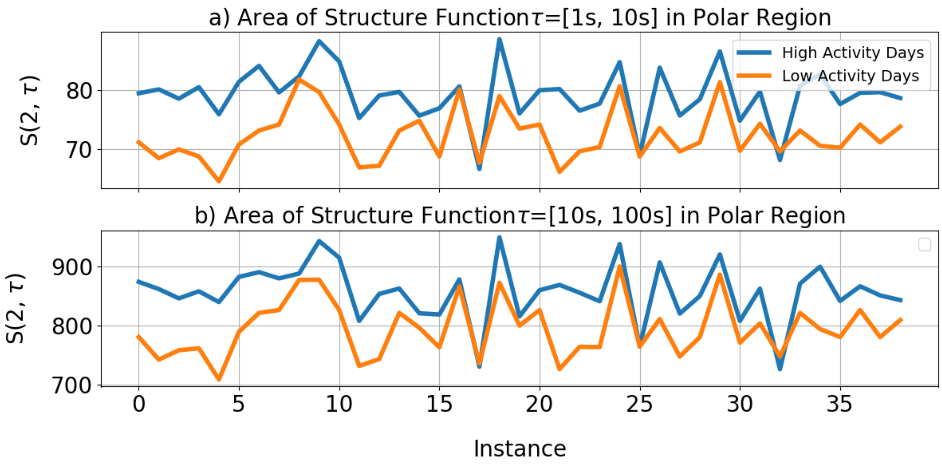


Figure 5.10: Area of structure function in the entire polar region. (a) Area for smaller structures $\tau = [1s, 10s]$ calculated for all instances. (b) Area for larger structures $\tau = [10s, 100s]$ calculated for all instances.

Figure 5.11 shows the area covered by the structure functions in the polar cap region. There are larger variations for some instances during days of low geomagnetic activity, especially instance 28. Otherwise the results are fairly similar to those presented in Figure 5.10

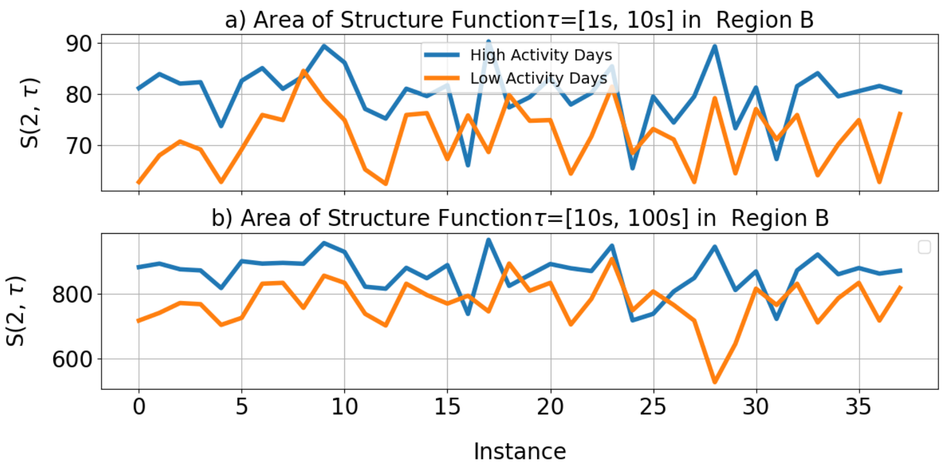


Figure 5.11: Area of structure function in the polar cap region. (a) Area for smaller structures $\tau = [1s, 10s]$ calculated for all instances. (b) Area for larger structures $\tau = [10s, 100s]$ calculated for all instances.

Furthermore we move on to the auroral regions. Figure 5.12 displays the area covered by the structure functions in region A. There is a similar trend to that for the polar cap. If we inspect the y-axis, the differences between structure functions that cover large areas and those that cover small areas are much larger at $\tau = [10s, 100s]$ than those in the entire polar region and polar cap at this scale.

Figure 5.13 show the area covered by the structure functions in region C. We see a smaller difference between active and quiet day areas at large scales than in the other regions.

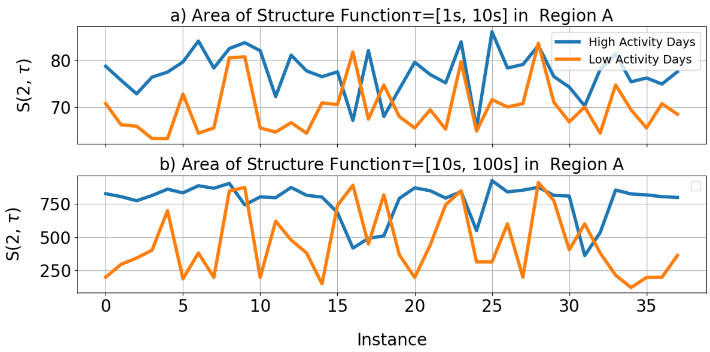


Figure 5.12: Area of structure function in region A. (a) Area for smaller structures $\tau = [1s, 10s]$ calculated for all instances. (b) Area for larger structures $\tau = [10s, 100s]$ calculated for all instances.

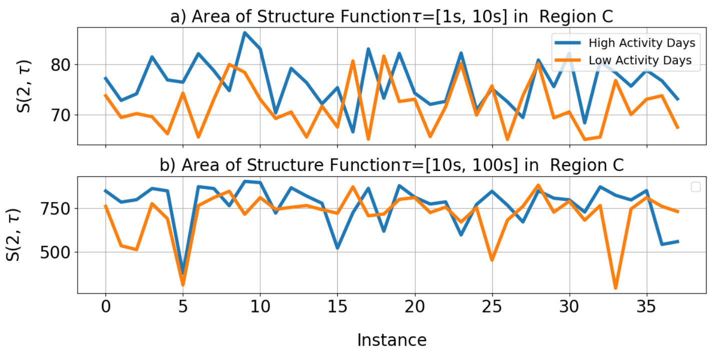


Figure 5.13: Area of structure function in region C. (a) Area for smaller structures $\tau = [1s, 10s]$ calculated for all instances. (b) Area for larger structures $\tau = [10s, 100s]$ calculated for all instances.

5.2 Empirical Flatness

All regions are presented in the same manner as in section 5.1. We first start with displaying the empirical flatness for all instances along with the corresponding area distribution. Additionally we show figures of the areas of each instance, similar to in subsection 5.1.3.

5.2.1 Empirical Flatness and Area Distribution

Figures 5.14, 5.15, 5.16 and 5.17 show the empirical flatness for all instances in a plot each for $\tau = [1s, 10s]$ (a) and $\tau = [10s, 100s]$ (b). The dashed line represents a flatness value $F(\tau) = 3$. Additionally, we provide two plots containing the distribution of the area covered by each empirical flatness for $\tau = [1s, 10s]$ (c) and $\tau = [10s, 100s]$ (d).

Figure 5.14 presents the empirical flatness calculated in the entire polar region for $\tau = [1s, 10s]$ and $\tau = [10s, 100s]$ in Figure a) and b), respectively. Figures c) and d) show the corresponding distribution of areas in histograms. We can see larger values for the output during low activity days. We will discuss this further in chapter 6. We can also see how the values decrease for all instances of empirical flatness as the scale increases, which is in agreement with the expected result.

Figure 5.15 exhibits the empirical flatness calculated in the polar cap region. We can see a similar result as in Figure 5.14 with some slight change. We have a smaller quantity of larger values for the output during low activity days. Consequently we see a larger amount of high activity days with larger output values than the low activity days. The values decrease for all instances of empirical flatness as the scale increases in this case as well.

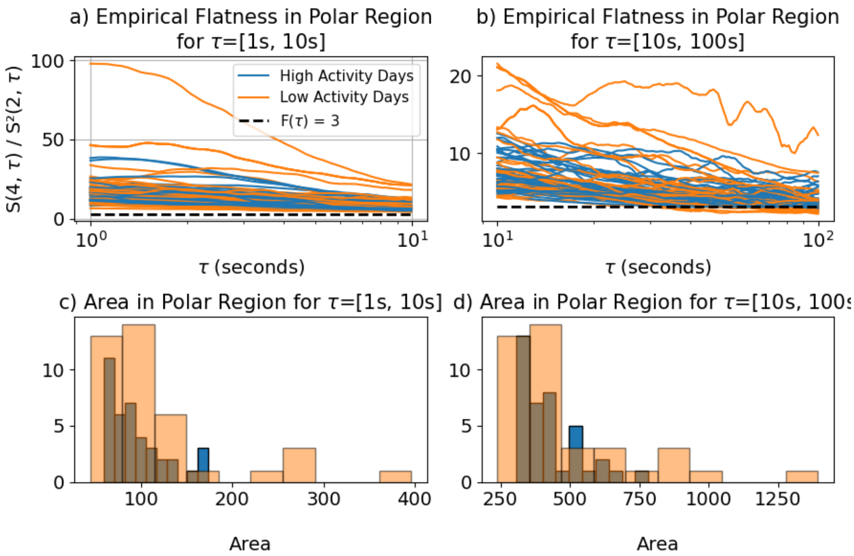


Figure 5.14: Dashed line represents $F(\tau) = 3$. (a) active days (blue) and quiet days (orange) for $\tau = [1s, 10s]$. (b) active days (blue) and quiet days (orange) for $\tau = [10s, 100s]$. (c) Area distribution of active days (blue) and quiet days (orange) for $\tau = [1s, 10s]$. (d) Area Distribution for active days (blue) and quiet days (orange) for $\tau = [10s, 100s]$.

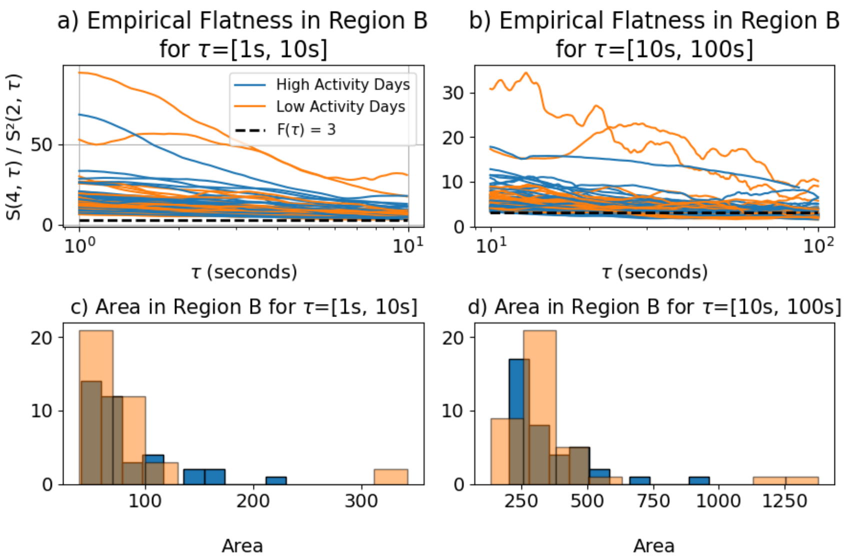


Figure 5.15: Dashed line represents $F(\tau) = 3$. (a) active days (blue) and quiet days (orange) for $\tau = [1s, 10s]$. (b) active days (blue) and quiet days (orange) for $\tau = [10s, 100s]$. (c) Area distribution of active days (blue) and quiet days (orange) for $\tau = [1s, 10s]$. (d) Area distribution of active days (blue) and quiet days (orange) for $\tau = [10s, 100s]$.

Figure 5.16 shows the empirical flatness calculated in region A. The results are fairly similar to the results for the polar cap region. Aside from some outliers, $F(\tau)$ is generally higher during days of high geomagnetic activity. As the scale increases and the flatness approaches $F(\tau) = 3$ the flatness during quiet days become more spread out.

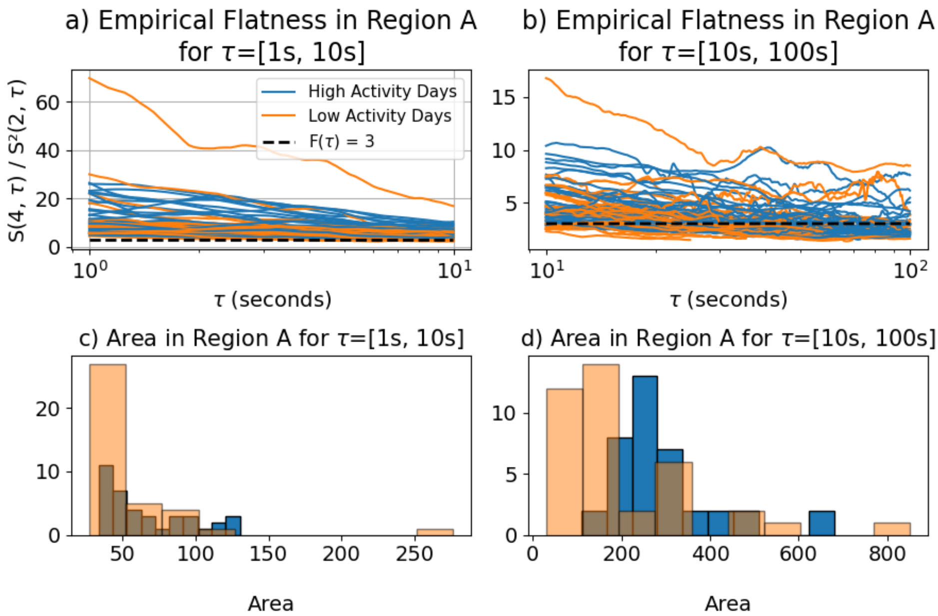


Figure 5.16: Dashed line represents $F(\tau) = 3$. (a) active days (blue) and quiet days (orange) for $\tau = [1s, 10s]$. (b) active days (blue) and quiet days (orange) for $\tau = [10s, 100s]$. (c) Area distribution of active days (blue) and quiet days (orange) for $\tau = [1s, 10s]$. (d) Area distribution of active days (blue) and quiet days (orange) for $\tau = [10s, 100s]$.

Figure 5.17 shows the empirical flatness calculated in region C. The result in Figure a) and c) share a likeness to what we saw in the entire polar region. At larger scales as shown in Figures b) and d) we see more similarities to the polar cap and region A, however.

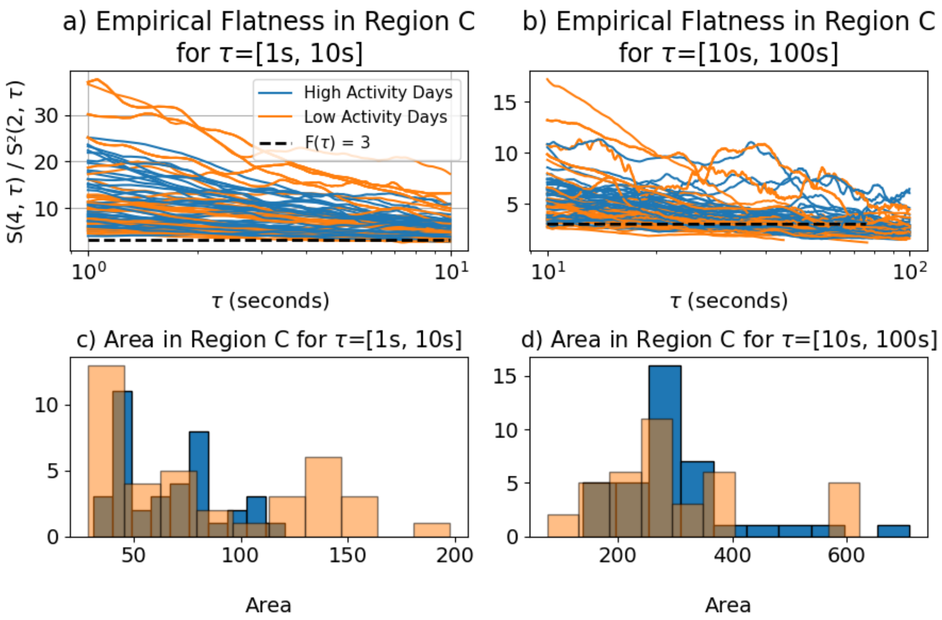


Figure 5.17: Dashed line represents $F(\tau) = 3$. (a) active days (blue) and quiet days (orange) for $\tau = [1s, 10s]$. (b) active days (blue) and quiet days (orange) for $\tau = [10s, 100s]$. (c) Area distribution of active days (blue) and quiet days (orange) for $\tau = [1s, 10s]$. (d) Area distribution of active days (blue) and quiet days (orange) for $\tau = [10s, 100s]$.

5.2.2 Area of Empirical Flatness

Figures 5.18, 5.19, 5.20 and 5.21 display the area covered by the empirical flatness for each instance, totaling 39. We do not remove the duplicates in these figures like in subsection 5.1.3. Figure a) and b) show $\tau = [1s, 10s]$ and $\tau = [10s, 100s]$ respectively when calculating the empirical flatness. The blue plots depict areas calculated during days of high geomagnetic activity while the orange plots represent areas calculated during days of low activity. We have also added the average area of all instances for high and low activity periods, displayed as dotted lines in the corresponding colors.

Figure 5.18 displays the area covered by the empirical flatness in the entire polar region. We see a spike during the low activity day for instance 14. This will be discussed in chapter 6. We cannot observe any noticeable difference outside this spike. The average is slightly higher during the low activity days, although not by a significant amount.

Figure 5.19 shows the area covered by the empirical flatness in the polar cap region. We can see two spikes at instance 5 and 14 during the low activity days.

Otherwise, the results are similar to what we saw in Figure 5.18, with a slightly smaller difference between the means.

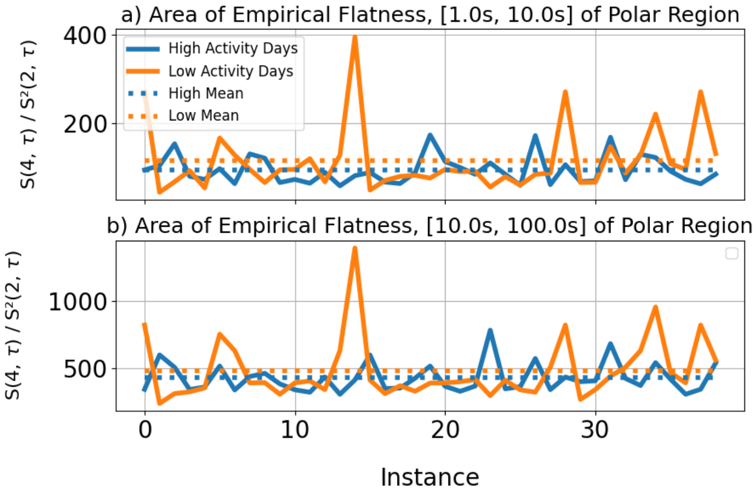


Figure 5.18: Area of empirical flatness in the entire polar region. (a) Area for smaller structures $\tau = [1s, 10s]$ calculated for all instances. (b) Area for larger structures $\tau = [10s, 100s]$ calculated for all instances.

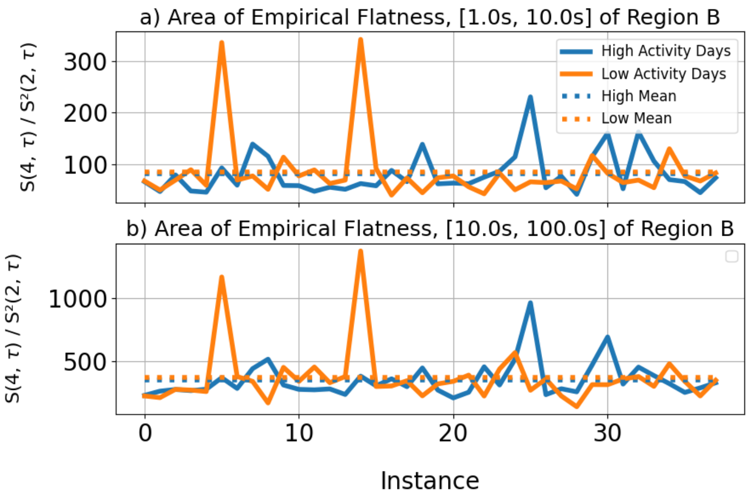


Figure 5.19: Area of empirical flatness in the polar cap region. (a) Area for smaller structures $\tau = [1s, 10s]$ calculated for all instances. (b) Area for larger structures $\tau = [10s, 100s]$ calculated for all instances.

In Figure 5.20 we can see the area covered by the empirical flatness in region A. We can see a spike at instance 18 during the low activity days. This is the only case where the average is larger for days of high geomagnetic activity than during quiet days.

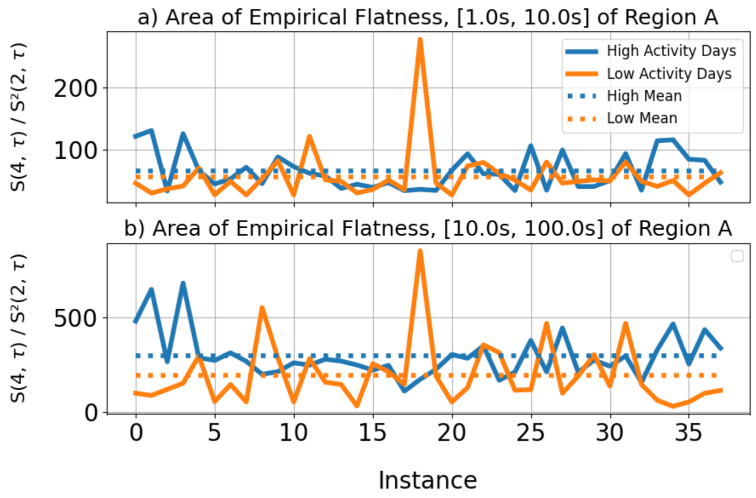


Figure 5.20: Area of empirical flatness in region A. (a) Area for smaller structures $\tau = [1s, 10s]$ calculated for all instances. (b) Area for larger structures $\tau = [10s, 100s]$ calculated for all instances.

Figure 5.21 presents the area covered by the empirical flatness in region C. We observe a large amount of high values during low activity days for smaller scales in Figure a). In Figure b) we see more evenly distributed values, like those in Figures 5.19 and 5.20.

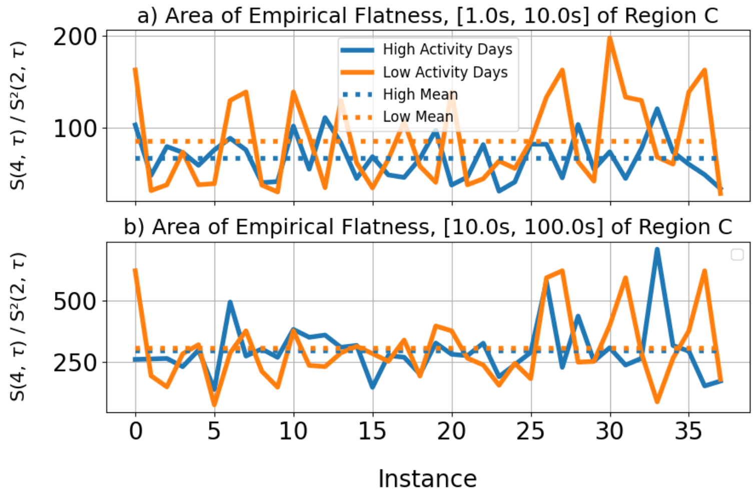


Figure 5.21: Area of empirical flatness in region C. (a) Area for smaller structures $\tau = [1s, 10s]$ calculated for all instances. (b) Area for larger structures $\tau = [10s, 100s]$ calculated for all instances.

