FACULTY OF SCIENCE AND TECHNOLOGY

MASTER THESIS

Study programme / specialisation:
Engineering Structures and Materials /
Civil and Offshore Structures

The spring semester, 2022

Open / ~~Confidential~~

Author: Namrata Singh

N. Singh
.....
(signature author)

Course coordinator: Sudath Siriwardane

Supervisor(s): Jungao Wang (UiS)
Hugo Goncalo Antunes Moreira (Statens Vegvesen)

Thesis title: Evaluation of environmental conditions in a Norwegian fjord based on field measurements

Credits (ECTS): 30

Keywords:

Bjørnafjorden
Measurement data
Wind characteristics
Wind spectra
Coherence
Atmospheric stability
Python

Pages: 120

+ appendix: 56

Stavanger, 15.06.2022

Approved by the Dean 30 Sep 21
Faculty of Science and Technology

Abstract

As part of the project “Ferry-free E39”, a 5 km long floating bridge is planned to be built in Bjørnafjorden to replace the existing ferry connection. The demanding nature of the fjord and the sensitivity of the bridge to wind and wave excitations make the stability of the structure under harsh environmental conditions important. This thesis focuses on evaluating the environmental conditions in the fjord based on wind measurements in the area, by studying the effect of mean wind speed, mean wind direction and atmospheric stability on the various wind characteristics in the fjord. A total of 289 wind events were identified from 2015 to 2020 by selecting a threshold for the mean wind speed of 6 m/s. These wind events were used to study the wind characteristics, including angle of attack, turbulence intensity, wind spectra and coherence. The wind spectrum and coherence model from N400 were fitted to the wind measurements, and the fitted parameters were compared to the suggested values in the code.

Overall results indicate that the complex topography of the fjord surrounding the wind masts significantly affects the wind characteristics. The main directions for the wind flow in the fjord were found to be from the north-west, south-west and south-east, which all correspond to wind flow over water with long fetch. As the mean wind speed increases, the suggested values in N400 represents the wind characteristics better compared to lower wind speeds. For wind flow over land, it was observed that N400 tends to underestimate the turbulence intensity and spectral parameters, while for wind flow over water it tends to overestimate these same values. For the coherence, N400 consistently underestimates the coherence in the horizontal plane and overestimates the coherence in the vertical plane. The distribution of the fitted parameters with atmospheric stability was difficult to interpret, but it was concluded that the stability only affected the distribution of the spectral parameter, and not the coherence parameter.

Preface

This master thesis is the final work of the master's degree in Engineering Structures and Materials at the University of Stavanger. The thesis is written in collaboration with the Norwegian Public Roads Administration. The work has been challenging, but very educational, and I am grateful for being given the opportunity to work on this topic which I had an interest in.

I would like to sincerely thank my supervisors Dr. Jungao Wang and Hugo Goncalo Antunes Moreira for their guidance and patience in numerous meetings and e-mails throughout the semester. I am very grateful for the time they spent in helping to continuously improve my work, the advice, and help with any questions related to Python.

Finally, I would like to thank my family and friends for their endless support and motivation.

Stavanger, 15.06.2022
Namrata Singh

Contents

ABSTRACT ..... **I**

PREFACE ..... **II**

CONTENTS ..... **III**

LIST OF FIGURES ..... **VI**

LIST OF TABLES ..... **X**

1. INTRODUCTION **1**

1.1 Background 1

1.2 Objectives 2

2. THEORY **4**

2.1 Wind 4

2.1.1 Mean wind 5

2.1.1.1 Logarithmic profile 6

2.1.1.2 Power law 7

2.1.1.3 Mean wind profile in Eurocode 1 8

2.1.2 Turbulence 9

2.1.3 Friction velocity 11

2.1.4 Integral length scales 11

2.2 Atmospheric stability 14

2.3 Wind spectra 15

2.3.1 Kaimal spectrum 17

2.3.2 N400 17

2.3.3 Harris spectrum 18

2.3.4 Busch-Panofsky spectrum 18

2.3.5 Frøya spectrum 18

2.4 Coherence 19
2.5 Signal processing 21
2.5.1 Fourier analysis 21
2.5.2 Welch's method 23
2.5.3 Correlation and autocorrelation 23
2.5.4 Transformation matrix 25
3. WIND MEASUREMENTS AT BJØRNAFJORDEN 27
3.1 Topography 27
3.2 Wind masts 28
4. METHODOLOGY 29
4.1 Identification of wind events 29
4.2 Data processing and visualization 29
4.3 Spectrum fitting 31
4.4 Coherence fitting 32
5. RESULTS AND DISCUSSION 34
5.1 Angle of attack 36
5.2 Turbulence intensity 46
5.2.1 Directional distribution 46
5.2.2 Distribution with mean wind speed 49
5.3 Atmospheric stability 58
5.4 Friction velocity 62
5.5 Length scales 65
5.6 Spectral models 67
5.6.1 Fitted N400 spectrum 70
5.6.1.1 Distribution of the fitted parameters 73
5.6.1.2 Distribution with atmospheric stability 85