5.3 Probability Density Function

We present the probability density functions for three different increments presented as the small increment, middle increment and large increment. The increments are as follows: $\tau = [0.0625s, 0.125s, 0.25s, 0.5s, 1.0s]$, $\tau = [1.0s, 5.0s, 10.0s]$ and $\tau = [10.0s, 50.0s, 100.0s]$. Each increment is calculated over the entire polar region. The results for the polar cap and auroral regions are hard to distinguish from the entire polar region and are not included. We only show the PDF for one instance, on 4th November 2014 during the day of high geomagnetic activity and 3rd November 2014 during the day of low geomagnetic activity. This is due to not observing any noticeable difference when investigating the results of different days. All probability density function for the high activity days are presented in Figure a) and low activity days are presented in Figure b). We have superposed a Gaussian to all of the figures to more clearly determine Gaussianity.

5.3.1 Small Increment

Figure 5.22 presents the probability density function at small increments $\tau = [0.0625s, 0.125s, 0.25s, 0.5s, 1.0s]$ in the entire polar region. Other than a sharper peak in Figure b), it is difficult to make out any differences. Both cases have sharp peaks and heavy tails, however.

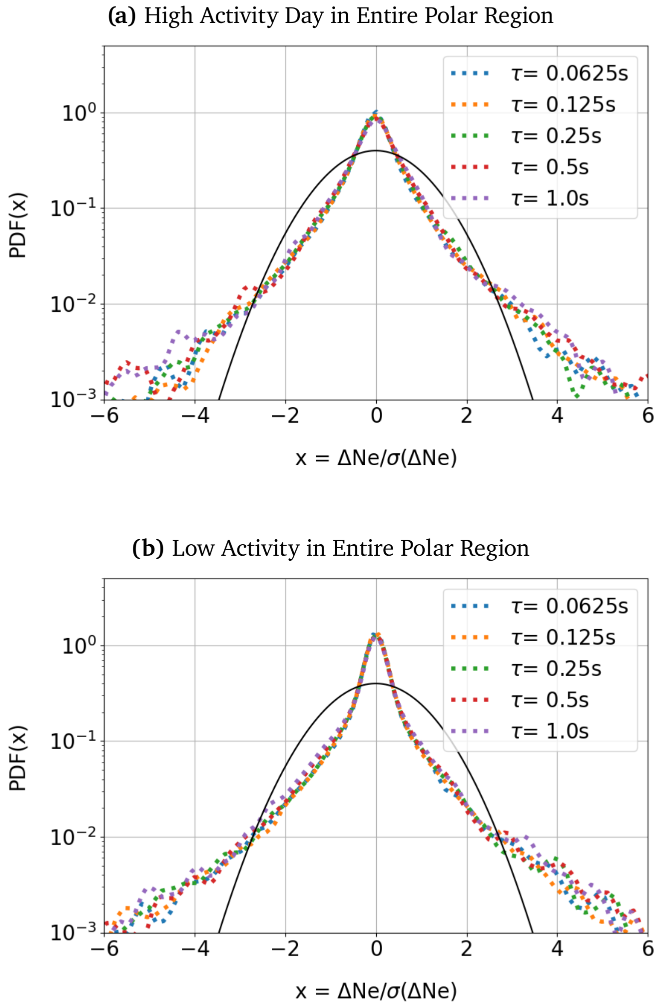


Figure 5.22: PDF for five time lags. $\tau = 0.0625s$ (dotted blue), $\tau = 0.125s$ (dotted orange), $\tau = 0.25s$, $\tau = 0.50s$ (dotted red) and $\tau = 1.0s$ (dotted purple). A Gaussian (solid black) is superposed. (a) high activity period on 4th November 2014 (b) low activity period on 3rd November 2014.

5.3.2 Middle Increment

Figure 5.23 presents the probability density function at medium increments $\tau = [1.0s, 5.0s, 10.0s]$ in the entire polar region. The Figures exhibit similar results to for the small increments.

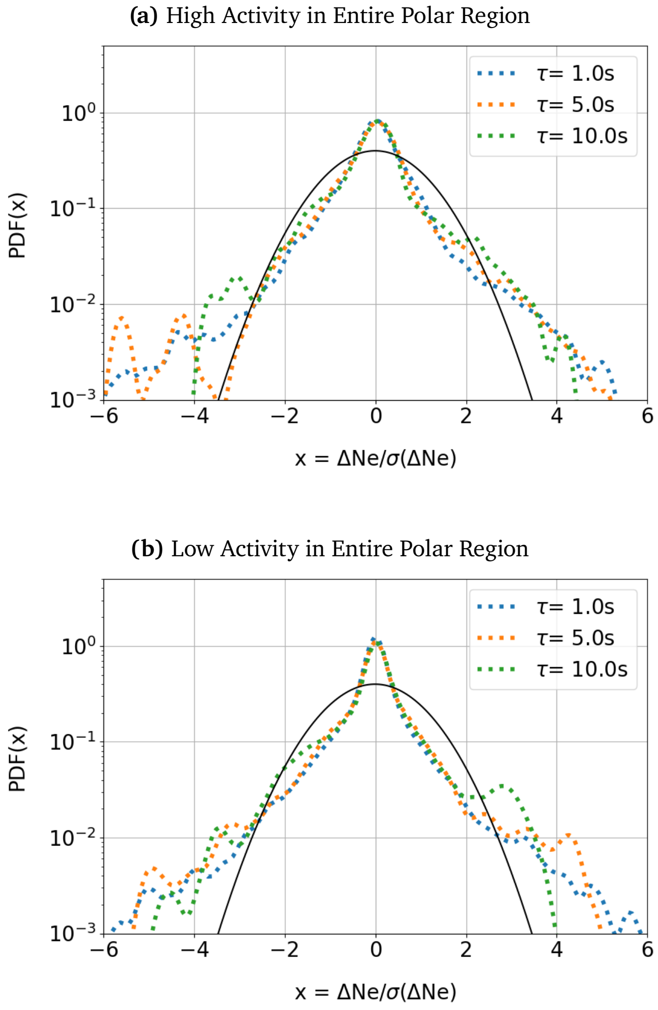
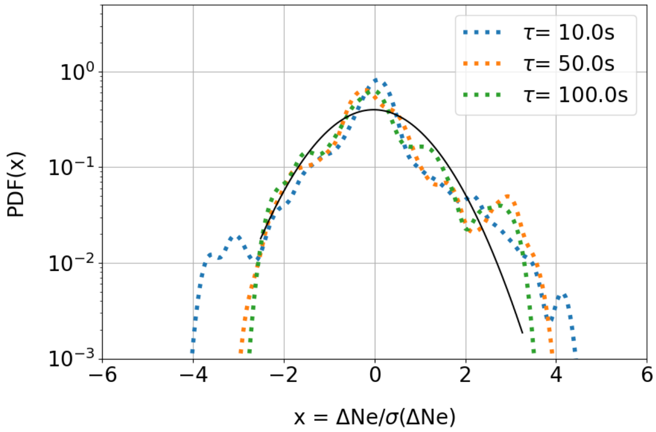


Figure 5.23: PDF for three time lags. $\tau = 1.0s$ (dotted blue), $\tau = 5.0$ (dotted orange) and $\tau = 10.0$. A Gaussian (solid black) is superposed. (a) high activity period on 4th November 2014 (b) low activity period on 3rd November 2014.

5.3.3 Large Increment

Figure 5.24 presents the probability density function at large increments $\tau = [10.0s, 50.0s, 100.0s]$ in the entire polar region. Both Figure a) and b) is closer to the superposed Gaussian compared to at the two smaller increments.

(a) High Activity in Entire Polar Region



(b) Low Activity in Entire Polar Region

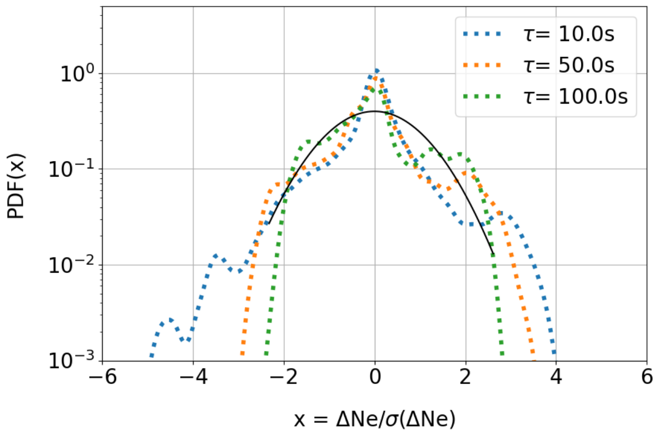


Figure 5.24: PDF for three time lags. $\tau = 1.0s$ (dotted blue), $\tau = 5.0$ (dotted orange) and $\tau = 10.0$. A Gaussian (solid black) is superposed. (a) high activity period on 4th November 2014 (b) low activity period on 3rd November 2014.

6 Discussion

In this chapter we discuss the results presented in the previous chapter. Our aim was to investigate the polar region for any differences and similarities in intermittent behaviour during days of high and low geomagnetic activity using different tools at 16 Hz resolution. We noticed the largest difference when investigating the structure functions, which may provide more insight into how structures behave at different scales. The slopes for structure functions in the polar cap was significantly different at large and small scales, and during different levels of geomagnetic activity. The auroral regions were not distinguishable between different levels of geomagnetic activity. We also note that region A and C does not relate to the dayside and nightside auroral regions and these results should be discussed in conjunction with each other. Our findings support those of previous work [Spicher et al., 2015] [De Michelis et al., 2020], which discussed the differences in polar regions and geomagnetic activity level respectively. The results may help understand how energy dissipation varies in the different high-latitude regions and how geomagnetic activity affects convection in these regions.

We structure this chapter in the same way as our results in chapter 5. We begin discussing the results we obtained by analysing the structure function. Moreover, we discuss the results of the empirical flatness. Lastly we discuss the probability density fluctuations.

6.1 Structure Function

We discuss the results presented in subsections 5.1.1 and 5.1.2 simultaneously as the figures in the first subsection can help explain some of what we see in the latter. We use the same order as before, starting with the entire polar region before moving on to the polar cap and finishing with the auroral region. The structure function was our main focus due to observing most differences using this tool.

6.1.1 Slopes

Calculations in the entire polar region exhibit similar behaviour to those made in the polar cap (Region B). At smaller scales, the slopes were generally steeper, and there was a clear difference between days of high and low geomagnetic activity. At first glance, one may conclude that since the polar cap is usually the largest of regions A, B and C, it has the most significant contribution the entire polar region. One must not forget that region A and C combined can be as large as the polar cap, however. The similarities between the polar cap and the entire polar region could still be due to the contributions of the polar cap though. Observing the histogram of slopes for regions A and C in Figures 5.8 and 5.9 we can hardly distinguish the high and low activity days. When we combine this data with the observations of slopes for the polar cap in Figure 5.7 where we have a clear distinction, we expect the results to be an intermediate between all regions. Indeed, we noted how the differences between high and low activity days are more pronounced in the polar cap than the entire polar region, which supports this explanation.

Let us further discuss the results for the polar cap. It is a common trend for slopes to become shallower as scale increase from tens of kilometres at $\tau = [1s, 10s]$ to hundreds of kilometres at $\tau = [10s, 100s]$ during days of high geomagnetic activity. This is related to a difference in energy transfer rate as plasma cascades down to smaller scales. We can expect most structure functions under these conditions to have a "knee" as the scale increases, as seen in Figure 5.2 and Figure 5.3. This tendency is not repeated in days of low geomagnetic activity. This result indicates how scale invariance in the polar cap is connected to the level of geomagnetic activity, which agrees with the findings of [De Michelis et al., 2020] who observed a notable increase in intermittency in the polar cap during a geomagnetic storm event. Similar findings are presented in [Tozzi et al., 2023], who observed higher scaling exponents for polar cap patches during periods of increased geomagnetic activity. The physical mechanisms behind a decrease in slope at larger scales can be connected to a change in energy transfer rate in turbulence as the energy cascades down into smaller scales. This leads to sporadic localised structures, i.e. intermittency. At even smaller scales the results are fairly difficult to analyse. A reason could be that the regression is calculated for 16 data points for $\tau = [0.0625s, 1s]$ in contrast with 160 for $\tau = [1s, 10s]$ and 1600 for $\tau = [10s, 100s]$. The result indicates structures in the polar cap being affected by the geomagnetic level. Both [Consolini et al., 2020] and [Spicher et al., 2015] suggested that the strong gradient drift is one of the primary mechanisms behind plasma irregularities in the polar cap. This mechanism is also mentioned by [Tozzi et al., 2023] who also linked these events and velocity-shear to a heightened geomagnetic activity.

In region A the difference in steepness between large and small scales are not as pronounced as in the polar cap, while Region C does not exhibit any significant difference. We can not observe any clear difference when inspecting days of high and low geomagnetic activity in either regions which suggest that the the creation of intermittent structures in the auroral region is not affected by geomagnetic activity. [Consolini et al., 2020] found signatures of intermittency in the auroral oval, which we could not observe by analysing the structure function. A possible explanation is not separating Region A and C into dayside and nightside regions. We see a slight difference in the distribution of slopes at small scales when comparing Region A and C, which may be due to an uneven distribution of dayside and nightside regions. [De Michelis et al., 2020] discussed how the geomagnetic activity caused more pronounced results during the geomagnetic storm which is not obvious in our results concerning the auroral region.However, we can only observe the scaling behaviour of the entire system, which may vary more locally dependent on geomagnetic activity. We suggest that other methods should be used to try and distinguish days of varying geomagnetic activity from each other, such as investigating the scaling exponents from $m = 1$ to $m = 4$ as in [Spicher et al., 2015].

The length of our data is also longer in the polar cap, as mentioned in chapter 4.1, which leads to more precise calculations in this region. Region A and C are significantly shorter, which especially affects our data at larger scales.

6.1.2 Area

The area calculated by integrating the structure functions are mainly to give us an estimate of which structure function produces higher output values. The high activity periods are in almost all cases larger than the low activity periods. When we consider an increase in particle precipitation during the active periods this makes sense for the auroral region where particle precipitation occur. We mentioned in section 2.1 how plasma can be transported across the polar cap from other regions such as the auroral oval through convection, which explains the heightened density.

which was primarily associated with geomagnetic storms. Larger values are not necessarily a product of intermittency, instead they reveal more about how high the background electron density is. The results are best interpreted in hand with the results discussing the slopes of the structure functions.

6.2 Empirical Flatness

We anticipate larger flatness values at smaller scales based on previous work by [Spicher et al., 2015], [Chian et al., 2008], [Sahraoui, 2008] and [Wernik et al., 2003]. We do not however see the largest values during days of high geomagnetic activity.

In Panels a) and b) in Figures 5.14, 5.16, 5.15 and 5.17 some instances during low activity days have much higher flatness at small scales, which suggests more intermittency [Sahraoui, 2008]. The highest flatness during a quiet day is on 3rd December 2015, with corresponding high activity period on 5th December 2015 represented by instance 14 in Figure 5.18 covering the entire polar region. High intermittency is characterised by a large flatness for small structures which we discuss further.

Figure 6.1 exhibit the empirical flatness during instance 14 on 5th and 3rd December 2015 for the high and low activity day, respectively. The calculations are made over the entire polar region. We see an extremely large flatness for the inactive day, which without further investigating would look like signs of intermittency.

Figure 6.2 shows the structure functions for instance 3 on 5th and 3rd December 2015. Figure a) display the structure function of order $m = 2$ for both days while Figure b) display the structure function of order $m = 4$ for both days. Notice how the structure function is higher during the active day for $m = 2$, but the low activity day is higher for $m = 4$. The m-th order of the structure function enhances outliers when calculating the average [Dyrud et al., 2008], and a higher order leads to these outliers contributing more to the final output of the structure function.

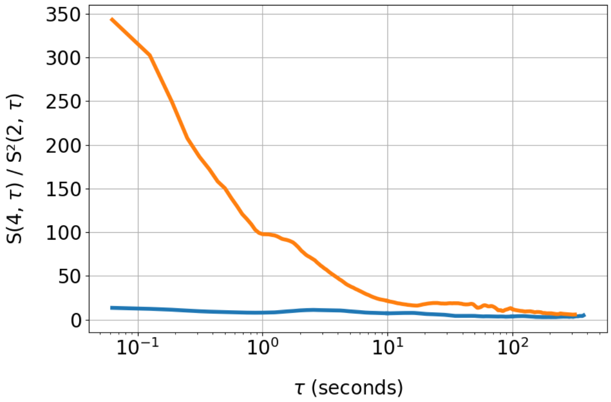
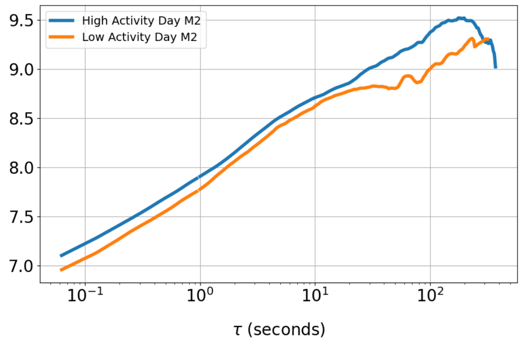


Figure 6.1: Empirical flatness for day of high geomagnetic activity (blue) and day of low geomagnetic activity (orange)

(a) Structure Function for M=2



(b) Structure Function for M=4

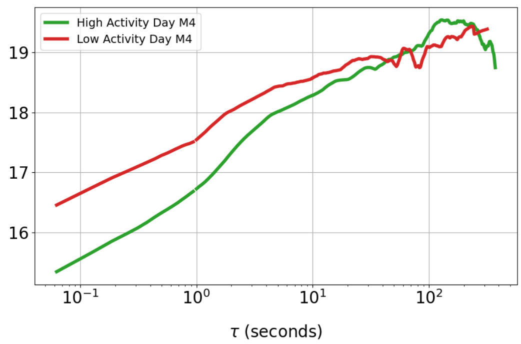
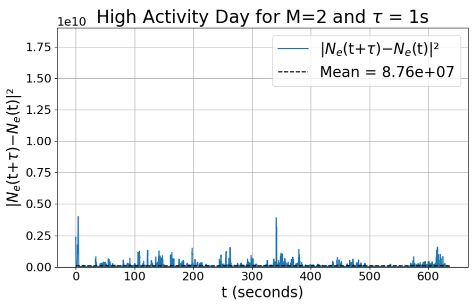


Figure 6.2: Structure function for $m = 2$ and $m = 4$ during active day 5th December 2015, and quiet day 3rd December 2015. (a) Active $m = 2$ (blue) and quiet $m = 2$ (orange). (b) Active $m = 4$ (green) and quiet $m = 4$ (red).

To explain this further, Figures 6.3 and 6.4 present the plasma density used in the structure function before calculating the average. The average is calculated for $\tau = 1s$, which is at the very beginning of the structure function where the differences between the m-th orders are highest. The x-axis in each of these figures display the length of the signal. Since $\tau = 1$ we calculate $|N_e(t+\tau) - N_e(t)|$ for the entire length of the signal minus one second. Figures a) exhibits the calculations for the high activity day and Figure b exhibits the calculations during the low activity day. The dashed line represents the average of all data points.

For $m = 2$ we can in Figure 6.3 b) see a spike around 400 seconds while otherwise the values are much lower than in figure a). The average in Figure a) is still higher, however. In Figure 6.4 b) the value of the spike is amplified and the average have surpassed that of Figure 6.4 a).

(a) Subfigure A



(b) Subfigure B

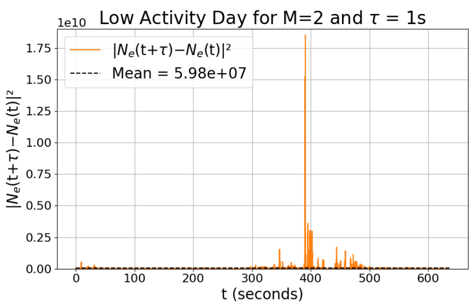
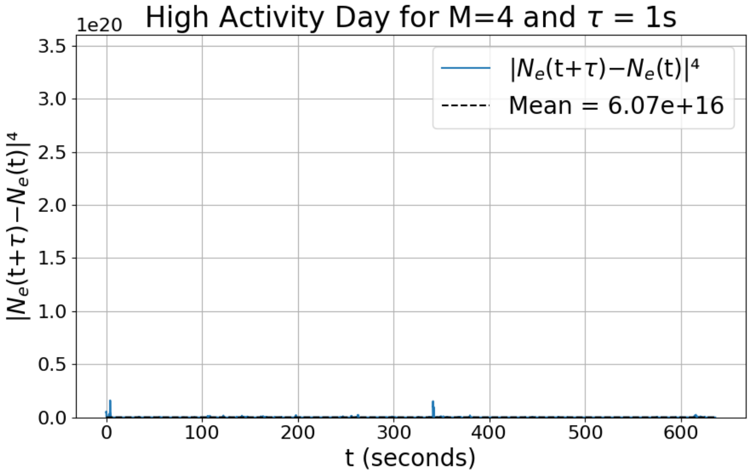


Figure 6.3: Average of change in electron density between time t and $\tau = 1s$ of order $m = 2$

(a) Subfigure A



(b) Subfigure B

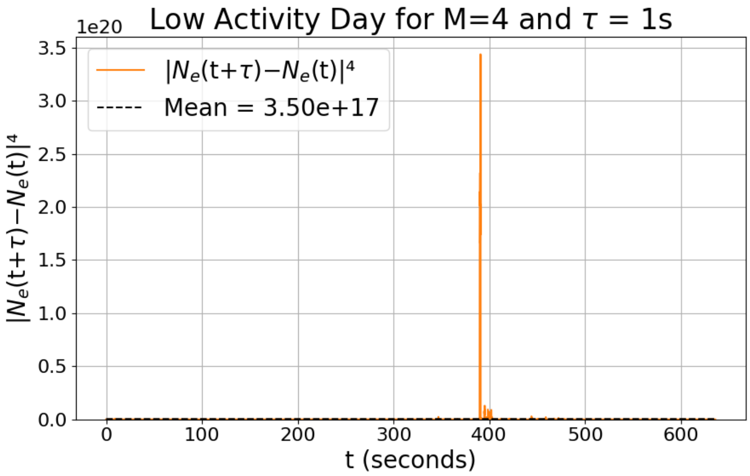


Figure 6.4: Average of change in electron density between time t and $\tau = 1s$ of order $m = 4$

This increase in flatness for the low activity day seems to come from a single event, reinforced by higher orders. These types of cases for the magnetic field is discussed in [Horbury and Balogh, 1997]. Their solution was to separate the data into multiple bins, and calculate the contribution of each bin to the total sum. Bins which contribution more than 2% but consisted of less than 10 points were rejected. A similar solution can be implemented in our algorithm, although for this to be viable the code requires further optimization due to memory consumption. This method could possibly remove outliers from our data.

When investigating other instances corresponding to very high values of empirical flatness similar observations were made, although not as extreme as in Figure 6.4. Indeed, singular events can have an effect on some of our data and measures should be taken to avoid these anomalies in the future.

We remark that our result is inconclusive. To be certain of our result, we would require implementing a method similar to [Horbury and Balogh, 1997] or examine each instance for singular events. We cannot with certainty say that the empirical flatness is higher during our days of high geomagnetic activity without further investigation. We introduce some more suggestions in 7.1.

6.3 Probability Density Function

In previous work by [Consolini et al., 2020], they showed that during periods of high geomagnetic activity the electron density fluctuations are non-Gaussian at short timescales $\tau < 50s$. Our results are very much alike, with shorter timescales having stronger departures from Gaussianity. Indeed, we see other work also achieving this result. A case covering atmospheric turbulence by [Chian et al., 2008] exhibited the same result. As τ increased, so did the Gaussianity of the data.

However, we see the same result during periods of low geomagnetic activity. Due to more intermittent behaviour having been observed when the geomagnetic activity was higher [Lovati et al., 2023][Tozzi et al., 2023] we expect the PDF to deviate from the Gaussian more during the active days. Other works have confirmed the results of the PDFs by measuring the empirical flatness [Sahraoui, 2008]. The single events may affect our probability density functions similar to the empirical flatness.

Overall, the results were fairly similar across the different instances and we could not see much difference between days of high and low geomagnetic activity. Further analysis on this subject is needed to investigate whether there are any differences.

7

Conclusion

In this work we investigated whether different data analysis tools can reveal any additional information about the nature of ionospheric turbulence and intermittency at various polar regions. This task involved using these tools to investigate any differences and similarities in the polar cap and auroral region during days of high and low geomagnetic activity. For this we used 16 Hz plasma density measurements from Swarm A in the winter months of 2014 and 2015.

We calculated the structure function for 39 different periods during high geomagnetic activity and 31 different periods during low geomagnetic activity. Additionally, we measured the slopes of the structure functions at scales of tens of kilometres $\tau = [1s, 10s]$ and hundreds of kilometres $\tau = [10s, 100s]$. We did this for the entire polar region, the auroral regions A and C on opposite sides of the polar cap, and within the polar cap (Region B). Our results indicated a difference in the scaling behaviour in the polar cap at both varying geomagnetic levels and scale sizes. [Consolini et al., 2020] and [Spicher et al., 2015] suggested the irregularities being driven by gradient drift instabilities, which have been linked to geomagnetic activity [Tozzi et al., 2023]. We could not observe any notable differences for the auroral regions, however. This may suggest a similar global scaling behaviour during various geomagnetic activity levels in the auroral regions. We did not separate the auroral regions into dayside and nightside, which may have affected the results. The reduced data length in the auroral region compared to the polar cap could have also have had an effect on the precision of the calculations. Future work should consist of separating

Region A and C into dayside and nightside regions using the magnetic local time, and investigating the scaling exponents of the structure function from $m = 1$ up to $m = 4$.

Furthermore we investigated the empirical flatness and probability density function in these regions. Both methods exhibited intermittent behaviour such as large flatness and non-Gaussian distributions at at small scales. However, we could not observe any notable differences in the different regions, nor the different geomagnetic activity levels. The only exceptions were some periods with singular large peaks in electron density fluctuations during quiet days where the flatness $F(\tau)$ was much higher than days which showed more irregularities when inspecting the density fluctuations. We therefore suggest filtering out outliers in future work concerning the empirical flatness and probability density function.

We observed differences when investigating the structure function in the polar cap and in the entire polar region, when comparing active and quiet days, by differentiating dayside and nightside auroral regions we may be able to obtain more information by investigating the slopes at various scales. Further investigation is needed before we can ascertain any differences when using tools such as the empirical flatness and probability density function.

7.1 Future Work

In this section we present some suggestions for future work.

We could make some improvements to the code, especially making it more memory efficient, so use of vectorization can speed up our calculations. This can be challenging in Python due to the way it handles memory management, however. We can also obtain more data to improve the statistical findings. By carrying out these two things we can process more data faster.

A particularly fascinating option is separating the auroral region into magnetic local time (dayside/nightside). This method is already partially implemented in our code, as we have converted latitude and longitude to magnetic coordinates, though it somewhat unoptimized. Another partial implementation is applying the tools within the south pole region. Everything needed to investigate the south pole is already implemented, but it was not investigated further in this work due to time constraints. We could also calculate the slopes for the 1st and 3rd order structure function such that we can check for scaling behaviour at various scales.

Furthermore we want to implement a method for removing the single events, assessing whether the empirical flatness yields different results. This may be done by using a smoothing window, or something similar to what [Horbury and Balogh, 1997] presented. We also want to investigate the probability density fluctuations in greater detail.

Implementing some, if not all of these methods may provide answers to some of our findings where we could not observe the same result as previous literature. We especially want to investigate further if we can detect changes in turbulence and intermittency during different geomagnetic activity levels in the auroral regions.

8

Appendix

Acknowledgements

Electron density measurements obtained by Swarm can be accessed at the European Space Agency's Swarm data access website:
(*https://swarm-diss.eo.esa.in*)
Further information can be found in the Swarm handbook:
(*https://swarmhandbook.earth.esa.int/*)

I want to thank my supervisor, Andres for invaluable help with the project. I also want to thank my fellow students for helpful discussions, especially Kian who also brought a fan to the office during hot summer days, and I want to thank family and friends for support.

Bibliography

[Akasofu et al., 1965] Akasofu, S.-I., Chapman, S., and Meng, C.-I. (1965). The polar electrojet. *Journal of Atmospheric and Terrestrial Physics*, 27(11):1275–1305.

[Anderson, 1999] Anderson, D. (1999). *The ionosphere*. Space Environment Center.

[Basu et al., 1990] Basu, S., Basu, S., MacKenzie, E., Coley, W. R., Sharber, J. R., and Hoegy, W. R. (1990). Plasma structuring by the gradient drift instability at high latitudes and comparison with velocity shear driven processes. *Journal of Geophysical Research: Space Physics*, 95(A6):7799–7818.

[Bruno et al., 2001] Bruno, R., Carbone, V., Veltri, P., Pietropaolo, E., and Bavassano, B. (2001). Identifying intermittency events in the solar wind. *Planetary and Space Science*, 49(12):1201–1210. Nonlinear Dynamics and Fraactals in Space.

[Chian et al., 2008] Chian, A. C.-L., Miranda, R. A., Koga, D., Bolzan, M. J. A., Ramos, F. M., and Rempel, E. L. (2008). Analysis of phase coherence in fully developed atmospheric turbulence: Amazon forest canopy. *Nonlinear Processes in Geophysics*, 15(4):567–573.

[Consolini et al., 2020] Consolini, G., De Michelis, P., Alberti, T., Coco, I., Giannattasio, F., Tozzi, R., and Carbone, V. (2020). Intermittency and passive scalar nature of electron density fluctuations in the high-latitude ionosphere at swarm altitude. *Geophysical Research Letters*, 47(18):e2020GL089628. e2020GL089628 10.1029/2020GL089628.

[Dandekar and Bullett, 1999] Dandekar, B. S. and Bullett, T. W. (1999). Morphology of polar cap patch activity. *Radio Science*, 34(5):1187–1205.

[Davis and Sugiura, 1966] Davis, T. N. and Sugiura, M. (1966). Auroral electrojet activity index ae and its universal time variations. *Journal of Geophysical Research (1896-1977)*, 71(3):785–801.

[De Michelis et al., 2020] De Michelis, P., Pignalberi, A., Consolini, G., Coco, I., Tozzi, R., Pezzopane, M., Giannattasio, F., and Balasis, G. (2020). On the 2015 st. patrick's storm turbulent state of the ionosphere: Hints from the swarm mission. *Journal of Geophysical Research: Space Physics*, 125(8):e2020JA027934. e2020JA027934 10.1029/2020JA027934.

[Dyrud et al., 2008] Dyrud, L., Krane, B., Oppenheim, M., Pécseli, H. L., Trulsen, J., and Wernik, A. W. (2008). Structure functions and intermittency in ionospheric plasma turbulence. *Nonlinear Processes in Geophysics*, 15(6):847–862.

[ESA, a] ESA. Swarm Mission. `https://earth.esa.int/eogateway/missions/swarm`. Spring 2024.

[ESA, b] ESA. Swarm Overview. `https://earth.esa.int/eogateway/missions/swarm/description`. Spring 2024.

[ESA, c] ESA. Swarm Terms and Conditions. `https://earth.esa.int/eogateway/documents/d/earth-online/esa-eo-data-policy`. Spring 2024.

[Frisch, 1995] Frisch, U. (1995). *Turbulence: The Legacy of A.N. Kolmogorov*.

[Horbury and Balogh, 1997] Horbury, T. S. and Balogh, A. (1997). Structure function measurements of the intermittent mhd turbulent cascade. *Nonlinear Processes in Geophysics*, 4(3):185–199.

[Iijima and Potemra, 1976] Iijima, T. and Potemra, T. A. (1976). Field-aligned currents in the dayside cusp observed by triad. *Journal of Geophysical Research (1896-1977)*, 81(34):5971–5979.

[Jin et al., 2019] Jin, Y., Spicher, A., Xiong, C., Clausen, L. B. N., Kervalishvili, G., Stolle, C., and Miloch, W. J. (2019). Ionospheric plasma irregularities characterized by the swarm satellites: Statistics at high latitudes. *Journal of Geophysical Research: Space Physics*, 124(2):1262–1282.

[Kelley, 2009] Kelley, M. C. (2009). *The Earth's Ionosphere: Plasma Physics and Electrodynamics*. Elsevier, Amsterdam, 2nd edition.

[Kintner and Seyler, 1985] Kintner, P. M. and Seyler, C. E. (1985). The status of observations and theory of high latitude ionospheric and magnetospheric plasma turbulence. *Space Science Reviews*, 41(1):91–129.

[Lester, 2003] Lester, M. (2003). Ionospheric convection and its relevance for space weather. *Advances in Space Research*, 31(4):941–950.

[Lovati et al., 2023] Lovati, G., Michelis, P. D., Consolini, G., Pezzopane, M., Pignalberi, A., and Berrilli, F. (2023). Decomposing solar and geomagnetic activity and seasonal dependencies to examine the relationship between gps loss of lock and ionospheric turbulence. *Scientific Reports*, 13(1):9287.

[Mandeep et al., 2014] Mandeep, J. S., Oliveira, K., Moraes, A. d. O., Costa, E., Honorato Muella, M. T. d. A., de Paula, E. R., and Perrella, W. (2014). Validation of the gps ionospheric amplitude scintillation model of the power spectral density. *International Journal of Antennas and Propagation*, 2014:573615.

[Mitchell et al., 2005] Mitchell, C. N., Alfonsi, L., De Franceschi, G., Lester, M., Romano, V., and Wernik, A. W. (2005). Gps tec and scintillation measurements from the polar ionosphere during the october 2003 storm. *Geophysical Research Letters*, 32(12).

[Moen, Jøran et al., 2013] Moen, Jøran, Oksavik, Kjellmar, Alfonsi, Lucilla, Daabakk, Yvonne, Romano, Vineenzo, and Spogli, Luca (2013). Space weather challenges of the polar cap ionosphere. *J. Space Weather Space Clim.*, 3:A02.

[Monin and Yaglom, 1975] Monin, A. S. and Yaglom, A. M. (1975). *Statistical Fluid Mechanics: Mechanics of Turbulence*, volume 1 and 2. MIT Press, Cambridge, MA.

[Noja et al.,] Noja, M., Stolle, C., Park, J., and Lühr, H. Long-term analysis of ionospheric polar patches based on champ tec data. *Radio Science*, 48(3):289–301.

[Phelps and Sagalyn, 1976] Phelps, A. D. R. and Sagalyn, R. C. (1976). Plasma density irregularities in the high-latitude top side ionosphere. *Journal of Geophysical Research (1896-1977)*, 81(4):515–523.

[Sahraoui, 2008] Sahraoui, F. (2008). Diagnosis of magnetic structures and intermittency in space-plasma turbulence using the technique of surrogate data. *Phys. Rev. E*, 78:026402.

[SciPy, a] SciPy. scipy.integrate.simpson. `https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.simpson.html#scipy.integrate.simpson`. Spring 2024.

[SciPy, b] SciPy. scipy.stats.linregress. `https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.linregress.html`. Spring 2024.

[Seaborn,] Seaborn. seaborn.kdeplot. `https://seaborn.pydata.org/`

generated/seaborn.kdeplot.html. Spring 2024.

[Spicher et al., 2015] Spicher, A., Miloch, W. J., Clausen, L. B. N., and Moen, J. I. (2015). Plasma turbulence and coherent structures in the polar cap observed by the ici-2 sounding rocket. *Journal of Geophysical Research: Space Physics*, 120(12):10,959–10,978.

[The European Space Agency,] The European Space Agency. Swarm Data Access. https://earth.esa.int/eogateway/missions/swarm/data. Spring 2024.

[Tozzi et al., 2023] Tozzi, R., De Michelis, P., Lovati, G., Consolini, G., Pignalberi, A., Pezzopane, M., Coco, I., Giannattasio, F., and Marcucci, M. F. (2023). Polar cap patches scaling properties: Insights from swarm data. *Remote Sensing*, 15(17).

[Tsunoda, 1988] Tsunoda, R. T. (1988). High-latitude f region irregularities: A review and synthesis. *Reviews of Geophysics*, 26(4):719–760.

[WDC, a] WDC. World Data Center for Geomagnetism, Kyoto. Provisional AE index Realtime. https://wdc.kugi.kyoto-u.ac.jp/ae_realtime/202405/index_20240511.html. Accessed:Spring 2024.

[WDC, b] WDC. World Data Center for Geomagnetism, Provisional AE index. https://wdc.kugi.kyoto-u.ac.jp/ae_provisional/index.html. Accessed:Spring 2024.

[Wernik et al., 2003] Wernik, A., Secan, J., and Fremouw, E. (2003). Ionospheric irregularities and scintillation. *Advances in Space Research*, 31(4):971–981.

[World Data System,] World Data System. Data Sharing Principles. https://worlddatasystem.org/about/data-sharing-principles/. Spring 2024.

8.1 Table of Days Used

We present a table with each day of high and corresponding low geomagnetic activity used in this work. It should be noted that even though many active days share the same quiet days, most events are not compared to the same period during the quiet days. In Table **??**, the days of high geomagnetic activity are displayed at the left hand side, while the days of low geomagnetic activity are shown on the right hand side.

Active	Quiet
2014, 11, 4	2014, 11, 3
2014, 12, 7	2014, 12, 6
2015, 11, 7	2015, 11, 2
2015, 11, 8	2015, 11, 12
2015, 11, 9	2015, 11, 12
2015, 11, 10	2015, 11, 12
2015, 11, 11	2015, 11, 12
2015, 12, 5	2015, 12, 3
2015, 12, 6	2015, 12, 4
2015, 12, 11	2015, 12, 4
2015, 12, 14	2015, 12, 3
2015, 12, 20	2015, 12, 19
2015, 12, 31	2015, 12, 30

Table 8.1: Dates of Active and Quiet periods

8.2 Power Spectral Density

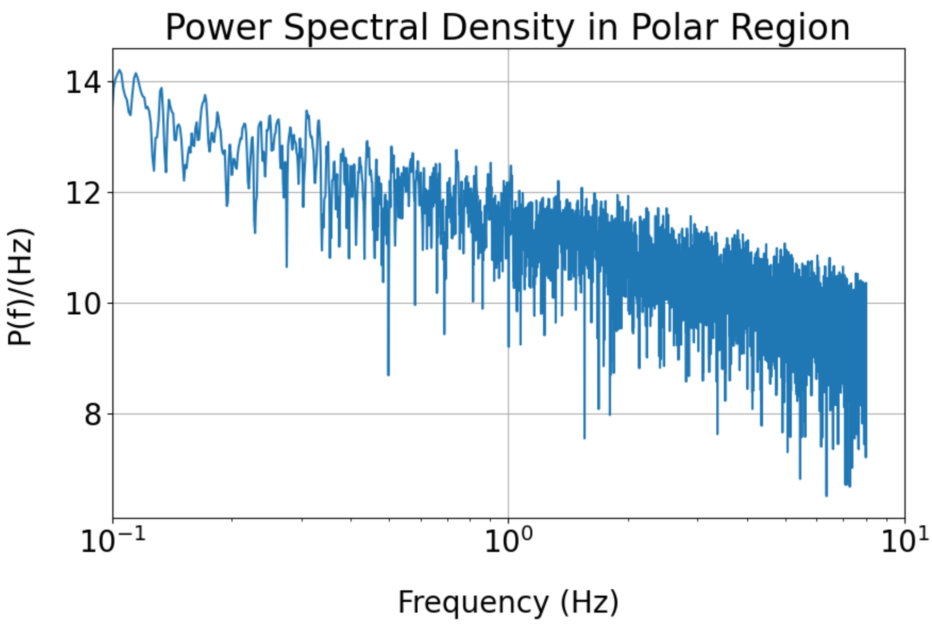


Figure 8.1: Power spectral density during 31st December 2015 over the entire polar region.

8.3 Slopes for Structure Functions at Very Small Scales

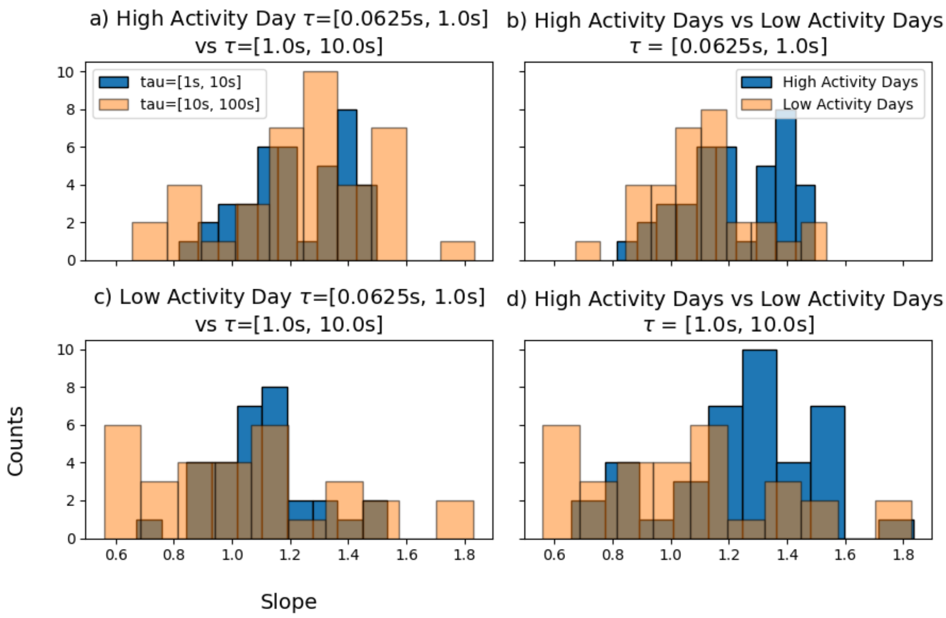


Figure 8.2: Slopes for all structure functions calculated over the entire polar region. (a) Slopes for $\tau = [0.0625s, 1s]$ vs $\tau = [1s, 10s]$ during days of high geomagnetic activity. (b) Slopes during high geomagnetic activity vs low geomagnetic activity for $\tau = [0.0625s, 1s]$. (c) Slopes for $\tau = [0.0625ss, 1s]$ vs $\tau = [1s, 10s]$ during days of low geomagnetic activity. (d) Slopes during high geomagnetic activity vs low geomagnetic activity for $\tau = [1s, 10s]$.

8.4 AE Index During Storm Event 11th May 2024

We have included the AE index of the storm event on 11th May 2024, as it is a quite interesting case. Similar to the other figures displaying the AE index, this was obtained from World Data Center for Geomagnetism [WDC, a], in the realtime section.

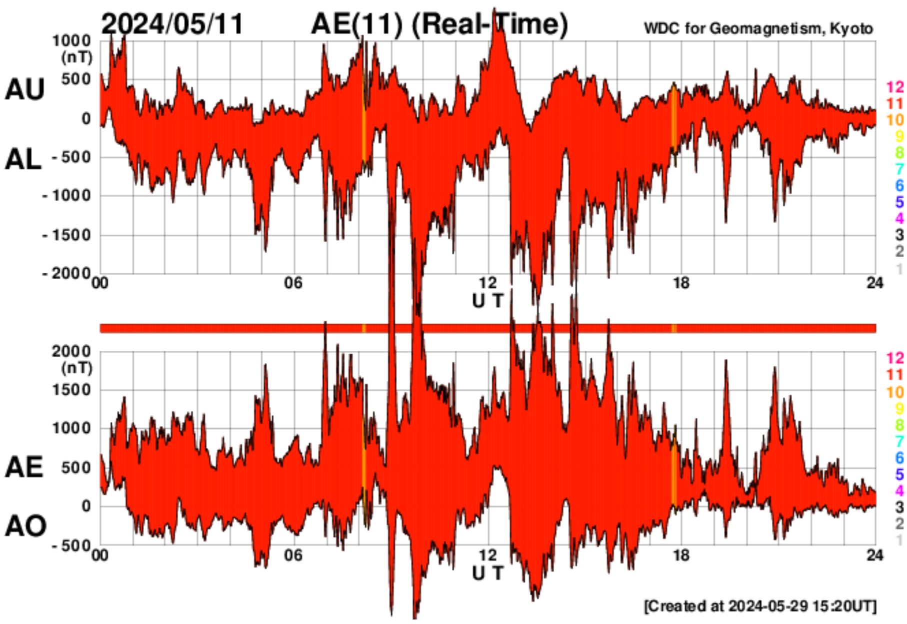


Figure 8.3: AE index 11th May 2024. [WDC, a]

8.5 Code

We present the code used to implement the turbulence data analysis tools in the following subsections. All files in the following subsections are needed for the code to work properly.