5.7 Coherence parameters 89
5.7.1 Distribution of the fitted parameters 90
5.7.1.1 Lateral turbulence 90
5.7.1.2 Vertical turbulence 97
5.7.2 Distribution with atmospheric stability 103
6. CONCLUSION 106
REFERENCES 108
APPENDIX 1
A: Python scripts 1
A1 Identification of wind events 1
A2 Data processing 14
A3 Integral length scale estimation 22
A4 Spectra fitting 27
A5 Coherence fitting 40
B: Distribution of fitted parameters with stability 49
B1 Spectral parameters 49
B2 Coherence parameters 52

List of Figures

Figure 1.1: The coastal highway route E39 on the western coast of Norway from Kristiansand to Trondheim. [1] 1
Figure 2.1: Mean wind speed and turbulence profile along the mean wind direction. [4] 4
Figure 2.2: Comparison between the logarithmic profile and the power law ($z_0 = 0.02$, $z_{ref} =$ 50 m and $\alpha = 0.128$). [4] 8
Figure 2.3: A time series of wind measurements showing the separation of the fluctuations from the mean wind. [4] 9
Figure 2.4: The autocorrelation function for a time series of the longitudinal turbulence component. The red marker indicates the first zero crossing, which in this case occurs at a time lag of 617. 13
Figure 2.5: Influence of each parameter on the shape of the wind spectrum when all other parameters are fixed at their median value. The horizontal axis represents the frequency, and the vertical axis represents the spectrum normalized by the standard deviation. [9] 16
Figure 2.6: An example of different realizations of the same stochastic process. [19] 24
Figure 2.7: Properties of the autocorrelation function. [19] 25
Figure 3.1: Overview of wind mast locations and topography in Bjørnafjorden. [3] 27
Figure 4.1: Evolution of a wind event over 48 hours on October 22, 2018 30
Figure 5.1: Directional distribution of measured mean wind speed with number of 1-hour events for each range. 34
Figure 5.2: Angle of attack at Ospøya 1. Contour plot is taken from Norgeskart. [27] 37
Figure 5.3: Distribution of angle of attack with direction and mean wind speed at Osp1 38
Figure 5.4: Angle of attack at Ospøya 2. Contour plot taken from Norgeskart. [27] 39
Figure 5.5: Distribution of angle of attack with direction and mean wind speed at Osp2 40
Figure 5.6: Angle of attack at Svarvhelleholmen. Contour plot taken from Norgeskart. [27] 41
Figure 5.7: Distribution of angle of attack with direction and mean wind speed at Svar. 42
Figure 5.8: Angle of attack at Synnøytangen. Contour plot taken from Norgeskart. [27] 43
Figure 5.9: Distribution of angle of attack with direction and mean wind speed at Synn. 44
Figure 5.10: Comparison of angle of attack between all sensors at Osp1. 45
Figure 5.11: Comparison of angle of attack between all sensors at Osp2. 45
Figure 5.12: Comparison of angle of attack between all sensors at Svar. 45
Figure 5.13: Comparison of angle of attack between all sensors at Synn. 46

Figure 5.14: Directional distribution of the longitudinal turbulence intensity for the four top sensors. 47
Figure 5.15: Directional distribution of the lateral turbulence intensity for the four top sensors. 48
Figure 5.16: Directional distribution of the vertical turbulence intensity for the four top sensors. 49
Figure 5.17: Distribution of the longitudinal turbulence intensity with mean wind speed at Osp1. 50
Figure 5.18: Distribution of the longitudinal turbulence intensity with mean wind speed at Svar. 51
Figure 5.19: Distribution of the longitudinal turbulence intensity with mean wind speed at Synn. 52
Figure 5.20: Distribution of the lateral turbulence intensity with mean wind speed at Osp1.. 53
Figure 5.21: Distribution of the lateral turbulence intensity with mean wind speed at Svar... 54
Figure 5.22: Distribution of the lateral turbulence intensity with mean wind speed at Synn.. 55
Figure 5.23: Distribution of the vertical turbulence intensity with mean wind speed at Osp1. 56
Figure 5.24: Distribution of the vertical turbulence intensity with mean wind speed at Svar. 57
Figure 5.25: Distribution of the vertical turbulence intensity with mean wind speed at Synn. 58
Figure 5.26: Variation of atmospheric stability with mean wind speed and direction at Osp1, Svar and Synn, ... 59
Figure 5.27: Distribution of atmospheric stability with mean wind speed at Osp1..... 60
Figure 5.28: Distribution of atmospheric stability with mean wind speed at Svar..... 61
Figure 5.29: Distribution of atmospheric stability with mean wind speed at Synn..... 62
Figure 5.30: Variation of friction velocity in relation to the longitudinal standard deviation. 63
Figure 5.31: Distribution of the turbulence ratio with mean wind speed from the four top sensors. 64
Figure 5.32: Distribution of the turbulence ratio from the four top sensors. 64
Figure 5.33:Variation of the turbulence ratio with stability at Osp1. 65
Figure 5.34: Comparison of length scales at Ospøya 1 and 2. 66
Figure 5.35: Variation of length scales with wind speed at Osp1, Svar and Synn. 66
Figure 5.36: Comparison of spectral models on October 23, 2018, at Osp1. 68
Figure 5.37: Comparison of spectral models on October 23, 2018, at Osp2. 68