8.5.1 Calculating Storing and Displaying Data

```
import getData as gd
import dataProcessing as dp
import plotting as pt
import matplotlib.pyplot as plt
from day_parameters import load_day
import pandas as pd
import numpy as np


def run(date, instance, processing_parameters=None, plotting_trajectory=False,
        plotting_region=False, plotting_structure_function=False,
```

```
        plotting_ratios=False, plotting_psd=False, plotting_pdf=False,
        write_to_csv=False):
    """

    Applies all other classes to display and calculate using various tools
    write_to_csv stores as datafram in .csv files.
    """

    if processing_parameters is None:
        processing_parameters = {'merged_region': True,
                                 'region_name': 'B',
                                 'tau_interval': 'auto',
                                 'm': 2,
                                 'comparison': 'all',
                                 'divide_structure_function': False,
                                 'print_time_interval': False,
                                 'target': False,
                                 'polar_region': 'all',
                                 'normalize': False,
                                 'write_to_pole': 'North'}

    year, month, day = date
    merged_region = processing_parameters['merged_region']
    region_name = processing_parameters['region_name']
    t = processing_parameters['tau_interval']
    m = processing_parameters['m']
    comparison = processing_parameters['comparison']
    divide_structure_function = processing_parameters['divide_structure_function']
    target = processing_parameters['target']
    polar_region = processing_parameters['polar_region']
    normalize_data = processing_parameters['normalize']
    write_to_pole = processing_parameters['write_to_pole']

    region = 'all' if merged_region else region_name

    dataset = load_day(year, month, day, instance, merged_region)

    fac_parameters_north = dataset['FAC_parameters_north']
    fac_parameters_north_inactive = dataset['FAC_parameters_north_inactive']
    fac_parameters_south = dataset['FAC_parameters_south']
    fac_parameters_south_inactive = dataset['FAC_parameters_south_inactive']
```

```
    date = dataset['date']
    date_inactive = dataset['date_inactive']

    day_start = dataset['day_start']
    day_stop = dataset['day_stop']
    day_start_inactive = dataset['day_start_inactive']
    day_stop_inactive = dataset['day_stop_inactive']


    active_day_16Hz = gd.GetData(day_start, day_stop, 'Ne').time()
    active_day_FAC = gd.GetData(day_start, day_stop, 'FAC').time()

    gd.GetData(day_start, day_stop, 'FAC').get_info()
    gd.GetData(day_start, day_stop, 'Ne').get_info()

    inactive_day_16Hz = gd.GetData(day_start_inactive, day_stop_inactive, 'Ne').time()
    inactive_day_FAC = gd.GetData(day_start_inactive, day_stop_inactive, 'FAC').time()

    active_day_north = dp.DataProcessing(active_day_16Hz, active_day_FAC, fac_parameters_north)
    inactive_day_north = dp.DataProcessing(inactive_day_16Hz, inactive_day_FAC, fac_parameters_north_inactive)
    active_day_south = dp.DataProcessing(active_day_16Hz, active_day_FAC, fac_parameters_south)
    inactive_day_south = dp.DataProcessing(inactive_day_16Hz, inactive_day_FAC, fac_parameters_south_inactive)

    if processing_parameters['print_time_interval']:
        if comparison != 'South' or comparison != 'ActiveInactiveSouth':
            print('North Active')
            print(active_day_north.return_data()['time_interval'])
        if comparison == 'all' or comparison == 'ActiveInactiveNorth':
            print('North Inactive')
            print(inactive_day_north.return_data()['time_interval'])
        if comparison == 'all' or comparison == 'South' or comparison == 'ActiveInactiveSouth':
            print('South Active')
            print(active_day_south.return_data()['time_interval'])
        if comparison == 'all' or comparison == 'ActiveInactiveSouth':
            print('South Inactive')
            print(inactive_day_south.return_data()['time_interval'])


    if plotting_trajectory:
```

```
        active_day_north.plot_trajectory(f'plots/_{date}_{instance}', other_day=inactive_day_north.return_data(), all_orbits=True)
        active_day_north.find_closest_region(inactive_day_north.return_data())


    if plotting_region:
        target_name = 'Target_'
        if not target:
            target_name = ''
            polar_region = 'all'

        match comparison:
            case 'NorthSouth':
                pt.plot_ne_and_fac(active_day_north.return_data(), name=f'plots/{target_name}Ne_and_FAC_{date}_{instance}_north', target=target, polar_region=polar_region)
                pt.plot_ne_and_fac(active_day_south.return_data(), name=f'plots/{target_name}Ne_and_FAC_{date}_{instance}_south', target=target, polar_region=polar_region)
            case 'ActiveInactiveNorth':
                pt.plot_ne_and_fac(inactive_day_north.return_data(), name=f'plots/{target_name}Ne_and_FAC_inactive_{date}_{instance}_north', target=target, polar_region=polar_region)
                pt.plot_ne_and_fac(active_day_north.return_data(), name=f'plots/{target_name}Ne_and_FAC_{date}_{instance}_north', target=target, polar_region=polar_region)
            case 'ActiveInactiveSouth':
                pt.plot_ne_and_fac(active_day_south.return_data(), name=f'plots/{target_name}Ne_and_FAC_{date}_{instance}_south', target=target, polar_region=polar_region)
                pt.plot_ne_and_fac(inactive_day_south.return_data(), name=f'plots/{target_name}Ne_and_FAC_inactive_{date}_{instance}_south', target=target, polar_region=polar_region)
            case 'all':
                pt.plot_ne_and_fac(inactive_day_north.return_data(), name=f'plots/{target_name}Ne_and_FAC_inactive_{date}_{instance}_north', target=target, polar_region=polar_region)
```

```
                pt.plot_ne_and_fac(active_day_north.return_data(), name=f'plots/{target_name}Ne_and_FAC_{date}_{instance}_north', target=target, polar_region=polar_region)
                pt.plot_ne_and_fac(active_day_south.return_data(), name=f'plots/{target_name}Ne_and_FAC_{date}_{instance}_south', target=target, polar_region=polar_region)
                pt.plot_ne_and_fac(inactive_day_south.return_data(), name=f'plots/{target_name}Ne_and_FAC_inactive_{date}_{instance}_south', target=target, polar_region=polar_region)
            case 'North':
                pt.plot_ne_and_fac(active_day_north.return_data(), name=f'plots/{target_name}Ne_and_FAC_{date}_{instance}_north', target=target, polar_region=polar_region)
            case 'South':
                pt.plot_ne_and_fac(active_day_south.return_data(), name=f'plots/{target_name}Ne_and_FAC_{date}_{instance}_south', target=target, polar_region=polar_region)

    if plotting_structure_function:
        norm_name = 'Normalized_' if normalize_data else ''
        name = f'{norm_name}Structure_Function_{region}_m={m}_{date}_{instance}'
        fig, axes = plt.subplots(2, figsize=(12, 8), tight_layout=True)

        match comparison:
            case 'NorthSouth':
                north = active_day_north.calculate_structure_function(region=region, seconds=t, m=m, normalize_data=normalize_data)
                south = active_day_south.calculate_structure_function(region=region, seconds=t, m=m, normalize_data=normalize_data)
            case 'ActiveInactiveNorth':
                north = active_day_north.calculate_structure_function(region=region, seconds=t, m=m, normalize_data=normalize_data, name='A')
                #breakpoint()
                north_inactive = inactive_day_north.calculate_structure_function(region=region, seconds=t, m=m, normalize_data=normalize_data)
            case 'ActiveInactiveSouth':
```

```
            south = active_day_south.calculate_structure_function(region=region, seconds=t, m=m, normalize_data=normalize_data)
            south_inactive = inactive_day_south.calculate_structure_function(region=region, seconds=t, m=m, normalize_data=normalize_data)
        case 'all':
            north = active_day_north.calculate_structure_function(region=region, seconds=t, m=m, normalize_data=normalize_data)
            north_inactive = inactive_day_north.calculate_structure_function(region=region, seconds=t, m=m, normalize_data=normalize_data)
            south = active_day_south.calculate_structure_function(region=region, seconds=t, m=m, normalize_data=normalize_data)
            south_inactive = inactive_day_south.calculate_structure_function(region=region, seconds=t, m=m, normalize_data=normalize_data)
        case 'North':
            north = active_day_north.calculate_structure_function(region=region, seconds=t, m=m, normalize_data=normalize_data)
        case 'South':
            south = active_day_south.calculate_structure_function(region=region, seconds=t, m=m, normalize_data=normalize_data)

    if divide_structure_function:
        match comparison:
            case 'NorthSouth':
                north_0 = active_day_north.calculate_structure_function_at_specific_time(start_time=0, stop_time=10)
                south_0 = active_day_south.calculate_structure_function_at_specific_time(start_time=0, stop_time=10)
                north_10 = active_day_north.calculate_structure_function_at_specific_time(start_time=10, stop_time=north['seconds'])
                south_10 = active_day_south.calculate_structure_function_at_specific_time(start_time=10, stop_time=south['seconds'])

            case 'ActiveInactiveNorth':
```

```
                    north_0 = active_day_north.calculate_structure_function_at_specific_time(start_time=0, stop_time=10)
                    north_10 = active_day_north.calculate_structure_function_at_specific_time(start_time=10, stop_time=north['seconds'])
                    north_inactive_0 = inactive_day_north.calculate_structure_function_at_specific_time(start_time=0, stop_time=10)
                    north_inactive_10 = inactive_day_north.calculate_structure_function_at_specific_time(start_time=10, stop_time=north_inactive['seconds'])

                case 'ActiveInactiveSouth':
                    south_0 = active_day_south.calculate_structure_function_at_specific_time(start_time=0, stop_time=10)
                    south_10 = active_day_north.calculate_structure_function_at_specific_time(start_time=10, stop_time=south['seconds'])
                    south_inactive_0 = inactive_day_south.calculate_structure_function_at_specific_time(start_time=0, stop_time=10)
                    south_inactive_10 = inactive_day_south.calculate_structure_function_at_specific_time(start_time=10, stop_time=south_inactive['seconds'])

                case 'all':
                    north_0 = active_day_north.calculate_structure_function_at_specific_time(start_time=0, stop_time=10)
                    south_0 = active_day_south.calculate_structure_function_at_specific_time(start_time=0, stop_time=10)
                    north_inactive_0 = inactive_day_north.calculate_structure_function_at_specific_time(start_time=0, stop_time=10)
                    south_inactive_0 = inactive_day_south.calculate_structure_function_at_specific_time(start_time=0, stop_time=10)
```

```
                north_10 = active_day_north.calculate_structure_function_at_specific_time(start_time=10, stop_time=north['seconds'])
                south_10 = active_day_south.calculate_structure_function_at_specific_time(start_time=10, stop_time=south['seconds'])
                north_inactive_10 = inactive_day_north.calculate_structure_function_at_specific_time(start_time=10, stop_time=north_inactive['seconds'])
                south_inactive_10 = inactive_day_south.calculate_structure_function_at_specific_time(start_time=10, stop_time=south_inactive['seconds'])

            case 'North':
                north_0 = active_day_north.calculate_structure_function_at_specific_time(start_time=0, stop_time=10)
                north_10 = active_day_north.calculate_structure_function_at_specific_time(start_time=10, stop_time=north['seconds'])
            case 'South':
                south_0 = active_day_south.calculate_structure_function_at_specific_time(start_time=0, stop_time=10)
                south_10 = active_day_south.calculate_structure_function_at_specific_time(start_time=10, stop_time=south['seconds'])

        try:
            pt.plot_structure_function(north_0, m, axes, tau_interval=[0, 10], keyword='North')
            pt.plot_structure_function(north_10, m, axes, tau_interval=[10, north['seconds']], keyword='North')
        except UnboundLocalError:
            pass
        try:
            pt.plot_structure_function(south_0, m, axes, tau_interval=[0, 10], keyword='South')
            pt.plot_structure_function(south_10, m, axes, tau_interval=[10, south['seconds']], keyword='South')
        except UnboundLocalError:
```

```
                pass
            try:
                pt.plot_structure_function(north_inactive_0, m, axes, tau_interval=[0, 10], keyword='North Inactive')
                pt.plot_structure_function(north_inactive_10, m, axes, tau_interval=[10, north_inactive['seconds']], keyword='North Inactive')
            except UnboundLocalError:
                pass
            try:
                pt.plot_structure_function(south_inactive_0, m, axes, tau_interval=[0, 10], keyword='South Inactive')
                pt.plot_structure_function(south_inactive_10, m, axes, tau_interval=[10, south_inactive['seconds']], keyword='South Inactive')
            except UnboundLocalError:
                pass

        elif not divide_structure_function:
            try:
                pt.plot_structure_function(north, m, axes, keyword='North')
            except UnboundLocalError:
                pass
            try:
                pt.plot_structure_function(north_inactive, m, axes, keyword='North Inactive')
            except UnboundLocalError:
                pass
            try:
                pt.plot_structure_function(south, m, axes, keyword='South')
            except UnboundLocalError:
                pass
            try:
                pt.plot_structure_function(south_inactive, m, axes, keyword='South Inactive')
            except UnboundLocalError:
                pass

        for ax in axes:
            ax.grid()
        fig.savefig(f'plots/{name}')
        plt.close(fig)
```

```
    if plotting_ratios and not merged_region:
        name = f'Ratio_m={m}_{date}_{instance}'
        fig_ratio, axes_ratio = plt.subplots(3, figsize=(12, 8), tight_layout=True)

        match comparison:
            case 'NorthSouth':
                active_ratio_north = active_day_north.calculate_structure_function_ratios(m=m)
                active_ratio_south = active_day_south.calculate_structure_function_ratios(m=m)
            case 'ActiveInactiveNorth':
                active_ratio_north = active_day_north.calculate_structure_function_ratios(m=m)
                inactive_ratio_north = inactive_day_north.calculate_structure_function_ratios(m=m)
            case 'ActiveInactiveSouth':
                active_ratio_south = active_day_south.calculate_structure_function_ratios(m=m)
                inactive_ratio_south = inactive_day_south.calculate_structure_function_ratios(m=m)
            case 'all':
                active_ratio_north = active_day_north.calculate_structure_function_ratios(m=m)
                inactive_ratio_north = inactive_day_north.calculate_structure_function_ratios(m=m)
                active_ratio_south = active_day_south.calculate_structure_function_ratios(m=m)
                inactive_ratio_south = inactive_day_south.calculate_structure_function_ratios(m=m)
            case 'North':
                active_ratio_north = active_day_north.calculate_structure_function_ratios(m=m)
            case 'South':
                active_ratio_south = active_day_south.calculate_structure_function_ratios(m=m)

        if divide_structure_function:
            match comparison:
                case 'NorthSouth':
                    active_ratio_north_0 = active_day_north.calculate_structure_function_ratios_at_specific_time(0, 10)
                    active_ratio_north_10 = active_day_north.calculate_structure_function_ratios_at_specific_time(10, active_ratio_north['seconds'])
```

```
            active_ratio_south_0 = active_day_south.calculate_structure_function_ratios_at_specific_time(0, 10)
            active_ratio_south_10 = active_day_south.calculate_structure_function_ratios_at_specific_time(10, active_ratio_south['seconds'])
        case 'ActiveInactiveNorth':
            active_ratio_north_0 = active_day_north.calculate_structure_function_ratios_at_specific_time(0, 10)
            active_ratio_north_10 = active_day_north.calculate_structure_function_ratios_at_specific_time(10, active_ratio_north['seconds'])
            inactive_ratio_north_0 = inactive_day_north.calculate_structure_function_ratios_at_specific_time(0, 10)
            inactive_ratio_north_10 = inactive_day_north.calculate_structure_function_ratios_at_specific_time(10, inactive_ratio_north['seconds'])
        case 'ActiveInactiveSouth':
            active_ratio_south_0 = active_day_south.calculate_structure_function_ratios_at_specific_time(0, 10)
            active_ratio_south_10 = active_day_south.calculate_structure_function_ratios_at_specific_time(10, active_ratio_south['seconds'])
            inactive_ratio_south_0 = inactive_day_south.calculate_structure_function_ratios_at_specific_time(0, 10)
            inactive_ratio_south_10 = inactive_day_south.calculate_structure_function_ratios_at_specific_time(10, inactive_ratio_south['seconds'])
        case 'all':
            active_ratio_north_0 = active_day_north.calculate_structure_function_ratios_at_specific_time(0, 10)
```

```
            active_ratio_north_10 = active_day_north.calculate_structure_function_ratios_at_specific_time(10, active_ratio_north['seconds'])
            inactive_ratio_north_0 = inactive_day_north.calculate_structure_function_ratios_at_specific_time(0, 10)
            inactive_ratio_north_10 = inactive_day_north.calculate_structure_function_ratios_at_specific_time(10, inactive_ratio_north['seconds'])
            active_ratio_south_0 = active_day_south.calculate_structure_function_ratios_at_specific_time(0, 10)
            active_ratio_south_10 = active_day_south.calculate_structure_function_ratios_at_specific_time(10, active_ratio_south['seconds'])
            inactive_ratio_south_0 = inactive_day_south.calculate_structure_function_ratios_at_specific_time(0, 10)
            inactive_ratio_south_10 = inactive_day_south.calculate_structure_function_ratios_at_specific_time(10, inactive_ratio_south['seconds'])
        case 'North':
            active_ratio_north_0 = active_day_north.calculate_structure_function_ratios_at_specific_time(0, 10)
            active_ratio_north_10 = active_day_north.calculate_structure_function_ratios_at_specific_time(10, active_ratio_north['seconds'])
        case 'South':
            active_ratio_south_0 = active_day_south.calculate_structure_function_ratios_at_specific_time(0, 10)
            active_ratio_south_10 = active_day_south.calculate_structure_function_ratios_at_specific_time(10, active_ratio_south['seconds'])


    try:
```

```
                pt.plot_structure_function_ratios(active_ratio_north_0, m, axes_ratio, tau_interval=[0, 10], keyword='North')
                pt.plot_structure_function_ratios(active_ratio_north_10, m, axes_ratio, tau_interval=[10, active_ratio_north['seconds']], keyword='North')
            except UnboundLocalError:
                pass
            try:
                pt.plot_structure_function_ratios(inactive_ratio_north_0, m, axes_ratio, tau_interval=[0, 10], keyword='North Inactive')
                pt.plot_structure_function_ratios(inactive_ratio_north_10, m, axes_ratio, tau_interval=[10, inactive_ratio_north['seconds']], keyword='North Inactive')
            except UnboundLocalError:
                pass
            try:
                pt.plot_structure_function_ratios(active_ratio_south_0, m, axes_ratio, tau_interval=[0, 10], keyword='South')
                pt.plot_structure_function_ratios(active_ratio_south_10, m, axes_ratio, tau_interval=[10, active_ratio_south['seconds']], keyword='South')
            except UnboundLocalError:
                pass
            try:
                pt.plot_structure_function_ratios(inactive_ratio_south_0, m, axes_ratio, tau_interval=[0, 10], keyword='South Inactive')
                pt.plot_structure_function_ratios(inactive_ratio_south_10, m, axes_ratio, tau_interval=[10, inactive_ratio_south['seconds']], keyword='South Inactive')
            except UnboundLocalError:
                pass

        elif not divide_structure_function:
            try:
                pt.plot_structure_function_ratios(active_ratio_north, fig_ratio, axes_ratio, keyword='North')
            except UnboundLocalError:
                pass
            try:
```

```python
                pt.plot_structure_function_ratios(inactive_ratio_north, fig_ratio, axes_ratio, keyword='North Inactive')
            except UnboundLocalError:
                pass
            try:
                pt.plot_structure_function_ratios(active_ratio_south, fig_ratio, axes_ratio, keyword='South')
            except UnboundLocalError:
                pass
            try:
                pt.plot_structure_function_ratios(inactive_ratio_south, fig_ratio, axes_ratio, keyword='South Inactive')
            except UnboundLocalError:
                pass

        for ax in axes_ratio:
            ax.grid()
        fig_ratio.savefig(f'plots/{name}')
        plt.close(fig_ratio)

    if plotting_psd:
        dt = 0
        time_interval = None
        p_value = False
        inertial_sub_range = False
        name = f'Power_Spectral_Density_Active_{region}_{date}_{instance}'
        fig_psd, axes_psd = plt.subplots(figsize=(9, 6), tight_layout=True)
        active_psd_north = active_day_north.calculate_power_spectral_density(region, dt=dt, time_interval=time_interval)
        pt.plot_power_spectral_density(active_psd_north, fig_psd, axes_psd, 'High Activity Day', p_value=p_value, region=region, dt=dt, inertial_sub_range=inertial_sub_range)
        axes_psd.grid()
        fig_psd.savefig(f'plots/{name}')
        plt.close(fig_psd)


        name = f'Power_Spectral_Density_Inactive_{region}_{date}_{instance}'
        fig_psd, axes_psd = plt.subplots(figsize=(9, 6), tight_layout=True)
```

```
        inactive_psd_north = inactive_day_north.calculate_power_spectral_density(region, dt=dt, time_interval=time_interval)
        pt.plot_power_spectral_density(inactive_psd_north, fig_psd, axes_psd, 'Low Activity Day', p_value=p_value, color='C1', region=region, dt=dt, inertial_sub_range=inertial_sub_range)
        axes_psd.grid()
        fig_psd.savefig(f'plots/{name}')
        plt.close(fig_psd)

        from scipy.integrate import simpson
        print(simpson(active_psd_north['power_spectral_density'], active_psd_north['frequency']))
        print(simpson(inactive_psd_north['power_spectral_density'], inactive_psd_north['frequency']))

        #active_psd_south = active_day_south.calculate_power_spectral_density(region)
        #inactive_psd_south = inactive_day_south.calculate_power_spectral_density(region)

        #pt.plot_power_spectral_density(active_psd_south, fig_psd, axes_psd, 'South', p_value=True)
        #pt.plot_power_spectral_density(inactive_psd_south, fig_psd, axes_psd, 'South Inactive', p_value=True)

        #active_psd_interval = active_day_north.calculate_power_spectral_density(region, start_time=348, stop_time=355)
        #pt.plot_power_spectral_density(active_psd_interval, fig_psd, axes_psd, 'Active 348-355', p_value=True)

    if plotting_pdf:
        match comparison:
            case 'NorthSouth':
                fig_north_active_pdf, axes_north_active_pdf = plt.subplots(figsize=(9, 6))
                fig_south_active_pdf, axes_south_active_pdf = plt.subplots(figsize=(9, 6))
            case 'ActiveInactiveNorth':
                fig_north_active_pdf, axes_north_active_pdf = plt.subplots(figsize=(9, 6))
                fig_north_inactive_pdf, axes_north_inactive_pdf = plt.subplots(figsize=(9, 6))
            case 'ActiveInactiveSouth':
                fig_south_active_pdf, axes_south_active_pdf = plt.subplots(figsize=(9, 6))
```

```
                fig_south_inactive_pdf, axes_south_inactive_pdf = plt.subplots(figsize=(9, 6))
            case 'all':
                fig_north_active_pdf, axes_north_active_pdf = plt.subplots(figsize=(9, 6))
                fig_north_inactive_pdf, axes_north_inactive_pdf = plt.subplots(figsize=(9, 6))
                fig_south_active_pdf, axes_south_active_pdf = plt.subplots(figsize=(9, 6))
                fig_south_inactive_pdf, axes_south_inactive_pdf = plt.subplots(figsize=(9, 6))
            case 'North':
                fig_north_active_pdf, axes_north_active_pdf = plt.subplots(figsize=(9, 6))
            case 'South':
                fig_south_active_pdf, axes_south_active_pdf = plt.subplots(figsize=(9, 6))


        name_active_north = f'Probability_Density_Fluctuations_North_{region}_{date}_{instance}'
        name_inactive_north = f'Probability_Density_Fluctuations_North_Inactive_{region}_{date}_{instance}'
        name_active_south = f'Probability_Density_Fluctuations_South_{region}_{date}_{instance}'
        name_inactive_south = f'Probability_Density_Fluctuations_South_Inactive_{region}_{date}_{instance}'

        try:
            active_day_north.plot_probability_density_fluctuations(fig_north_active_pdf, axes_north_active_pdf, region, limit=(1E-3, 5), name='High Activity Day')
            axes_north_active_pdf.grid()
            fig_north_active_pdf.savefig(f'plots/{name_active_north}')
            plt.close(fig_north_active_pdf)
        except UnboundLocalError:
            pass
        try:
            inactive_day_north.plot_probability_density_fluctuations(fig_north_inactive_pdf, axes_north_inactive_pdf, region, limit=(1E-3, 5), name='Low Activity Day')
            axes_north_inactive_pdf.grid()
            fig_north_inactive_pdf.savefig(f'plots/{name_inactive_north}')
            plt.close(fig_north_inactive_pdf)
        except UnboundLocalError:
```

```
            pass
        try:
            active_day_south.plot_probability_density_fluctuations(fig_south_active_pdf, axes_south_active_pdf, region, limit=(1E-3, 5), name='High Activity Day')
            axes_south_active_pdf.grid()
            fig_south_active_pdf.savefig(f'plots/{name_active_south}')
            plt.close(fig_south_active_pdf)
        except UnboundLocalError:
            pass
        try:
            inactive_day_south.plot_probability_density_fluctuations(fig_south_inactive_pdf, axes_south_inactive_pdf, region, limit=(1E-3, 5), name='Low Activity Day')
            axes_south_inactive_pdf.grid()
            fig_south_inactive_pdf.savefig(f'plots/{name_inactive_south}')
            plt.close(fig_south_inactive_pdf)
        except UnboundLocalError:
            pass

    if write_to_csv:

        if write_to_pole == 'North':
            active_north = active_day_north.calculate_structure_function(region=region, seconds=t, m=(2, 4), normalize_data=False, calculate_empirical_flatness=False)
            inactive_north = inactive_day_north.calculate_structure_function(region=region, seconds=t, m=(2, 4), normalize_data=False, calculate_empirical_flatness=False)
        elif write_to_pole == 'South':
            active_north = active_day_south.calculate_structure_function(region=region, seconds=t, m=(2, 4), normalize_data=False, calculate_empirical_flatness=False)
            inactive_north = inactive_day_south.calculate_structure_function(region=region, seconds=t, m=(2, 4), normalize_data=False, calculate_empirical_flatness=False)

        try:
            active = {'structure_function_M2': active_north['structure_function'][2],
```

```
                      'structure_function_M4': active_north['structure_function'][4],
                      'empirical_flatness': active_north['empirical_flatness']}
            active_tau = np.asarray(active_north['tau'])
            df_active = pd.DataFrame(active, index=active_tau)
            df_active.to_csv(f'Active_{region_name}_files_{write_to_pole}/{date}_{instance}')
        except TypeError:  # Dummy data frame so both corresponding active and inactive csv files can be loaded simultaneously
            active_dummy = {'structure_function_M2': np.zeros(10),
                            'structure_function_M4': np.zeros(10),
                            'empirical_flatness': np.zeros(10)}
            df_active_dummy = pd.DataFrame(active_dummy, index=np.arange(0, 10))
            df_active_dummy.to_csv(f'Active_{region_name}_files_{write_to_pole}/{date}_{instance}_dummy')

        try:
            inactive = {'structure_function_M2': inactive_north['structure_function'][2],
                        'structure_function_M4': inactive_north['structure_function'][4],
                        'empirical_flatness': inactive_north['empirical_flatness']}

            inactive_tau = np.asarray(inactive_north['tau'])
            df_inactive = pd.DataFrame(inactive, index=inactive_tau)
            df_inactive.to_csv(f'Inactive_{region_name}_files_{write_to_pole}/{date}_{instance}')
        except TypeError:
            inactive_dummy = {'structure_function_M2': np.zeros(10),
                              'structure_function_M4': np.zeros(10),
                              'empirical_flatness': np.zeros(10)}
            df_inactive_dummy = pd.DataFrame(inactive_dummy, index=np.arange(0, 10))
            df_inactive_dummy.to_csv(f'Inactive_{region_name}_files_{write_to_pole}/{date}_{instance}_dummy')

        print(F'DONE {date} {instance} of 3')


if __name__ == '__main__':


    #date = [year, month, day]


```

```
    # All available dates during high activity days
    # Corresponding low activity day is automatically detected
    dates = [[2014, 11, 4],
             [2014, 12, 7],
             [2015, 11, 7],
             [2015, 11, 8],
             [2015, 11, 9],
             [2015, 11, 10],
             [2015, 11, 11],
             [2015, 12, 5],
             [2015, 12, 6],
             [2015, 12, 11],
             [2015, 12, 14],
             [2015, 12, 20],
             [2015, 12, 31]]

    instances = [1, 2, 3]
    region_name = 'B'
    m = (2, 4)
    pole = 'North'
    # comparison = 'all', 'North' 'South' 'NorthSouth'(only active) 'ActiveInactiveNorth' 'ActiveInactiveSouth'
    # polar_region = 'all', 'A', 'B', 'C', 'AB', 'AC', 'BC'
    # polar_region overwritten if target=False
    # polar_region should be 'all' if merged_region = True
    # target = only if plotting_region

    processing_parameters = {'merged_region': True,
                             'region_name': region_name,
                             'tau_interval': 1,
                             'm': 2,
                             'comparison': 'ActiveInactiveNorth',
                             'divide_structure_function': False,
                             'print_time_interval': True,
                             'target': True,
                             'polar_region': 'all',
                             'normalize': False,
                             'write_to_pole': pole}

    #Uncomment and indent to iterate through all dates
    #for date in dates:
    #    for instance in instances:
    run([2015, 12, 31], 1, processing_parameters, plotting_trajectory=False, plotting_region=True, plotting_structure_function=False,
        plotting_ratios=False, plotting_psd=False, plotting_pdf=False, write_to_csv=False)
```

8.5.2 Calculating Slopes and Area

```
import os
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from scipy.stats import linregress
from scipy.integrate import simpson


#Loads calculated data from.csv - files
#Faster to analyse data

region = 'C'

#Instance
i = 4

#Scales (found out too late that m comes before n in the alphabet)
n = 10
m = 10 * 16
j = 100 * 16
k = -1

active_directory = f'Active_{region}_files_North'
inactive_directory = f'Inactive_{region}_files_North'

dates = []

active_slopes_m2_1_10 = []
inactive_slopes_m2_1_10 = []
active_slopes_m2_10_100 = []
inactive_slopes_m2_10_100 = []

active_slopes_m4_1_10 = []
inactive_slopes_m4_1_10 = []
active_slopes_m4_10_100 = []
inactive_slopes_m4_10_100 = []

active_sf_m2 = []
active_sf_m4 = []

inactive_sf_m2 = []
inactive_sf_m4 = []

active_ef = []
inactive_ef = []

```

```python
active_tau = []
inactive_tau = []

for f, g in zip(os.listdir(active_directory), os.listdir(inactive_directory)):
    with open(f'{active_directory}/{f}', 'r') as active_data, open(f'{inactive_directory}/{g}', 'r') as inactive_data:
        active_dataset = pd.read_csv(active_data)
        inactive_dataset = pd.read_csv(inactive_data)

        if not np.all(active_dataset['structure_function_M2']) == 0:

            active_tau.append(active_dataset.index)
            inactive_tau.append(inactive_dataset.index)

            active_sf_m2.append(np.array(active_dataset['structure_function_M2']))
            active_sf_m4.append(active_dataset['structure_function_M4'])

            inactive_sf_m2.append(np.array(inactive_dataset['structure_function_M2']))
            inactive_sf_m4.append(inactive_dataset['structure_function_M4'])

            active_ef.append(active_dataset['empirical_flatness'])
            inactive_ef.append(inactive_dataset['empirical_flatness'])

            dates.append(str(f))

            try:
                active_g_m2_1_10 = linregress(np.log10(active_dataset.index[n:m].astype(float)),
                                              active_dataset['structure_function_M2'][n:m].astype('float'))
                active_slopes_m2_1_10.append(active_g_m2_1_10.slope)
                active_g_m4_1_10 = linregress(np.log10(active_dataset.index[n:m].astype(float)),
                                              active_dataset['structure_function_M4'][n:m].astype('float'))
                active_slopes_m4_1_10.append(active_g_m4_1_10.slope)

```

```
            active_g_m2_10_100 = linregress(np.log10(active_dataset.index[m:j].astype(float)),
                                            active_dataset['structure_function_M2'][m:j].astype('float'))
            active_slopes_m2_10_100.append(active_g_m2_10_100.slope)
            active_g_m4_10_100 = linregress(np.log10(active_dataset.index[m:j].astype(float)),
                                            active_dataset['structure_function_M4'][m:j].astype('float'))
            active_slopes_m4_10_100.append(active_g_m4_10_100.slope)


        except ValueError:
            pass
        try:
            inactive_g_m2_1_10 = linregress(np.log10(inactive_dataset.index[n:m].astype(float)),
                                            inactive_dataset['structure_function_M2'][n:m].astype('float'))
            inactive_slopes_m2_1_10.append(inactive_g_m2_1_10.slope)
            inactive_g_m4_1_10 = linregress(np.log10(inactive_dataset.index[n:m].astype(float)),
                                            inactive_dataset['structure_function_M4'][n:m].astype('float'))
            inactive_slopes_m4_1_10.append(inactive_g_m4_1_10.slope)
            inactive_g_m2_10_100 = linregress(np.log10(inactive_dataset.index[m:j].astype(float)),
```

```
                                               inactive_dataset['structure_function_M2'][m:j].astype('float'))
                inactive_slopes_m2_10_100.append(inactive_g_m2_10_100.slope)
                inactive_g_m4_10_100 = linregress(np.log10(inactive_dataset.index[m:j].astype(float)),
                                               inactive_dataset['structure_function_M4'][m:j].astype('float'))
                inactive_slopes_m4_10_100.append(inactive_g_m4_10_100.slope)
            except ValueError:
                pass

# Uncomment to specify exact instance
# also used to find index for data
# as it is loaded in random order

"""i = 0
for elem in dates:
    if elem == '20151231_1':
        break
    else:
        i+=1
"""
#i=14
print(i)
#Redundant
d = i



def plotting():
    """
    Plots area of structure function by integration
    Both as normal plot high vs low activity and histogram.
    """
    area_active = []
    area_inactive = []

    area_active_100 = []
    area_inactive_100 = []
```

```
    maximas_active = []
    maximas_inactive = []

    maximas_active_100 = []
    maximas_inactive_100 = []

    fig, axes = plt.subplots(figsize=(9, 6), tight_layout=True)

    for x in range(len(active_sf_m2)):

        active_tau_ = active_tau[x][n:m]
        inactive_tau_ = inactive_tau[x][n:m]

        active_tau_100 = active_tau[x][m:j]
        inactive_tau_100 = inactive_tau[x][m:j]

        if x == len(active_sf_m2) - 1:
            axes.plot(active_tau[x][n:m] / 16,
                ↪  active_sf_m2[x][n:m], label='structure function
                ↪  active', color='C0')
            axes.plot(inactive_tau[x][n:m] / 16,
                ↪  inactive_sf_m2[x][n:m], label='structure
                ↪  function inactive',
                      color='C1')
            axes.legend()
        else:
            axes.plot(active_tau[x][n:m] / 16,
                ↪  active_sf_m2[x][n:m], color='C0')
            axes.plot(inactive_tau[x][n:m] / 16,
                ↪  inactive_sf_m2[x][n:m], color='C1')

        area_active.append(simpson(active_sf_m2[x][n:m],
            ↪  active_tau[x][n:m] / 16))
        area_inactive.append(simpson(inactive_sf_m2[x][n:m],
            ↪  inactive_tau[x][n:m] / 16))

        area_active_100.append(simpson(active_sf_m2[x][m:j],
            ↪  active_tau[x][m:j] / 16))
        area_inactive_100.append(simpson(inactive_sf_m2[x][m:j],
            ↪  inactive_tau[x][m:j] / 16))

        axes.set_xscale('log')
        axes.grid()
        axes.set_title(f'Structure Function and Maxima {region}
            ↪  [{int(n / 16)}, {int(m / 16)}]')
        a = active_sf_m2[x][n:m]
        b = inactive_sf_m2[x][n:m]

```

```
        a_100 = active_sf_m2[x][m:j]
        b_100 = inactive_sf_m2[x][m:j]

        maxima_active = np.argmax(a)
        maxima_inactive = np.argmax(b)

        maxima_active_100 = np.argmax(a_100)
        maxima_inactive_100 = np.argmax(b_100)

        maximas_active.append(active_tau_[maxima_active] / 16)
        maximas_inactive.append(inactive_tau_[maxima_inactive] / 16)

        maximas_active_100.append(active_tau_100[maxima_active_100] / 16)
        maximas_inactive_100.append(inactive_tau_100[maxima_inactive_100] / 16)

        axes.scatter(active_tau_[maxima_active] / 16, a[maxima_active], color='C0')
        axes.scatter(inactive_tau_[maxima_inactive] / 16, b[maxima_inactive], color='C1')
    fig.savefig(f'Structure_Function_{region}_{int(n / 16)}_{int(m / 16)}')

    bins = 20

    fig, axes = plt.subplots(2, figsize=(9, 6), tight_layout=True, sharex=False, sharey=True)
    if region != 'All':
        axes[0].set_title(r'a) Area of S(2, $\tau$), $\tau$=[1s, 10s] in' + f' Region {region}', fontsize=24)
        axes[1].set_title(r'b) Area of S(2, $\tau$), $\tau$=[10s, 100s] in' + f' Region {region}', fontsize=24)
    else:
        axes[0].set_title(r'a) Area of S(2, $\tau$), $\tau$=[1s, 10s] in' + ' Polar Region', fontsize=24)
        axes[1].set_title(r'b) Area of S(2, $\tau$), $\tau$=[10s, 100s] in' + ' Polar Region', fontsize=24)
    axes[0].hist(area_active, edgecolor='black', linewidth=1, label=f'High Activity Days', color='C0', bins=bins)
    axes[0].hist(set(area_inactive), edgecolor='black', linewidth=1, label=f'Low Activity Days', color='C1', alpha=0.5, bins=bins)

    axes[1].hist(area_active_100, edgecolor='black', linewidth=1, color='C0', bins=bins)
    axes[1].hist(set(area_inactive_100), edgecolor='black', linewidth=1, color='C1', alpha=0.5, bins=bins)
```

```python
    axes[1].set_xlabel('Area', fontsize=20, labelpad=20)
    axes[0].legend()
    for ax in axes:
        ax.set_ylabel(r'S(2, $\tau$)', fontsize=20, labelpad=20)
        #ax.grid()
        ax.tick_params(axis='both', which='major', labelsize=20)
        ax.legend(fontsize=20)
    fig.savefig(f'Structure_Function_Area_Histogram_{region}')


    fig, axes = plt.subplots(2, figsize=(12, 6), tight_layout=True, sharex=True)
    if region != 'All':
        axes[0].set_title(f'a) Area of Structure Function' + r'$\tau$=[1s, 10s] in ' + f' Region {region}', fontsize=20)
        axes[1].set_title(f'b) Area of Structure Function' + r'$\tau$=[10s, 100s] in ' + f' Region {region}', fontsize=20)
    else:
        axes[0].set_title(f'a) Area of Structure Function' + r'$\tau$=[1s, 10s] in ' + f'Polar Region', fontsize=20)
        axes[1].set_title(f'b) Area of Structure Function' + r'$\tau$=[10s, 100s] in ' + f'Polar Region', fontsize=20)
    axes[0].plot(area_active, label=f'High Activity Days', linewidth=4)
    axes[0].plot(area_inactive, label=f'Low Activity Days', linewidth=4)

    axes[1].plot(area_active_100, linewidth=4)
    axes[1].plot(area_inactive_100, linewidth=4)
    axes[0].legend()
    axes[1].set_xlabel('Instance', fontsize=20, labelpad=20)
    for ax in axes:
        ax.set_ylabel(r'S(2, $\tau$)', fontsize=20, labelpad=20)
        ax.grid()
        ax.tick_params(axis='both', which='major', labelsize=20)
        ax.legend(fontsize=14)
    fig.savefig(f'Structure_Function_Area_Alternate_{region}', dpi=100)


def get_slopes():
    """
    Plot slopes in histogram
    """
    bins = 10
```

```
fig, axes = plt.subplots(2, 2, figsize=(9, 6), tight_layout=True, sharex=True, sharey=True)
axes[0][0].set_title(r'a) High Activity Day $\tau$=' + f'[{n/16}s, {m/16}s] \n' + r'vs $\tau$=' + f'[{m/16}s, {j/16}s]', fontsize=14)
axes[0][0].hist(active_slopes_m4_1_10, edgecolor='black', linewidth=1, label=f'tau=[1s, 10s]', color='C0', bins=bins)
axes[1][0].hist(set(inactive_slopes_m4_1_10), edgecolor='black', linewidth=1, color='C0', bins=bins) # https://docs.python.org/3/library/stdtypes.html#set-types-set-frozenset
axes[1][0].set_title(r'c) Low Activity Day $\tau$=' + f'[{n/16}s, {m/16}s]\n' + r'vs $\tau$=' + f'[{m/16}s, {j/16}s]', fontsize=14)
axes[0][0].hist(active_slopes_m4_10_100, edgecolor='black', linewidth=1, label=f'tau=[10s, 100s]', color='C1', alpha=0.5, bins=bins)
axes[1][0].hist(set(inactive_slopes_m4_10_100), edgecolor='black', linewidth=1, color='C1', alpha=0.5, bins=bins)
axes[0][0].legend()
axes[0][1].set_title(f'b) High Activity Days vs Low Activity Days \n' + r'$\tau$ = ' + f'[{n/16}s, {m/16}s]', fontsize=14)
axes[0][1].hist(active_slopes_m4_1_10, edgecolor='black', linewidth=1, color='C0', bins=bins, label='High Activity Days')
axes[0][1].hist(set(inactive_slopes_m4_1_10), edgecolor='black', linewidth=1, color='C1', bins=bins, alpha=0.5, label='Low Activity Days')
axes[1][1].set_title(f'd) High Activity Days vs Low Activity Days \n' + r'$\tau$ = ' + f'[{m/16}s, {j/16}s]', fontsize=14)
axes[1][1].hist(active_slopes_m4_10_100, edgecolor='black', linewidth=1, label=f'tau=[10s, 100s]', color='C0', bins=bins)
axes[1][1].hist(set(inactive_slopes_m4_10_100), edgecolor='black', linewidth=1, color='C1', bins=bins, alpha=0.5)
axes[0][1].legend()
axes[1][0].set_xlabel(r'Slope', fontsize=14, labelpad=20)
axes[1][0].set_ylabel(r'Counts', fontsize=14, labelpad=20)
for axy in axes:
    for ax in axy:
        ax.tick_params(axis='both', which='major', labelsize=10)
fig.savefig(f'Histogram_{region}', dpi=100)
```

```python
    fig, axes = plt.subplots(figsize=(12, 6), tight_layout=True, sharex=True, sharey=True)
    axes.plot(active_slopes_m2_1_10, color='C0', linewidth=2)
    axes.plot(active_slopes_m4_1_10, color='C0', linewidth=2, ls='dashed')
    axes.plot(inactive_slopes_m2_1_10, color='C1', linewidth=2)
    axes.plot(inactive_slopes_m4_1_10, color='C1', linewidth=2, ls='dashed')
    axes.plot(np.ones(len(active_slopes_m2_1_10)) * np.mean(active_slopes_m2_1_10), color='black', linewidth=2)
    axes.plot(np.ones(len(inactive_slopes_m2_1_10)) * np.mean(inactive_slopes_m2_1_10), color='black', linewidth=2, ls='dotted')
    axes.plot(np.ones(len(active_slopes_m4_1_10)) * np.mean(active_slopes_m4_1_10), color='black', linewidth=2)
    axes.plot(np.ones(len(inactive_slopes_m4_1_10)) * np.mean(inactive_slopes_m4_1_10), color='black', linewidth=2, ls='dotted')
    plt.grid()
    fig.savefig(f'Histogram_{region}', dpi=100)

def plotting_ef():
    """
    plots empirical flatness and area in histogram and as regular plot
    """
    area_active = []
    area_inactive = []

    area_active_100 = []
    area_inactive_100 = []

    bins = 10
    fig, axes = plt.subplots(2, 2, figsize=(12, 6), tight_layout=True)
    for x in range(len(active_ef)):
        active = active_ef[x][n:m]
        inactive = inactive_ef[x][n:m]
        active_100 = active_ef[x][m:j]
        inactive_100 = inactive_ef[x][m:j]
        if x == 1:
            axes[0, 0].plot(active_tau[x][n:m] / 16, active, color='C0', label=f'High Activity Days')
            axes[0, 0].plot(inactive_tau[x][n:m] / 16, inactive, color='C1', label=f'Low Activity Days')
            area_active.append(simpson(active, active_tau[x][n:m] / 16))
```

```
            area_inactive.append(simpson(inactive, inactive_tau[x][n:m] / 16))
            axes[0, 1].plot(active_tau[x][m:j] / 16, active_100, color='C0', label=f'High Activity Days')
            axes[0, 1].plot(inactive_tau[x][m:j] / 16, inactive_100, color='C1', label=f'Low Activity Days')
            area_active_100.append(simpson(active_100, active_tau[x][m:j] / 16))
            area_inactive_100.append(simpson(inactive_100, inactive_tau[x][m:j] / 16))
        else:
            axes[0, 0].plot(active_tau[x][n:m] / 16, active, color='C0')
            axes[0, 0].plot(inactive_tau[x][n:m] / 16, inactive, color='C1')
            area_active.append(simpson(active, active_tau[x][n:m] / 16))
            area_inactive.append(simpson(inactive, inactive_tau[x][n:m] / 16))
            axes[0, 1].plot(active_tau[x][m:j] / 16, active_100, color='C0')
            axes[0, 1].plot(inactive_tau[x][m:j] / 16, inactive_100, color='C1')
            area_active_100.append(simpson(active_100, active_tau[x][m:j] / 16))
            area_inactive_100.append(simpson(inactive_100, inactive_tau[x][m:j] / 16))
    axes[0, 0].plot(active_tau[x][n:m] / 16, np.ones(len(active_tau[x][n:m])) * 3, color='black', linewidth=2, ls='dashed', label=r'F($\tau$) = 3')
    axes[0, 1].plot(active_tau[x][m:j] / 16, np.ones(len(active_tau[x][m:j])) * 3, color='black', linewidth=2, ls='dashed')


    axes[1, 0].hist(area_active, edgecolor='black', linewidth=1, label=f'Active Day {region}', color='C0', bins=bins)
    axes[1, 0].hist(area_inactive, edgecolor='black', linewidth=1, label=f'Inactive Day {region}', color='C1', alpha=0.5, bins=bins)
    axes[1, 1].hist(area_active_100, edgecolor='black', linewidth=1, label=f'Active Day {region}', color='C0', bins=bins)
    axes[1, 1].hist(area_inactive_100, edgecolor='black', linewidth=1, label=f'Inactive Day {region}', color='C1', alpha=0.5, bins=bins)
    axes[0, 0].set_xscale('log')
    axes[0, 1].set_xscale('log')
```

```
    axes[0, 0].grid()
    if region != 'All':
        axes[1, 0].set_title(f'c) Area in Region {region} \n' + r'for $\tau$=' + f'[{n/16}s, {m/16}s]', fontsize=15)
        axes[0, 0].set_title(f'a) Empirical Flatness in Region {region}\n' + r'for $\tau$=' + f'[{n/16}s, {m/16}s]', fontsize=17)
        axes[1, 1].set_title(f'd) Area in Region {region} \n' + r'for $\tau$=' + f'[{n/16}s, {m/16}s]', fontsize=15)
        axes[0, 1].set_title(f'b) Empirical Flatness in Region {region}\n' + r'for $\tau$=' + f'[{m/16}s, {j/16}s]', fontsize=17)
    else:
        axes[1, 0].set_title(f'c) Area in Polar Region \n' + r'for $\tau$=' + f'[{n/16}s, {m/16}s]', fontsize=15)
        axes[0, 0].set_title(f'a) Empirical Flatness in Region {region}\n' + r'for $\tau$=' + f'[{n/16}s, {m/16}s]', fontsize=17)
        axes[1, 1].set_title(f'd) Area in Polar Region \n' + r'for $\tau$=' + f'[{n/16}s, {m/16}s]', fontsize=15)
        axes[0, 1].set_title(f'b) Empirical Flatness in Region {region}\n' + r'for $\tau$=' + f'[{m/16}s, {j/16}s]', fontsize=17)
    axes[0, 0].set_ylabel(r'S(4, $\tau$) / S²(2, $\tau$)', fontsize=14)
    axes[0, 0].set_xlabel(r'$\tau$ (seconds)', fontsize=14)
    axes[0, 1].set_xlabel(r'$\tau$ (seconds)', fontsize=14)
    axes[1, 0].set_ylabel('Counts', fontsize=14)
    axes[1, 0].set_xlabel('Area', fontsize=14, labelpad=20)
    axes[1, 1].set_xlabel('Area', fontsize=14, labelpad=20)
    for axy in axes:
        for ax in axy:
            ax.tick_params(axis='both', which='major', labelsize=14)
    axes[0, 0].legend(fontsize=11)
    fig.savefig(f'Empirical_Flatness_and_Histogram_{region}')

    fig, axes = plt.subplots(2, figsize=(9, 6), tight_layout=True, sharex=True)
    if region != 'All':
        axes[0].set_title(r'a) Area of Empirical Flatness, ' + f'[{n/16}s, {m/16}s] of ' + f'Region {region}', fontsize=18)
        axes[1].set_title(r'b) Area of Empirical Flatness, ' + f'[{m/16}s, {j/16}s] of ' + f'Region {region}', fontsize=18)
    else:
        axes[0].set_title(f'a) Area of Empirical Flatness, [{n/16}s, {m/16}s] of Polar Region', fontsize=18)
```

```
        axes[1].set_title(f'b) Area of Empirical Flatness, [{m / 16}s, {j / 16}s] of Polar Region', fontsize=18)
    axes[0].plot(area_active, label=f'High Activity Days', linewidth=4)
    axes[0].plot(area_inactive, label=f'Low Activity Days', linewidth=4)
    axes[1].plot(area_active_100, linewidth=4)
    axes[1].plot(area_inactive_100, linewidth=4)
    axes[0].legend()
    axes[1].set_xlabel('Instance', fontsize=20, labelpad=20)
    axes[0].plot(np.ones(len(area_active)) * np.mean(area_active), color='C0', ls='dotted', linewidth=4, label='High Mean')
    axes[0].plot(np.ones(len(area_inactive)) * np.mean(area_inactive), color='C1', ls='dotted', linewidth=4, label='Low Mean')
    axes[1].plot(np.ones(len(area_active_100)) * np.mean(area_active_100), color='C0', ls='dotted', linewidth=4)
    axes[1].plot(np.ones(len(area_inactive_100)) * np.mean(area_inactive_100), color='C1', ls='dotted', linewidth=4)
    for ax in axes:
        ax.set_ylabel(r'S(4, $\tau$) / S²(2, $\tau$)', fontsize=16, labelpad=20)
        ax.grid()
        ax.tick_params(axis='both', which='major', labelsize=20)
        ax.legend(fontsize=12)
    fig.savefig(f'Empirical_Fatness_Area_Alternate_{region}', dpi=100)


# Calculates regression at intervals
ga1 = linregress(np.log10(active_tau[i][n:m].astype(float)), active_sf_m2[i][n:m].astype('float'))
ga2 = linregress(np.log10(active_tau[i][m:j].astype(float)), active_sf_m2[i][m:j].astype('float'))
#ga3 = linregress(np.log10(active_tau[i][j:k].astype(float)), active_sf_m2[i][j:k].astype('float'))
gi1 = linregress(np.log10(inactive_tau[i][n:m].astype(float)), inactive_sf_m2[i][n:m].astype('float'))
gi2 = linregress(np.log10(inactive_tau[i][m:j].astype(float)), inactive_sf_m2[i][m:j].astype('float'))
#gi3 = linregress(np.log10(inactive_tau[i][j:k].astype(float)), inactive_sf_m2[i][j:k].astype('float'))



fig, axes = plt.subplots(figsize=(9, 6), tight_layout=True)

```

```python

#Set either of the if blocks to true to plot empirical flatness or strucutre function for specific case
if True:
    #axes.plot(active_tau[d][1:m] / 16, active_ef[d][1:m], color='C0', linewidth=4, label=f'High Activity Day')
    #axes.plot(inactive_tau[d][1:m] / 16, inactive_ef[d][1:m], color='C1', linewidth=4, label=f'Low Activity Day')
    axes.plot(active_tau[d][m:j] / 16, active_ef[d][m:j], color='C0', linewidth=4)
    axes.plot(inactive_tau[d][m:j] / 16, inactive_ef[d][m:j], color='C1', linewidth=4)
    #axes.plot(active_tau[d][j:-1] / 16, active_ef[d][j:-1], color='C0', linewidth=4)
    #axes.plot(inactive_tau[d][j:-1] / 16, inactive_ef[d][j:-1], color='C1', linewidth=4)
    #axes.plot(active_tau[d] / 16, np.ones(len(active_tau[d])) * 3, color='black', linewidth=2, ls='dashed', label=r'S(4, $\tau$) / S$^2$(2, $\tau$) = 3')
    axes.set_ylabel(r'S(4, $\tau$) / S$^2$(2, $\tau$)', fontsize=20, labelpad=20)

if False:
    axes.plot(active_tau[d][n:m] / 16, active_sf_m2[d][n:m], color='C0', linewidth=4, label=f'High Activity Day M2')
    axes.plot(inactive_tau[d][n:m] / 16, inactive_sf_m2[d][n:m], color='C1', linewidth=4, label=f'Low Activity Day M2')
    axes.plot(active_tau[d][m:j] / 16, active_sf_m2[d][m:j], color='C0', linewidth=4)
    axes.plot(inactive_tau[d][m:j] / 16, inactive_sf_m2[d][m:j], color='C1', linewidth=4)
    axes.plot(active_tau[d][j:k] / 16, active_sf_m2[d][j:k], color='C0', linewidth=4)
    axes.plot(inactive_tau[d][j:k] / 16, inactive_sf_m2[d][j:k], color='C1', linewidth=4)

    axes.plot(active_tau[d][k:-1] / 16, active_sf_m2[d][k:-1], color='C0', linewidth=4)
    axes.plot(inactive_tau[d][k:-1] / 16, inactive_sf_m2[d][k:-1], color='C1', linewidth=4)

    #axes.plot(active_tau[d][n:m] / 16, ga1.intercept + ga1.slope * np.log10(active_tau[d][n:m].astype(float)), color='black', linewidth=4, ls='dashed')
    #axes.plot(inactive_tau[d][n:m] / 16, gi1.intercept + gi1.slope * np.log10(inactive_tau[d][n:m].astype(float)), color='black', linewidth=4, ls='dashed')
```

```
    #axes.plot(active_tau[d][m:j] / 16, ga2.intercept + ga2.slope *
        ↪ np.log10(active_tau[d][m:j].astype(float)),
        ↪ color='black', linewidth=4, ls='dashed')
    #axes.plot(inactive_tau[d][m:j] / 16, gi2.intercept + gi2.slope
        ↪ * np.log10(inactive_tau[d][m:j].astype(float)),
        ↪ color='black', linewidth=4, ls='dashed')
    #axes.plot(active_tau[d][j:k] / 16, ga3.intercept + ga3.slope *
        ↪ np.log10(active_tau[d][j:k].astype(float)),
        ↪ color='black', linewidth=4, ls='dashed')
    #axes.plot(inactive_tau[d][j:k] / 16, gi3.intercept + gi3.slope
        ↪ * np.log10(inactive_tau[d][j:k].astype(float)),
        ↪ color='black', linewidth=4, ls='dashed')

    axes.plot(active_tau[d][n:m] / 16, active_sf_m4[d][1:m],
        ↪ color='C2', linewidth=4, label=f'High Activity Day M4')
    axes.plot(inactive_tau[d][n:m] / 16, inactive_sf_m4[d][1:m],
        ↪ color='C3', linewidth=4, label=f'Low Activity Day M4')
    axes.plot(active_tau[d][m:j] / 16, active_sf_m4[d][m:j],
        ↪ color='C2', linewidth=4)
    axes.plot(inactive_tau[d][m:j] / 16, inactive_sf_m4[d][m:j],
        ↪ color='C3', linewidth=4)
    axes.plot(active_tau[d][j:k] / 16, active_sf_m4[d][j:-1],
        ↪ color='C2', linewidth=4)
    axes.plot(inactive_tau[d][j:k] / 16, inactive_sf_m4[d][j:-1],
        ↪ color='C3', linewidth=4)
    """correction_active = np.abs(np.log10((active_tau[d][1] / 16)
        ↪ ** (2 / 3)))
    correction_inactive = np.abs(np.log10((inactive_tau[d][1] / 16)
        ↪ ** (2 / 3)))
    scaling_active = np.log10((active_tau[d][1:n] / 16) ** (2 / 3))
        ↪ + correction_active + active_sf_m2[d][1]
    scaling_inactive = np.log10((inactive_tau[d][1:n] / 16)**(2 /
        ↪ 3)) + correction_inactive + inactive_sf_m2[d][1]
    axes.plot(active_tau[d][1:n] / 16, scaling_active,
        ↪ color='black', ls='dashed', linewidth=2, label=r'Power
        ↪ Law = $\tau^\frac{2}{3}$')
    axes.plot(inactive_tau[d][1:n] / 16, scaling_inactive,
        ↪ color='black', ls='dashed', linewidth=2)
    axes.set_ylabel(r'S(2, $\tau$)', fontsize=20, labelpad=20)"""

if region != 'All':
    axes.set_title(f'Empirical Flatness of Region {region}',
        ↪ fontsize=24)
    axes.set_title(f'Structure Function of Region {region} for
        ↪ m=2', fontsize=24)
else:
    axes.set_title(r'Structure Function of Polar Region for m=2',
        ↪ fontsize=24)
```

```
    axes.set_title(r'Empirical Flatness of Polar Region',
        ↪  fontsize=24)

axes.set_xscale('log')
axes.grid()
axes.set_xlabel(r'$\tau$ (seconds)', fontsize=20, labelpad=20)
axes.tick_params(axis='both', which='major', labelsize=20)
axes.legend(fontsize=14)
fig.savefig('test')

get_slopes()
```

8.5.3 Loading Data From File

```
import os

os.environ["CDF_LIB"] = '/home/sondre/.local/lib'
from spacepy import pycdf
import numpy as np


class GetData:
    """
    Loads data from .CDF files
    Returns dictionary between specified start and stop time
    """
    def __init__(self, start_time, stop_time, measured_data='FAC'):
        self.start_time = start_time
        self.stop_time = stop_time
        main_directory = 'swarm_data'
        sub_directory = start_time.strftime('%Y%m%d')
        file_path = os.path.join(main_directory, sub_directory)
        for f in os.listdir(file_path):
            if 'EXTD' in f:
                plasma_density = f
            elif 'OPER' in f:
                fac = f
        if measured_data == 'Ne':
            filename = os.path.join(file_path, plasma_density)
        elif measured_data == 'FAC':
            filename = os.path.join(file_path, fac)
        self.cdf = pycdf.CDF(filename)
        try:
            self.timestamp = self.cdf['Timestamp'][:]
        except KeyError:
            print('Error: No timestamp')
```

```
    def get_info(self):
        """
        Returns available information
        """
        print(self.cdf)
        return

    def time(self, arr=None):
        """
        Returns dictionary corresponding to specified timestamps.
        If no timestamp, return available timestamp instead
        """
        if arr is None:
            arr = self.timestamp
        idx = np.where((self.start_time <= arr) & (arr <=
            ↪ self.stop_time))[0]
        dict_data = {}
        for key in self.cdf.keys():
            dict_data[key] = self.cdf[key][:][idx]
        return dict_data
```

8.5.4 Detecting Polar (Sub)Region

```
from datetime import timedelta
from functions import normalize, geographic_to_magnetic
import numpy as np


class DetectRegion:
    def __init__(self, dataset_ne, dataset_fac,
        ↪ region_parameters=None, normalize_data=False,
        ↪ magnetic=False):
        """

        :param dataset_ne: dataset for electron density (dictionary)
        :param dataset_fac: dataset for field-aligned currents
            ↪ (dictionary)
        :param region_parameters: threshold, time between peaks,
            ↪ etc... (dictionary)
        :param normalize_data: (Boolean)
        :param magnetic: magnetic coordinates (Boolean)
        """
        if region_parameters is None:
            region_parameters = {'time_interval': 2, 'threshold':
                ↪ 0.5, 'region_num': False, 'total_region': False}
```

```
        self.timestamp_16hz = dataset_ne['Timestamp']
        self.ne = dataset_ne['Density']

        self.timestamp_fac = dataset_fac['Timestamp']
        self.fac = dataset_fac['FAC']

        self.normalize_data = normalize_data

        self.region_parameters = region_parameters

        self.ne_region = self.detect_region(self.ne, self.region_parameters['region_num'])
        self.fac_region = self.detect_region(self.fac, self.region_parameters['region_num'])

        time_interval = {}

        if isinstance(self.ne_region, dict):
            for key in self.ne_region.keys():
                time_interval_region = {}
                time = self.timestamp_16hz[self.ne_region[key] != None]
                time_interval_region['start'] = "{}h {}m {}s".format(np.min(time).hour, np.min(time).minute, np.min(time).second)
                time_interval_region['stop'] = "{}h {}m {}s".format(np.max(time).hour, np.max(time).minute, np.max(time).second)
                time_interval[key] = time_interval_region

        else:
            time = self.timestamp_16hz[self.ne_region != None]
            time_interval['start'] = "{}h {}m {}s".format(np.min(time).hour, np.min(time).minute, np.min(time).second)
            time_interval['stop'] = "{}h {}m {}s".format(np.max(time).hour, np.max(time).minute, np.max(time).second)

        self.time_interval = time_interval


        self.magnetic_coordinates = None
        if magnetic:
            altitude = dataset_fac['Radius'][self.fac_region != None]
            latitude = dataset_fac['Latitude'][self.fac_region != None]
```

```
            longitude = dataset_fac['Longitude'][self.fac_region != None]
            time = dataset_fac['Timestamp'][self.fac_region != None]

            self.magnetic_coordinates = geographic_to_magnetic(altitude, latitude, longitude, time)


    def detect_region(self, arr, region_num):
        """
        Uses the FAC to detect auroral regions and potentially polar cap region.
        Finds each element above threshold value and calculates amount of time between each of said elements.
        Elements below threshold value but inside the time interval are
        added to array along with the elements above the threshold to avoid unnecessary amounts of very small regions.

        :param arr: array of either Ne or FAC.
        :param region_num: Boolean/integer/tuple - which region (from 1 to n) to return. False for all regions (in 1 array)
        :return: region_num = False --> one array containing all regions.
                 region_num = integer --> one array containing one region.
                 region_num = tuple --> three arrays, one between regions in tuple (polar cap) and the other two as these regions
        """
        if isinstance(region_num, tuple) and self.normalize_data:
            self.normalize_data = 'regular_for_polar_cap'
        if arr.shape == self.fac.shape:
            if (arr == self.fac).all:
                self.normalize_data = False  # Only normalizes Ne
        time_diff = []
        datetime_arr = self.timestamp_fac[np.abs(self.fac) >= self.region_parameters['threshold']]
        for i in range(1, len(datetime_arr)):
            diff = datetime_arr[i] - datetime_arr[i - 1]
            if diff <= timedelta(minutes=self.region_parameters['time_interval']):
                time_diff.append(datetime_arr[i])
        if len(arr) == len(self.timestamp_16hz):
            timestamp = self.timestamp_16hz
        elif len(arr) == len(self.timestamp_fac):
            timestamp = self.timestamp_fac
```

```
        new_time_diff = [time_diff[0]]
        for i in range(0, len(time_diff)):
            diff = time_diff[i] - time_diff[i - 1]
            if diff >= timedelta(minutes=2):
                new_time_diff.append(time_diff[i - 1])
                new_time_diff.append(time_diff[i])
            elif i == len(time_diff) - 1:
                new_time_diff.append(time_diff[i])  # Last FAC
                ↪  region have no timedelta to compare with,
                ↪  but knows it's > threshold so is in
                ↪  time_diff
        result = np.zeros(len(timestamp))
        max_range = int(np.ceil(len(new_time_diff) / 2))  # Round up
        for i in range(max_range):
            result[np.where((new_time_diff[2 * i] <= timestamp) &
            ↪  (timestamp <= new_time_diff[2 * i + 1]))] = i +
            ↪  1
        if not region_num:
            if self.normalize_data == 'regular':
                region = np.where(np.isin(result, range(1,
                ↪  max_range + 1)), arr, None)
                region[region != None] = normalize(region)
                return region
            else:
                return np.where(np.isin(result, range(1, max_range
                ↪  + 1)), arr, None)
        elif isinstance(region_num, int):
            if self.normalize_data == 'regular':
                region = np.where(result == region_num, arr, None)
                region[region != None] = normalize(region)
                return region
            else:
                return np.where(result == region_num, arr, None)
        elif isinstance(region_num, tuple):
            region = np.where(np.isin(result, region_num), arr,
            ↪  None)  # Regions m,n and area between
            polar_cap = np.where(np.isin(result, region_num), None,
            ↪  None)  # Only region between m and n
            region_idx = np.where(region != None)
            for idx in range(np.min(region_idx),
            ↪  np.max(region_idx)):
                if region[idx] is None:
                    polar_cap[idx] = arr[idx]
            pre_polar_cap = self.detect_region(arr,
            ↪  region_num=region_num[0])
            post_polar_cap = self.detect_region(arr,
            ↪  region_num=region_num[1])
            if self.normalize_data == 'regular_for_polar_cap':
```

```
                result_new = np.concatenate((pre_polar_cap[pre_polar_cap != None], polar_cap[polar_cap != None], post_polar_cap[post_polar_cap != None]))
                new_normalize = 1 / np.max(result_new)
                pre_polar_cap[pre_polar_cap != None] *= new_normalize
                polar_cap[polar_cap != None] *= new_normalize
                post_polar_cap[post_polar_cap != None] *= new_normalize
            elif self.normalize_data == 'independent':
                pre_polar_cap[pre_polar_cap != None] = normalize(pre_polar_cap)
                polar_cap[polar_cap != None] = normalize(polar_cap)
                post_polar_cap[post_polar_cap != None] = normalize(post_polar_cap)
            if self.region_parameters['total_region']:
                complete_region = polar_cap
                complete_region[pre_polar_cap != None] = pre_polar_cap[pre_polar_cap != None]
                complete_region[post_polar_cap != None] = post_polar_cap[post_polar_cap != None]
                return complete_region
            elif not self.region_parameters['total_region']:
                return {'A': pre_polar_cap, 'B': polar_cap, 'C': post_polar_cap}

    def return_region(self):
        return {'Ne': self.ne_region, 'FAC': self.fac_region, 'time_interval': self.time_interval, 'magnetic_coordinates': self.magnetic_coordinates}
```

8.5.5 Data Processing

```
import matplotlib.pyplot as plt
import detectRegion as dr
from functions import *
import numpy as np
import scipy as sp
from scipy.stats import linregress, norm
import seaborn as sns
from scipy.signal import hann
from mpl_toolkits.basemap import Basemap
from itertools import chain


```

```python
class DataProcessing:
    """
    Class with methods for actually applying our data to our functions.
    Also some more encompassing implementations which would not fit in the functions.py - file
    """
    def __init__(self, data_16Hz, data_FAC, region_parameters, magnetic=False):
        self.ratio_dataset = None
        self.dataset = None
        self.closest_latitudes = None
        self.timestamp_16Hz = data_16Hz['Timestamp']
        self.timestamp_FAC = data_FAC['Timestamp']
        self.latitude_16Hz = data_16Hz['Latitude']
        self.longitude_16Hz = data_16Hz['Longitude']
        self.latitude_FAC = data_FAC['Latitude']
        self.longitude_FAC = data_FAC['Longitude']
        self.ne = data_16Hz['Density']
        self.fac = data_FAC['FAC']

        working_data = dr.DetectRegion(data_16Hz, data_FAC, region_parameters=region_parameters, magnetic=magnetic).return_region()

        ne_region = working_data['Ne']
        fac_region = working_data['FAC']

        time_interval = working_data['time_interval']
        magnetic_coordinates = working_data['magnetic_coordinates']

        self.ne_region = ne_region
        self.fac_region = fac_region

        self.time_interval = time_interval
        self.magnetic_coordinates = magnetic_coordinates

    def calculate_structure_function(self, region='all', seconds='auto', m='all', normalize_data=False, calculate_empirical_flatness=True, name =''):
        """

        :param region: Either All, A, B or C (string)
        :param seconds: chose max time lag (string or int)
        :param m: m-th order structure function.
        Set to 'all' for m=[1, 2, 3, 4] (int, tuple or string)
        :param normalize_data: Unused
        :param calculate_empirical_flatness: Set to False to avoid unecessary computation(Boolean)
```

```
        :param name: (string)
        :return: (dictionary)
        """
        try:
            if region == 'all':
                region = self.ne_region
            elif region == 'A':
                region = self.ne_region['A']
            elif region == 'B':
                region = self.ne_region['B']
            elif region == 'C':
                region = self.ne_region['C']
        except IndexError:
            return self.dataset

        region = region[region != None]
        #region = np.abs(delta_n(region, 1))
        dataset = {}
        slope_dataset = {}
        structure_function_dataset = {}
        regression_dataset = {}

        if seconds == 'auto':
            seconds = len(region)
            tau = np.arange(1, int(seconds))
        elif seconds == 'auto_half':
            seconds = len(region) / 2
            tau = np.arange(1, int(seconds))
        elif isinstance(seconds, int) or isinstance(seconds, float):
            tau = np.arange(1, int(seconds * 16))

        dataset['seconds'] = int(seconds / 16)
        if m == 'all':
            for m in range(1, 5):
                sf = structure_function(region, tau, m)

                sf = np.log10(sf.astype(float))

                #if normalize_data:
                #    sf = normalize(sf)

                g = linregress(np.log10(tau.astype(float)), sf.astype('float'))

                structure_function_dataset[m] = sf
                slope_dataset[m] = g.slope
                regression_dataset[m] = g.intercept + g.slope * np.log10(tau.astype(float))

```

```
            print(f'{m} of 4 complete')

    elif isinstance(m, int):
        sf = structure_function(region, tau, m, name)[0]
        print()

        sf = np.log10(sf.astype(float))

        #if normalize_data:
        #    sf = normalize(sf)

        #g = linregress(np.log10(tau.astype(float)), sf.astype('float'))

        structure_function_dataset[m] = sf
        slope_dataset[m] = 'dummy'#g.slope
        regression_dataset[m] = 'dummy'#g.intercept + g.slope * np.log10(tau.astype(float))

        print('Done')

    elif isinstance(m, tuple):
        for elem in m:
            sf = structure_function(region, tau, elem)

            sf = np.log10(sf.astype(float))

            g = linregress(np.log10(tau.astype(float)), sf.astype('float'))

            structure_function_dataset[elem] = sf
            slope_dataset[elem] = g.slope
            regression_dataset[elem] = g.intercept + g.slope * np.log10(tau.astype(float))

            print('Done')

    del sf

    dataset['structure_function'] = structure_function_dataset

    if calculate_empirical_flatness:
        ef = empirical_flatness(region, tau)

        #if normalize_data:
        #    ef = normalize(ef)

    elif isinstance(m, tuple):
        try:
```

```
                ef = (10 ** dataset['structure_function'][4]) / ((10 ** dataset['structure_function'][2]) ** 2)
            except KeyError:
                ef = None


        else:
            ef = None

        dataset['slope'] = slope_dataset
        dataset['regression'] = regression_dataset
        dataset['empirical_flatness'] = ef
        dataset['tau'] = tau / 16
        self.dataset = dataset
        return dataset

    def calculate_structure_function_at_specific_time(self, start_time, stop_time):
        """
        Returns structure function only calculatedat a specific time scale
        :param start_time: (int)
        :param stop_time: (int)
        :return: (dictionary)
        """
        new_dataset = {}

        new_slope_dataset = {}
        new_structure_function_dataset = {}
        new_regression_dataset = {}

        start_tau = int(start_time * 16) if start_time != 0 else 1
        stop_tau = int(stop_time * 16)

        new_dataset['tau'] = self.dataset['tau'][start_tau:stop_tau]
        if self.dataset['empirical_flatness'] is not None:
            new_dataset['empirical_flatness'] = self.dataset['empirical_flatness'][start_tau:stop_tau]
        for key in self.dataset['structure_function'].keys():
            new_structure_function_dataset[key] = self.dataset['structure_function'][key][start_tau:stop_tau]

            g = linregress(np.log10(new_dataset['tau']).astype('float'), new_structure_function_dataset[key].astype('float'))
            new_regression_dataset[key] = g.intercept + g.slope * np.log10(new_dataset['tau']).astype('float')
```

```
            new_slope_dataset[key] = g.slope

        new_dataset['structure_function'] = new_structure_function_dataset
        new_dataset['slope'] = new_slope_dataset
        new_dataset['regression'] = new_regression_dataset
        new_dataset['seconds'] = stop_time
        return new_dataset

    def calculate_structure_function_ratios(self, seconds='auto_half', m=2):
        """
        Returns ratios between strucutre function in regions A, B and C

        :param seconds: maximum time lag (int or string)
        :param m: m-th order
        :return: (dictionary)
        """
        region = self.ne_region
        region_A = region['A'][region['A'] != None]
        region_B = region['B'][region['B'] != None]
        region_C = region['C'][region['C'] != None]

        if seconds == 'auto':
            region_lengths = [len(region_A), len(region_B), len(region_C)]
            seconds = int(np.min(region_lengths))
            tau = np.arange(1, int(seconds - 1))
        if seconds == 'auto_half':
            region_lengths = [len(region_A), len(region_B), len(region_C)]
            seconds = int(np.min(region_lengths) / 2)
            tau = np.arange(1, int(seconds))
        elif isinstance(seconds, int) or isinstance(seconds, float):
            tau = np.arange(1, int(seconds * 16))

        dataset = {'B/A': structure_function(region_B, tau, m) / structure_function(region_A, tau, m),
                   'B/C': structure_function(region_B, tau, m) / structure_function(region_C, tau, m),
                   'A/C': structure_function(region_A, tau, m) / structure_function(region_C, tau, m),
                   'tau': tau,
                   'seconds': int(seconds / 16)}

        self.ratio_dataset = dataset
        return dataset
```

```
    def calculate_structure_function_ratios_at_specific_time(self, start_time, stop_time):
        """
        Same as calculate_structure_function_at_specific_time
        :param start_time: (int)
        :param stop_time: (int)
        :return: (dictionary)
        """
        new_dataset = {}

        start_tau = int(start_time * 16) if start_time != 0 else 1
        stop_tau = int(stop_time * 16)

        new_dataset['tau'] = self.ratio_dataset['tau'][start_tau:stop_tau]
        new_dataset['B/A'] = self.ratio_dataset['B/A'][start_tau:stop_tau]
        new_dataset['B/C'] = self.ratio_dataset['B/C'][start_tau:stop_tau]
        new_dataset['A/C'] = self.ratio_dataset['A/C'][start_tau:stop_tau]

        new_dataset['seconds'] = stop_time
        return new_dataset

    def calculate_power_spectral_density(self, region='all', dt=0, time_interval=None):
        """

        :param region: Either A, B, C or All (string)
        :param dt: PSD for if dNe/dt (int or float)
        :param time_interval: If a specific time interval is desired (tuple)
        :return: (dictionary)
        """
        if region == 'all':
            region = self.ne_region
        elif region == 'A':
            region = self.ne_region['A']
        elif region == 'B':
            region = self.ne_region['B']
        elif region == 'C':
            region = self.ne_region['C']

        idx = np.where(region != None)
        new_region = region[idx]
        if dt != 0:
            dt *= 16
            dNe = delta_n(new_region, dt)
```

```
            new_region = dNe / new_region[:len(dNe)]


        if time_interval:
            start_time = time_interval[0] * 16
            stop_time = time_interval[1] * 16
            new_region = new_region[start_time: stop_time]

        window = hann(len(new_region))
        new_region = new_region * window

        region_fft = sp.fft.rfft(new_region)  # https://docs.scipy.org/doc/scipy/reference/generated/scipy.fft.rfft.html
        # https://docs.scipy.org/doc/scipy/reference/generated/scipy.fft.fft.html#scipy.fft.fft
        frequency = sp.fft.rfftfreq(len(new_region), d=1/16)

        amplitude = np.log10(np.abs(region_fft) ** 2)

        target = 1E-1
        differences = np.abs(frequency - target)
        start = np.argmin(differences)
        print(start)
        target = 1
        differences = np.abs(frequency - target)
        middle = np.argmin(differences)
        print(middle)
        p_1 = linregress(np.log10(frequency[start:middle]).astype(float), amplitude[start:middle].astype('float'))
        regression_1 = p_1.intercept + p_1.slope * np.log10(frequency[start:middle]).astype('float')

        p_2 = linregress(np.log10(frequency[middle:]).astype(float), amplitude[middle:].astype('float'))
        regression_2 = p_2.intercept + p_2.slope * np.log10(frequency[middle:]).astype('float')

        return {'power_spectral_density': amplitude[start:], 'frequency': frequency[start:], 'slope': (p_1.slope, p_2.slope), 'regression': (regression_1, regression_2)}

    def plot_probability_density_fluctuations(self, fig, axes, region='all', limit=False, name=''):
        """

```

```
        :param fig: as in fig, axes = plt.subplots()
        :param axes: as in fig, axes = plt.subplots()
        :param region: Either A, B, C or All (string)
        :param limit: Limits axes (boolean)
        :param name: unused (string)
        :return: figure of PDF
        """
        region_name = region
        if region == 'A':
            region = self.ne_region['A']
        elif region == 'B':
            region = self.ne_region['B']
        elif region == 'C':
            region = self.ne_region['C']
        elif region == 'all':
            region = self.ne_region

        idx = np.where(region != None)

        #increments = np.array([0.0625, 0.125, 0.25, 0.5, 1]) * 16
        #increments = np.array([1, 5, 10]) * 16
        increments = np.array([10, 50, 100]) * 16
        increments = np.array(increments, dtype=int)

        mean_sets = []
        std_sets = []

        dataset = {}

        for increment in increments:
            dne = delta_n(region[idx], increment)

            data = dne / np.std(dne)

            dataset[f'{increment}'] = data

            sns.kdeplot(data, ax=axes, linewidth=4, ls='dotted',
                ↪ label=r'$\tau$' f'= {increment / 16}s')
            x = np.linspace(np.min(data), np.max(data), len(data))
            mean_sets.append(np.mean(data))
            std_sets.append(np.std(data))

        gaussian = norm.pdf(x, np.mean(mean_sets),
            ↪ np.mean(std_sets))
        axes.plot(x, gaussian, color='black')  # Normal
            ↪ distribution https://stackoverflow.com/questions/10
            ↪ 138085/how-to-plot-normal-distribution
        axes.legend(fontsize=20)
        axes.set_yscale('log')
```

```python
        axes.tick_params(axis='both', which='major', labelsize=20)
        axes.set_ylabel(f'PDF(x)', fontsize=20, labelpad=20)
        axes.set_xlabel(f'x = $\Delta$Ne/$\sigma$($\Delta$Ne)', fontsize=20, labelpad=20)
        if limit:
            axes.set_ylim(limit[0], limit[1])
            axes.set_xlim(-6, 6)
        fig.tight_layout()

    def plot_trajectory(self, name, latitude_limit=0, other_day=False, all_orbits=False):
        """

        :param name: (string)
        :param latitude_limit: limits latitude to avoid having to compute more than necessary (int)
        :param other_day: only used when calculating the trajectory of 2nd day (dictionary)
        :param all_orbits: if all trajectories and not just the closest one is wanted (boolean)
        :return:
        """
        try:
            longitude = self.longitude_FAC[self.fac_region['B'] != None]
            latitude = self.latitude_FAC[self.fac_region['B'] != None]
        except IndexError:
            longitude = self.longitude_FAC[self.fac_region != None]
            latitude = self.latitude_FAC[self.fac_region != None]
        latitude_limit = np.abs(latitude_limit)
        if np.max(latitude) > 0:
            north = True
            viewing_latitude = 50
        else:
            north = False
            viewing_latitude = -50


        # https://jakevdp.github.io/PythonDataScienceHandbook/04.13-geographic-data-with-basemap.html
        def draw_map(m, scale=0.2):
            """
            Code is used from this website
            https://jakevdp.github.io/PythonDataScienceHandbook/04.13-geographic-data-with-basemap.html

            It displays a projection of earth.
            """
```

```
        # draw a shaded-relief image
        m.shadedrelief(scale=scale)

        # lats and longs are returned as a dictionary
        lats = m.drawparallels(np.linspace(-90, 90, 13))
        lons = m.drawmeridians(np.linspace(-180, 180, 13))

        # keys contain the plt.Line2D instances
        lat_lines = chain(*(tup[1][0] for tup in lats.items()))
        lon_lines = chain(*(tup[1][0] for tup in lons.items()))
        all_lines = chain(lat_lines, lon_lines)

        # cycle through these lines and set the desired style
        for line in all_lines:
            line.set(linestyle='-', alpha=0.3, color='w')


    fig = plt.figure(figsize=(12, 8))
    m = Basemap(projection='ortho', resolution=None,
                lon_0=0, lat_0=viewing_latitude)
    draw_map(m)
    longitude = longitude[np.abs(latitude) >= latitude_limit]
    latitude = latitude[np.abs(latitude) >= latitude_limit]
    coordinates = np.column_stack((latitude, longitude))
    x, y = m(longitude, latitude)
    m.scatter(x, y, marker='D', color='m', s=0.5, label='High Activity Trajectory')
    if other_day:
        latitude_other = other_day['Latitude_FAC']
        longitude_other = other_day['Longitude_FAC']

        if north:
            latitude_other_limit = np.where(latitude_other >= latitude_limit, latitude_other, None)
            longitude_other_limit = np.where(latitude_other >= latitude_limit, longitude_other, None)
        else:
            latitude_other_limit = np.where(latitude_other <= latitude_limit, latitude_other, None)
            longitude_other_limit = np.where(latitude_other <= latitude_limit, longitude_other, None)

        # Divides array of latitudes into multiple sub-arrays for each trajectory
        # Then each trajectory's distance from main trajectory is calculated
        # Closest trajectory is registered
        latitude_other_idx = np.where(latitude_other_limit == None)[0]
```

```python
latitude_other_subarrays = np.split(latitude_other_limit, latitude_other_idx)
longitude_other_subarrays = np.split(longitude_other_limit, latitude_other_idx)
latitude_other_new = [np.array(subarray[subarray != None], dtype=np.float64) for subarray in latitude_other_subarrays if len(subarray) > 1]
longitude_other_new = [np.array(subarray[subarray != None], dtype=np.float64) for subarray in longitude_other_subarrays if len(subarray) > 1]
closest_indices_set = []
mean_dist = []
for subarray_lat, subarray_lon in zip(latitude_other_new, longitude_other_new):
    coordinates_other_new = np.column_stack((subarray_lat[subarray_lat != None], subarray_lon[subarray_lon != None]))
    distances = np.linalg.norm(coordinates[:, None, :] - coordinates_other_new, axis=2)
    closest_indices = np.argmin(distances, axis=1)
    closest_indices_set.append(closest_indices)
    coordinates_other_new_2 = coordinates_other_new[closest_indices]
    distance = np.abs(coordinates - coordinates_other_new_2).sum()
    mean_dist.append(distance)
for latitude_other_subarray, longitude_other_subarray, closest_indices, i in zip(latitude_other_new, longitude_other_new, closest_indices_set, range(len(mean_dist))):
    if i == np.argmin(mean_dist):
        self.closest_latitudes = latitude_other_subarray[closest_indices]
        x_other, y_other = m(longitude_other_subarray[closest_indices], latitude_other_subarray[closest_indices])
        m.scatter(x_other, y_other, marker='D', color='black', s=0.5, label='Closest Low Activity Trajectory')
    else:
        if all_orbits:
            x_other, y_other = m(longitude_other_subarray[closest_indices], latitude_other_subarray[closest_indices])
```

```
                        m.scatter(x_other, y_other, marker='D',
                            ↪  color='grey', s=0.5)
        plt.legend(fontsize=14, markerscale=14, loc='upper right',
            ↪  bbox_to_anchor=(1.4, 1.1))
        plt.savefig(f'{name}')
        plt.close()

    def find_closest_region(self, other_day):
        """

        :param other_day: you need two days to find closest regions
            ↪  (dictionary)
        :return: Timestamps at the start and end of the closest
            ↪  trajectory(string)
        """
        try:
            idx_start = np.where([other_day['Latitude_FAC'] ==
                ↪  self.closest_latitudes[0]])[-1][0]
            idx_stop = np.where([other_day['Latitude_FAC'] ==
                ↪  self.closest_latitudes[-1]])[-1][0]
            print(other_day['Timestamp_FAC'][idx_start])
            print(other_day['Timestamp_FAC'][idx_stop])
        except TypeError:
            print('closest trajectory for previous/next day not
                ↪  calculated, or region_num parameter in
                ↪  other_day is not set to false')

    def return_data(self):
        """
        Classes won't return anything on their own
        :return: (dictionary)
        """
        return {'Density_Full': self.ne, 'FAC_Full': self.fac,
                'Density': self.ne_region, 'FAC': self.fac_region,
                'Latitude_FAC': self.latitude_FAC, 'Longitude_FAC':
                    ↪  self.longitude_FAC,
                'Latitude_16Hz': self.latitude_16Hz,
                    ↪  'Longitude_16Hz': self.longitude_16Hz,
                'Timestamp_FAC': self.timestamp_FAC,
                    ↪  'Timestamp_16Hz': self.timestamp_16Hz,
                'time_interval': self.time_interval,
                'magnetic_coordinates': self.magnetic_coordinates}
```

8.5.6 Functions

```python
import numpy as np
from spacepy.coordinates import Coords
from spacepy.time import Ticktock


def mean_square(y, n):
    """
    Mean square function
    :param y: Electron Density (array)
    :param n: window size (int)
    :return: MS (array)
    """
    y = np.array(y, dtype=np.float64)
    window = np.ones(n) / n
    return np.convolve(y ** 2, window)


def root_mean_square(y, n):
    """
    Root mean square function
    :param y: Electron Density (array)
    :param n: window size (int)
    :return: RMS (array)
    """
    return np.sqrt(mean_square(y, n))


def _structure_function(y, tau, m=2):
    """
    Structure Function A
    Fast but inaccurate
    Vectorized
    :param y: Electron Density (array)
    :param tau: time lag (array)
    :param m: structure function order (int)
    :return: structure function (array)
    """
    t = np.arange(len(y) - np.max(tau))
    y_diff = np.abs(y[t + tau[:, None]] - y[t])
    return np.mean(np.array(y_diff) ** m, axis=1)


def structure_function(y, tau, m=2):
    """
    Structure Function A
    Slow but accurate
```

```
    Partially Vectorized
    :param y: Electron Density (array)
    :param tau: time lag (array)
    :param m: structure function order (int)
    :return: structure function (array)
    """
    t = np.arange(0, len(y) - 1)
    y_difference = []
    for n in range(0, len(tau)):
        #Loop necessary to always use maximum available data points
        y_tau_shifted = y[(t[:(len(t) - n)] + tau[:, None])[n]]
        y_original_time = y[t[:(len(t) - n)]]
        y_difference.append(np.mean(np.abs(y_tau_shifted - y_original_time) ** m))
    return np.array(y_difference)


def empirical_flatness(y, tau):
    """

    :param y: Electron Density (array)
    :param tau: time lag (array)
    :return: empirical flatness (array)
    """
    # tau must not start at zero
    return structure_function(y, tau, 4) / structure_function(y, tau, 2) ** 2


def empirical_flatness_alt(m4, m2):
    """
    Alternate empirical flatness so structure function calculations
    doesn't have to be repeated
    :param m4: strucuture function for m=4 (array)
    :param m2: strucuture function for m=2 (array)
    :return: empirical flatness (array)
    """
    return m4 / m2 ** 2


def normalize(y):
    """

    :param y: array
    :return: normalized by dividing maximum value (array)
    """
    y_norm = y[y != None] / np.max(y[y != None])
    return y_norm
```

```python

def delta_n(n, dt):
    """
    dN/dt
    :param n: electron density (array)
    :param dt: time shift (int)
    :return: (array)
    """
    if dt == 0:
        return n
    dt = int(dt)
    t = np.arange(len(n) - dt)
    return n[t + dt] - n[t]


# https://stackoverflow.com/questions/7948450/conversion-from-geographic-to-geomagnetic-coordinates
def geographic_to_magnetic(altitude, latitude, longitude, time):
    """
    Slow
    :param altitude: int
    :param latitude: int
    :param longitude: int
    :param time: datetime
    :return: array
    """
    data = np.array([altitude, latitude, longitude])
    data = np.squeeze(data, axis=1)

    cvals = Coords(data.T, 'GEO', 'sph')

    new_time = time.astype(np.datetime64)
    even_newer_time = np.squeeze(np.datetime_as_string(new_time),
        axis=0)
    cvals.ticks = Ticktock(even_newer_time, 'UTC')
    print('ok')
    return cvals.convert('MAG', 'sph')
```

8.5.7 Plotting Basic Figures

```python
import matplotlib.pyplot as plt
import numpy as np


def plot_ne_and_fac_(data, name, target=False, polar_region='all'):
    """
```

```
    plots electron density and
    field-aligned currents

    :param data: (dictionary)
    :param name: (string)
    :param target: entire region or just specified polar region (boolean)
    :param polar_region: if target is True,
    can choose one or more subregions
    (region A, B, C) (string)
    :return: figure
    """
    fig, axes = plt.subplots(2, figsize=(9, 6), tight_layout=True, sharex=True)
    if not target:
        axes[0].plot(data['Timestamp_16Hz'], data['Density_Full'])
        axes[1].plot(data['Timestamp_FAC'], data['FAC_Full'])

    if isinstance(data['FAC'], dict):
        if polar_region == 'A' or polar_region == 'AC' or polar_region == 'AB' or polar_region == 'all':
            axes[0].plot(data['Timestamp_16Hz'], data['Density']['A'], label='A')
            axes[1].plot(data['Timestamp_FAC'], data['FAC']['A'])
        if polar_region == 'B' or polar_region == 'AB' or polar_region == 'BC' or polar_region == 'all':
            axes[0].plot(data['Timestamp_16Hz'], data['Density']['B'], label='B')
            axes[1].plot(data['Timestamp_FAC'], data['FAC']['B'])
        if polar_region == 'C' or polar_region == 'AC' or polar_region == 'BC' or polar_region == 'all':
            axes[0].plot(data['Timestamp_16Hz'], data['Density']['C'], label='C')
            axes[1].plot(data['Timestamp_FAC'], data['FAC']['C'])

    else:
        axes[0].plot(data['Timestamp_16Hz'], data['Density'], label='Polar Region')
        axes[1].plot(data['Timestamp_FAC'], data['FAC'], label='Polar Region')

    for ax in axes:
        ax.grid()
        ax.legend()
    axes[0].set_title('a) Plasma Density', fontsize=16)
    axes[0].set_ylabel('$cm^{-3}$', fontsize=14)
    axes[1].set_xlabel('Time', fontsize=14)
    axes[1].set_ylabel('A/$m^2$', fontsize=14)
    axes[1].set_title('b) Field-Aligned Current', fontsize=16)
```

```
    fig.savefig(f'{name}')
    plt.close(fig)


def plot_structure_function(data, m, axes, tau_interval='tau', keyword=''):
    """
    Commented out scaling exponent plot

    :param data: (dictionary)
    :param m: m-th order (tuple or int)
    :param axes: as in fig, axes = plt.subplots()
    :param tau_interval: for labeling (string)
    :param keyword: for labeling (string)
    :return: figure
    """
    if isinstance(m, tuple):
        for elem in m:
            axes[0].plot(data['tau'], data['structure_function'][elem], label=f'S({elem}, {tau_interval}) {keyword}')
            axes[0].plot(data['tau'], data['regression'][elem], ls='dotted', c='black')
            # axes[2].scatter(m, data['slope'][elem], label=f'm = {elem}')
            slope = data['slope'][elem]
            print(f'Slope = {slope}')

    elif isinstance(m, int):
        axes[0].plot(data['tau'], data['structure_function'][m], label=f'{keyword} S({m}, {tau_interval}) {keyword}')
        axes[0].plot(data['tau'], data['regression'][m], ls='dotted', c='black')
        #axes[2].scatter(m, data['slope'][m], label=f'm = {m}')
        slope = data['slope'][m]
        print(f'Slope = {slope}')

    axes[0].legend(bbox_to_anchor=(1.0, 1.0), prop={'size': 6})
    axes[0].legend(prop={'size': 6})
    axes[0].set_xscale('log')

    axes[1].plot(data['tau'], data['empirical_flatness'])
    axes[1].set_xscale('log')


def plot_structure_function_ratios(data, m, axes, limit=False, tau_interval='tau', keyword=''):
    """
```

```
    :param data: (dictionary)
    :param m: m-th order (tuple or int)
    :param axes: as in fig, axes = plt.subplots()
    :param limit: optional tuple if axes should be shared (tuple)
    :param tau_interval: for labeling (string)
    :param keyword: for labeling (string)
    :return:
    """
    axes[0].plot(data['tau'] / 16, data['B/A'], label=f'{keyword} S({m}, {tau_interval}) B/A')
    axes[1].plot(data['tau'] / 16, data['B/C'], label=f'{keyword} S({m}, {tau_interval}) B/C')
    axes[2].plot(data['tau'] / 16, data['A/C'], label=f'{keyword} S({m}, {tau_interval}) A/C')
    for ax in axes:
        ax.grid()
        ax.set_xscale('log')
        ax.legend()
        if limit:
            ax.set_ylim(0, limit)


def plot_power_spectral_density(data, fig, axes, label='', region='all', p_value=False, color='C0', dt='', inertial_sub_range=False):
    """

    :param data: (dictionary)
    :param fig: as in fig, axes = plt.subplots()
    :param axes: as in fig, axes = plt.subplots()
    :param label: (string)
    :param region: Either region A, B, C or All (string)
    :param p_value: displays p value of regression (boolean)
    :param color: (string)
    :param dt: displays dt if used (int)
    :param inertial_sub_range: fits kolmogorov scaling exponent k/³ (boolean)
    :return:
    """
    psd = data['power_spectral_density']
    frequency = data['frequency']
    if region == 'all':
        axes.set_title(f'Power Spectral Density in Polar Region', fontsize=24)
    else:
        axes.set_title(f'Power Spectral Density in Region {region}, dt={dt}s', fontsize=24)
    if inertial_sub_range:
```

```
        axes.plot(frequency, np.log10(frequency**(5 / 3)) * psd, label=label, color=color)
    else:
        axes.plot(frequency, psd, label=label, color=color)
        correction = - np.log10(np.abs(frequency**(-5 / 3)))[0]
        axes.plot(frequency, np.log10(frequency**(-5 / 3)) + correction + np.abs(psd[0]), label=r'k^{-5/3}', color='black')
        if p_value:
            p_1 = data['slope'][0]
            p_2 = data['slope'][1]
            print(f'1st p value = {p_1}')
            print(f'2nd p value = {p_2}')
            axes.plot(frequency[:len(data['regression'][0])], data['regression'][0], ls='dotted', c='black')
            axes.plot(frequency[len(data['regression'][0]):], data['regression'][1], ls='dotted', c='black')
    axes.set_xscale('log')
    axes.set_xlabel('Frequency (Hz)', fontsize=20, labelpad=20)
    axes.set_ylabel('P(f)/(Hz)', fontsize=20, labelpad=20)
    axes.set_xlim(1E-1, 1E1)
    axes.tick_params(axis='both', which='major', labelsize=20)
    axes.legend(fontsize=20)
    fig.tight_layout()
```

8.5.8 Choosing Instance

```
from datetime import datetime


def load_day(year, month, day, instance, merged_region):
    """
    Loads one of the days (cases)
    set merged_region to True to use the entire polar region

    :param year: Int
    :param month: Int
    :param day: Int
    :param instance: Int
    :param merged_region: Boolean
    :return:
    """

    dataset = {}

    if year == 2014 and month == 11 and day == 4:
```

```
        time_interval_start = 6
        time_interval_stop = 18
        day_inactive = 3
        match instance:
            case 1:
                fac_parameters_north = {"time_interval": 2, "threshold": 1, "region_num": (10, 11), 'total_region': merged_region}
                fac_parameters_north_inactive = {"time_interval": 2, "threshold": 0.5, "region_num": (11, 12), 'total_region': merged_region}
                fac_parameters_south = {"time_interval": 2, "threshold": 1, "region_num": (8, 9), 'total_region': merged_region}
                fac_parameters_south_inactive = {"time_interval": 2, "threshold": 0.5, "region_num": (9, 10), 'total_region': merged_region}
            case 2:
                fac_parameters_north = {"time_interval": 2, "threshold": 1, "region_num": (18, 19), 'total_region': merged_region}
                fac_parameters_north_inactive = {"time_interval": 2, "threshold": 0.5, "region_num": (19, 20), 'total_region': merged_region}
                fac_parameters_south = {"time_interval": 2, "threshold": 1.2, "region_num": (16, 17), 'total_region': merged_region}
                fac_parameters_south_inactive = {"time_interval": 2, "threshold": 0.5, "region_num": (17, 18), 'total_region': merged_region}
            case 3:
                fac_parameters_north = {"time_interval": 2, "threshold": 1, "region_num": (22, 23), 'total_region': merged_region}
                fac_parameters_north_inactive = {"time_interval": 2, "threshold": 0.5, "region_num": (23, 24), 'total_region': merged_region}
                fac_parameters_south = {"time_interval": 2, "threshold": 1, "region_num": (20, 21), 'total_region': merged_region}
                fac_parameters_south_inactive = {"time_interval": 2, "threshold": 0.5, "region_num": (21, 22), 'total_region': merged_region}

    if year == 2014 and month == 12 and day == 7:
        time_interval_start = 12
        time_interval_stop = 18
        day_inactive = 6
        match instance:
```

```
            case 1:
                fac_parameters_north = {"time_interval": 2, "threshold": 1, "region_num": (2, 3), 'total_region': merged_region}
                fac_parameters_north_inactive = {"time_interval": 3, "threshold": 3, "region_num": (5, 6), 'total_region': merged_region}
                fac_parameters_south = {"time_interval": 2, "threshold": 2, "region_num": (1, 2), 'total_region': merged_region}
                fac_parameters_south_inactive = {"time_interval": 3, "threshold": 0.5, "region_num": 2, 'total_region': merged_region}
            case 2:
                fac_parameters_north = {"time_interval": 2, "threshold": 1, "region_num": (5, 6), 'total_region': merged_region}
                fac_parameters_north_inactive = {"time_interval": 3, "threshold": 3, "region_num": (8, 9), 'total_region': merged_region}
                fac_parameters_south = {"time_interval": 2, "threshold": 6, "region_num": (5, 6), 'total_region': merged_region}
                fac_parameters_south_inactive = {"time_interval": 3, "threshold": 2, "region_num": 5, 'total_region': merged_region}
            case 3:
                fac_parameters_north = {"time_interval": 2, "threshold": 1, "region_num": (8, 9), 'total_region': merged_region}
                fac_parameters_north_inactive = {"time_interval": 3, "threshold": 3, "region_num": (12, 13), 'total_region': merged_region}
                fac_parameters_south = {"time_interval": 2, "threshold": 3, "region_num": (9, 10), 'total_region': merged_region}
                fac_parameters_south_inactive = {"time_interval": 3, "threshold": 2, "region_num": (8, 9), 'total_region': merged_region}

    if year == 2015 and month == 11 and day == 7:
        time_interval_start = 4
        time_interval_stop = 18
        day_inactive = 2
        match instance:
            case 1:
                fac_parameters_north = {"time_interval": 2, "threshold": 1, "region_num": (4, 5), 'total_region': merged_region}
```

```
                fac_parameters_north_inactive = {"time_interval": 2, "threshold": 0.5, "region_num": (4, 5), 'total_region': merged_region}
                fac_parameters_south = {"time_interval": 10, "threshold": 5, "region_num": (4, 5), 'total_region': merged_region}
                fac_parameters_south_inactive = {"time_interval": 2, "threshold": 2, "region_num": (2, 3), 'total_region': merged_region}
            case 2:
                fac_parameters_north = {"time_interval": 2, "threshold": 0.4, "region_num": (8, 9), 'total_region': merged_region}
                fac_parameters_north_inactive = {"time_interval": 2, "threshold": 0.5, "region_num": (7, 8), 'total_region': merged_region}
                fac_parameters_south = {"time_interval": 2, "threshold": 2, "region_num": (6, 7), 'total_region': merged_region}
                fac_parameters_south_inactive = {"time_interval": 2, "threshold": 1, "region_num": (7, 8), 'total_region': merged_region}
            case 3:
                fac_parameters_north = {"time_interval": 2, "threshold": 0.4, "region_num": (11, 12), 'total_region': merged_region}
                fac_parameters_north_inactive = {"time_interval": 2, "threshold": 0.5, "region_num": (10, 11), 'total_region': merged_region}
                fac_parameters_south = {"time_interval": 2, "threshold": 2, "region_num": (10, 11), 'total_region': merged_region}
                fac_parameters_south_inactive = {"time_interval": 2, "threshold": 0.5, "region_num": 9, 'total_region': merged_region}

    if year == 2015 and month == 11 and day == 8:
        time_interval_start = 10
        time_interval_stop = 18
        day_inactive = 12
        match instance:
            case 1:
                fac_parameters_north = {"time_interval": 2, "threshold": 0.5, "region_num": (8, 9), 'total_region': merged_region}
                fac_parameters_north_inactive = {"time_interval": 2, "threshold": 0.5, "region_num": (5, 6), 'total_region': merged_region}
```

```
                fac_parameters_south = {"time_interval": 2,
                    ↪ "threshold": 0.5, "region_num": (6, 7),
                    ↪ 'total_region': merged_region}
                fac_parameters_south_inactive = {"time_interval":
                    ↪ 2, "threshold": 0.5, "region_num": (3, 4),
                    ↪ 'total_region': merged_region}
            case 2:
                fac_parameters_north = {"time_interval": 2,
                    ↪ "threshold": 0.5, "region_num": (12, 13),
                    ↪ 'total_region': merged_region}
                fac_parameters_north_inactive = {"time_interval":
                    ↪ 2, "threshold": 0.5, "region_num": (9, 10),
                    ↪ 'total_region': merged_region}
                fac_parameters_south = {"time_interval": 2,
                    ↪ "threshold": 0.5, "region_num": (10, 11),
                    ↪ 'total_region': merged_region}
                fac_parameters_south_inactive = {"time_interval":
                    ↪ 2, "threshold": 0.5, "region_num": (7, 8),
                    ↪ 'total_region': merged_region}
            case 3:
                fac_parameters_north = {"time_interval": 2,
                    ↪ "threshold": 0.5, "region_num": (16, 17),
                    ↪ 'total_region': merged_region}
                fac_parameters_north_inactive = {"time_interval":
                    ↪ 2, "threshold": 0.5, "region_num": (13,
                    ↪ 14), 'total_region': merged_region}
                fac_parameters_south = {"time_interval": 2,
                    ↪ "threshold": 0.5, "region_num": (14, 15),
                    ↪ 'total_region': merged_region}
                fac_parameters_south_inactive = {"time_interval":
                    ↪ 2, "threshold": 0.5, "region_num": (11,
                    ↪ 12), 'total_region': merged_region}

    if year == 2015 and month == 11 and day == 9:
        time_interval_start = 6
        time_interval_stop = 18
        day_inactive = 12
        match instance:
            case 1:
                fac_parameters_north = {"time_interval": 2,
                    ↪ "threshold": 0.5, "region_num": (16, 17),
                    ↪ 'total_region': merged_region}
                fac_parameters_north_inactive = {"time_interval":
                    ↪ 2, "threshold": 0.5, "region_num": (13,
                    ↪ 14), 'total_region': merged_region}
                fac_parameters_south = {"time_interval": 2,
                    ↪ "threshold": 0.5, "region_num": (14, 15),
                    ↪ 'total_region': merged_region}
```

```
                fac_parameters_south_inactive = {"time_interval": 2, "threshold": 0.5, "region_num": (11, 12), 'total_region': merged_region}
            case 2:
                fac_parameters_north = {"time_interval": 2, "threshold": 0.5, "region_num": (20, 21), 'total_region': merged_region}
                fac_parameters_north_inactive = {"time_interval": 2, "threshold": 0.5, "region_num": (17, 18), 'total_region': merged_region}
                fac_parameters_south = {"time_interval": 2, "threshold": 0.5, "region_num": (18, 19), 'total_region': merged_region}
                fac_parameters_south_inactive = {"time_interval": 2, "threshold": 0.5, "region_num": (15, 16), 'total_region': merged_region}
            case 3:
                fac_parameters_north = {"time_interval": 2, "threshold": 0.5, "region_num": (24, 25), 'total_region': merged_region}
                fac_parameters_north_inactive = {"time_interval": 2, "threshold": 0.5, "region_num": (21, 22), 'total_region': merged_region}
                fac_parameters_south = {"time_interval": 2, "threshold": 0.5, "region_num": (22, 23), 'total_region': merged_region}
                fac_parameters_south_inactive = {"time_interval": 2, "threshold": 0.5, "region_num": (19, 20), 'total_region': merged_region}

    if year == 2015 and month == 11 and day == 10:
        time_interval_start = 8
        time_interval_stop = 16
        day_inactive = 12
        match instance:
            case 1:
                fac_parameters_north = {"time_interval": 2, "threshold": 0.5, "region_num": (2, 3), 'total_region': merged_region}
                fac_parameters_north_inactive = {"time_interval": 2, "threshold": 0.5, "region_num": (2, 3), 'total_region': merged_region}
                fac_parameters_south = {"time_interval": 2, "threshold": 1, "region_num": (5, 6), 'total_region': merged_region}
                fac_parameters_south_inactive = {"time_interval": 2, "threshold": 0.5, "region_num": (4, 5), 'total_region': merged_region}
            case 2:
```

```
            fac_parameters_north = {"time_interval": 2, "threshold": 0.5, "region_num": (10, 11), 'total_region': merged_region}
            fac_parameters_north_inactive = {"time_interval": 2, "threshold": 0.5, "region_num": (10, 11), 'total_region': merged_region}
            fac_parameters_south = {"time_interval": 2, "threshold": 0.5, "region_num": (12, 13), 'total_region': merged_region}
            fac_parameters_south_inactive = {"time_interval": 2, "threshold": 0.5, "region_num": (12, 13), 'total_region': merged_region}
        case 3:
            fac_parameters_north = {"time_interval": 2, "threshold": 0.5, "region_num": (14, 15), 'total_region': merged_region}
            fac_parameters_north_inactive = {"time_interval": 2, "threshold": 0.5, "region_num": (14, 15), 'total_region': merged_region}
            fac_parameters_south = {"time_interval": 2, "threshold": 0.5, "region_num": (16, 17), 'total_region': merged_region}
            fac_parameters_south_inactive = {"time_interval": 2, "threshold": 0.5, "region_num": (16, 17), 'total_region': merged_region}

if year == 2015 and month == 11 and day == 11:
    time_interval_start = 6
    time_interval_stop = 18
    day_inactive = 12
    match instance:
        case 1:
            fac_parameters_north = {"time_interval": 2, "threshold": 0.5, "region_num": (17, 18), 'total_region': merged_region}
            fac_parameters_north_inactive = {"time_interval": 2, "threshold": 0.5, "region_num": (17, 18), 'total_region': merged_region}
            fac_parameters_south = {"time_interval": 2, "threshold": 0.5, "region_num": (15, 16), 'total_region': merged_region}
            fac_parameters_south_inactive = {"time_interval": 2, "threshold": 0.5, "region_num": (15, 16), 'total_region': merged_region}
        case 2:
            fac_parameters_north = {"time_interval": 2, "threshold": 0.5, "region_num": (23, 24), 'total_region': merged_region}
```

```
                fac_parameters_north_inactive = {"time_interval": 2, "threshold": 0.5, "region_num": (21, 22), 'total_region': merged_region}
                fac_parameters_south = {"time_interval": 2, "threshold": 0.5, "region_num": (21, 22), 'total_region': merged_region}
                fac_parameters_south_inactive = {"time_interval": 2, "threshold": 0.5, "region_num": (19, 20), 'total_region': merged_region}
            case 3:
                fac_parameters_north = {"time_interval": 2, "threshold": 0.5, "region_num": (27, 28), 'total_region': merged_region}
                fac_parameters_north_inactive = {"time_interval": 2, "threshold": 0.5, "region_num": (24, 25), 'total_region': merged_region}
                fac_parameters_south = {"time_interval": 2, "threshold": 0.5, "region_num": (25, 26), 'total_region': merged_region}
                fac_parameters_south_inactive = {"time_interval": 10, "threshold": 0.5, "region_num": 24, 'total_region': merged_region}

    if year == 2015 and month == 12 and day == 5:
        time_interval_start = 8
        time_interval_stop = 22
        day_inactive = 3
        match instance:
            case 1:
                fac_parameters_north = {"time_interval": 2, "threshold": 0.5, "region_num": (7, 8), 'total_region': merged_region}
                fac_parameters_north_inactive = {"time_interval": 2, "threshold": 0.5, "region_num": (6, 7), 'total_region': merged_region}
                fac_parameters_south = {"time_interval": 2, "threshold": 0.5, "region_num": (5, 6), 'total_region': merged_region}
                fac_parameters_south_inactive = {"time_interval": 2, "threshold": 0.5, "region_num": (4, 5), 'total_region': merged_region}
            case 2:
                fac_parameters_north = {"time_interval": 2, "threshold": 0.5, "region_num": (20, 21), 'total_region': merged_region}
                fac_parameters_north_inactive = {"time_interval": 2, "threshold": 0.6, "region_num": (20, 21), 'total_region': merged_region}
```

```python
                fac_parameters_south = {"time_interval": 1, "threshold": 2, "region_num": (20, 21), 'total_region': merged_region}
                fac_parameters_south_inactive = {"time_interval": 2, "threshold": 0.1, "region_num": 18, 'total_region': merged_region}
            case 3:
                fac_parameters_north = {"time_interval": 2, "threshold": 0.5, "region_num": 23, 'total_region': merged_region}
                fac_parameters_north_inactive = {"time_interval": 2, "threshold": 0.5, "region_num": (22, 23), 'total_region': merged_region}
                fac_parameters_south = {"time_interval": 2, "threshold": 0.5, "region_num": 22, 'total_region': merged_region}
                fac_parameters_south_inactive = {"time_interval": 2, "threshold": 0.5, "region_num": 21, 'total_region': merged_region}

    if year == 2015 and month == 12 and day == 6:
        time_interval_start = 8
        time_interval_stop = 22
        day_inactive = 4
        match instance:
            case 1:
                fac_parameters_north = {"time_interval": 2, "threshold": 0.5, "region_num": (9, 10), 'total_region': merged_region}
                fac_parameters_north_inactive = {"time_interval": 2, "threshold": 0.5, "region_num": (9, 10), 'total_region': merged_region}
                fac_parameters_south = {"time_interval": 2, "threshold": 0.5, "region_num": (7, 8), 'total_region': merged_region}
                fac_parameters_south_inactive = {"time_interval": 2, "threshold": 1, "region_num": (7, 8), 'total_region': merged_region}
            case 2:
                fac_parameters_north = {"time_interval": 2, "threshold": 0.5, "region_num": (13, 14), 'total_region': merged_region}
                fac_parameters_north_inactive = {"time_interval": 2, "threshold": 0.5, "region_num": (13, 14), 'total_region': merged_region}
                fac_parameters_south = {"time_interval": 2, "threshold": 0.5, "region_num": (11, 12), 'total_region': merged_region}
```

```
                fac_parameters_south_inactive = {"time_interval": 2, "threshold": 0.5, "region_num": (11, 12), 'total_region': merged_region}
            case 3:
                fac_parameters_north = {"time_interval": 2, "threshold": 1, "region_num": (21, 22), 'total_region': merged_region}
                fac_parameters_north_inactive = {"time_interval": 2, "threshold": 0.5, "region_num": (19, 20), 'total_region': merged_region}
                fac_parameters_south = {"time_interval": 2, "threshold": 1, "region_num": (19, 20), 'total_region': merged_region}
                fac_parameters_south_inactive = {"time_interval": 2, "threshold": 0.5, "region_num": 18, 'total_region': merged_region}

    if year == 2015 and month == 12 and day == 11:
        time_interval_start = 10
        time_interval_stop = 22
        day_inactive = 4
        match instance:
            case 1:
                fac_parameters_north = {"time_interval": 2, "threshold": 1, "region_num": (15, 16), 'total_region': merged_region}
                fac_parameters_north_inactive = {"time_interval": 2, "threshold": 1.2, "region_num": (17, 18), 'total_region': merged_region}
                fac_parameters_south = {"time_interval": 2, "threshold": 2, "region_num": (16, 17), 'total_region': merged_region}
                fac_parameters_south_inactive = {"time_interval": 2, "threshold": 0.2, "region_num": 13, 'total_region': merged_region}
            case 2:
                fac_parameters_north = {"time_interval": 2, "threshold": 1, "region_num": (18, 19), 'total_region': merged_region}
                fac_parameters_north_inactive = {"time_interval": 2, "threshold": 1.2, "region_num": (19, 20), 'total_region': merged_region}
                fac_parameters_south = {"time_interval": 2, "threshold": 1, "region_num": 17, 'total_region': merged_region}
                fac_parameters_south_inactive = {"time_interval": 2, "threshold": 0.2, "region_num": (15, 16), 'total_region': merged_region}
            case 3:
```

```
                fac_parameters_north = {"time_interval": 2, "threshold": 1, "region_num": (21, 22), 'total_region': merged_region}
                fac_parameters_north_inactive = {"time_interval": 2, "threshold": 1, "region_num": (23, 24), 'total_region': merged_region}
                fac_parameters_south = {"time_interval": 2, "threshold": 1, "region_num": 20, 'total_region': merged_region}
                fac_parameters_south_inactive = {"time_interval": 2, "threshold": 0.2, "region_num": 20, 'total_region': merged_region}

    if year == 2015 and month == 12 and day == 14:
        time_interval_start = 8
        time_interval_stop = 14
        day_inactive = 3
        match instance:
            case 1:
                fac_parameters_north = {"time_interval": 2, "threshold": 0.5, "region_num": (5, 6), 'total_region': merged_region}
                fac_parameters_north_inactive = {"time_interval": 2, "threshold": 0.5, "region_num": (6, 7), 'total_region': merged_region}
                fac_parameters_south = {"time_interval": 2, "threshold": 0.5, "region_num": (3, 4), 'total_region': merged_region}
                fac_parameters_south_inactive = {"time_interval": 2, "threshold": 0.5, "region_num": (4, 5), 'total_region': merged_region}
            case 2:
                fac_parameters_north = {"time_interval": 2, "threshold": 0.5, "region_num": (8, 9), 'total_region': merged_region}
                fac_parameters_north_inactive = {"time_interval": 2, "threshold": 0.5, "region_num": (10, 11), 'total_region': merged_region}
                fac_parameters_south = {"time_interval": 2, "threshold": 1, "region_num": (7, 8), 'total_region': merged_region}
                fac_parameters_south_inactive = {"time_interval": 2, "threshold": 0.5, "region_num": (8, 9), 'total_region': merged_region}
            case 3:
                fac_parameters_north = {"time_interval": 2, "threshold": 0.5, "region_num": (11, 12), 'total_region': merged_region}
```

```
            fac_parameters_north_inactive = {"time_interval": 2, "threshold": 0.5, "region_num": (13, 14), 'total_region': merged_region}
            fac_parameters_south = {"time_interval": 2, "threshold": 0.5, "region_num": 10, 'total_region': merged_region}
            fac_parameters_south_inactive = {"time_interval": 2, "threshold": 0.5, "region_num": 12, 'total_region': merged_region}

if year == 2015 and month == 12 and day == 20:
    time_interval_start = 8
    time_interval_stop = 14
    day_inactive = 19
    match instance:
        case 1:
            fac_parameters_north = {"time_interval": 2, "threshold": 0.5, "region_num": (5, 6), 'total_region': merged_region}
            fac_parameters_north_inactive = {"time_interval": 2, "threshold": 0.5, "region_num": (5, 6), 'total_region': merged_region}
            fac_parameters_south = {"time_interval": 2, "threshold": 1, "region_num": (3, 4), 'total_region': merged_region}
            fac_parameters_south_inactive = {"time_interval": 2, "threshold": 0.5, "region_num": 4, 'total_region': merged_region}
        case 2:
            fac_parameters_north = {"time_interval": 2, "threshold": 0.5, "region_num": (8, 9), 'total_region': merged_region}
            fac_parameters_north_inactive = {"time_interval": 2, "threshold": 0.5, "region_num": (9, 10), 'total_region': merged_region}
            fac_parameters_south = {"time_interval": 2, "threshold": 1, "region_num": (7, 8), 'total_region': merged_region}
            fac_parameters_south_inactive = {"time_interval": 2, "threshold": 1, "region_num": (8, 9), 'total_region': merged_region}
        case 3:
            fac_parameters_north = {"time_interval": 2, "threshold": 0.5, "region_num": (11, 12), 'total_region': merged_region}
            fac_parameters_north_inactive = {"time_interval": 2, "threshold": 0.3, "region_num": (13, 14), 'total_region': merged_region}
```

```
                fac_parameters_south = {"time_interval": 2, "threshold": 1, "region_num": (11, 12), 'total_region': merged_region}
                fac_parameters_south_inactive = {"time_interval": 2, "threshold": 0.5, "region_num": (11, 12), 'total_region': merged_region}

    if year == 2015 and month == 12 and day == 31:
        time_interval_start = 8
        time_interval_stop = 18
        day_inactive = 30
        match instance:
            case 1:
                fac_parameters_north = {"time_interval": 2, "threshold": 1, "region_num": (6, 7), 'total_region': merged_region}
                fac_parameters_north_inactive = {"time_interval": 2, "threshold": 0.5, "region_num": (7, 8), 'total_region': merged_region}
                fac_parameters_south = {"time_interval": 2, "threshold": 5, "region_num": (5, 6), 'total_region': merged_region}
                fac_parameters_south_inactive = {"time_interval": 2, "threshold": 0.5, "region_num": 6, 'total_region': merged_region}
            case 2:
                fac_parameters_north = {"time_interval": 2, "threshold": 0.5, "region_num": (9, 10), 'total_region': merged_region}
                fac_parameters_north_inactive = {"time_interval": 2, "threshold": 0.5, "region_num": (10, 11), 'total_region': merged_region}
                fac_parameters_south = {"time_interval": 2, "threshold": 3, "region_num": (8, 9), 'total_region': merged_region}
                fac_parameters_south_inactive = {"time_interval": 2, "threshold": 0.5, "region_num": 9, 'total_region': merged_region}
            case 3:
                fac_parameters_north = {"time_interval": 2, "threshold": 0.5, "region_num": (12, 13), 'total_region': merged_region}
                fac_parameters_north_inactive = {"time_interval": 2, "threshold": 0.5, "region_num": (13, 14), 'total_region': merged_region}
                fac_parameters_south = {"time_interval": 2, "threshold": 1, "region_num": (11, 12), 'total_region': merged_region}
```

```python
            fac_parameters_south_inactive = {"time_interval": 2, "threshold": 0.5, "region_num": 12, 'total_region': merged_region}

    dataset['FAC_parameters_north'] = fac_parameters_north
    dataset['FAC_parameters_north_inactive'] = fac_parameters_north_inactive
    dataset['FAC_parameters_south'] = fac_parameters_south
    dataset['FAC_parameters_south_inactive'] = fac_parameters_south_inactive

    day_start = datetime(year, month, day, time_interval_start, 00, 00, 00)
    day_stop = datetime(year, month, day, time_interval_stop, 00, 00, 00)

    day_start_inactive = datetime(year, month, day_inactive, time_interval_start, 00, 00, 00)
    day_stop_inactive = datetime(year, month, day_inactive, time_interval_stop, 00, 00, 00)

    date = day_start.strftime('%Y%m%d')
    date_inactive = day_start_inactive.strftime('%Y%m%d')

    dataset['day_start'] = day_start
    dataset['day_stop'] = day_stop

    dataset['day_start_inactive'] = day_start_inactive
    dataset['day_stop_inactive'] = day_stop_inactive

    dataset['date'] = date
    dataset['date_inactive'] = date_inactive

    return dataset
```