Figure 5.38: Comparison of spectral models on October 23, 2018, at Svar. 69
Figure 5.39: Comparison of spectral models on October 23, 2018, at Synn. 69
Figure 5.40: Comparison of fitted N400 spectra on October 23, 2018, at Osp1. 70
Figure 5.41: Comparison of fitted N400 spectra on October 23, 2018, at Osp2. 71
Figure 5.42: Comparison of fitted N400 spectra on October 23, 2018, at Svar. 71
Figure 5.43: Comparison of fitted N400 spectra on October 23, 2018, at Synn. 72
Figure 5.44: Comparison of fitted N400 spectra on January 3, 2020, at Svar. 72
Figure 5.45: Distribution of the fitted A_u with mean wind speed at Osp1 74
Figure 5.46: Distribution of the fitted A_u with mean wind speed at Svar. 75
Figure 5.47: Distribution of the fitted A_u with mean wind speed at Synn. 76
Figure 5.48: Distribution of the fitted A_v with mean wind speed at Osp1 77
Figure 5.49: Distribution of the fitted A_v with mean wind speed at Svar. 78
Figure 5.50: Distribution of the fitted A_v with mean wind speed at Synn. 79
Figure 5.51: Distribution of the fitted A_w with mean wind speed at Osp1. 80
Figure 5.52: Distribution of the fitted A_w with mean wind speed at Svar. 81
Figure 5.53: Distribution of the fitted A_w with mean wind speed at Synn. 82
Figure 5.54: Distribution of the fitted A_u with sectors at Osp1. 83
Figure 5.55: Distribution of the fitted A_u with sectors at Svar. 83
Figure 5.56: Distribution of the fitted A_u with sectors at Synn. 83
Figure 5.57: Distribution of the fitted A_v with sectors at Osp1. 84
Figure 5.58: Distribution of the fitted A_v with sectors at Svar. 84
Figure 5.59: Distribution of the fitted A_v with sectors at Synn. 84
Figure 5.60: Distribution of the fitted A_w with sectors at Osp1 85
Figure 5.61: Distribution of the fitted A_w with sectors at Svar. 85
Figure 5.62: Distribution of the fitted A_w with sectors at Synn. 85
Figure 5.63: Distribution of the fitted A_u with stability at Osp1. 86
Figure 5.64: Distribution of the fitted A_u with stability at Svar. 87
Figure 5.65: Distribution of the fitted A_u with stability at Synn. 87
Figure 5.66: Distribution of the fitted A_u with stability in the eastern sector at Osp1, Svar and Synn. 88
Figure 5.67: Comparison of fitted and reference coherence model on October 23, 2018, at Osp1. 89
Figure 5.68: Comparison of fitted and reference coherence model on October 23, 2018, at Osp2. 90

Figure 5.69: Distribution of the fitted C_{uy} with mean wind speed at Osp1. 91
Figure 5.70: Distribution of the fitted C_{uy} with mean wind speed at Osp2. 92
Figure 5.71: Distribution of the fitted C_{uy} with sectors at Osp1 and Osp2. 92
Figure 5.72: Distribution of the fitted C_{vy} with mean wind speed at Osp1. 93
Figure 5.73: Distribution of the fitted C_{vy} with mean wind speed at Osp2. 94
Figure 5.74: Distribution of the fitted C_{vy} with sectors at Osp1 and Osp2. 94
Figure 5.75: Distribution of the fitted C_{wy} with mean wind speed at Osp1. 95
Figure 5.76: Distribution of the fitted C_{wy} with mean wind speed at Osp2. 96
Figure 5.77: Distribution of the fitted C_{wy} with sectors at Osp1 and Osp2. 96
Figure 5.78: Distribution of the fitted C_{uz} with mean wind speed at Osp1 97
Figure 5.79: Distribution of the fitted C_{uz} with mean wind speed at Osp2 98
Figure 5.80: Distribution of the fitted C_{uz} with sectors at Osp1 and Osp2. 98
Figure 5.81: Distribution of the fitted C_{vz} with mean wind speed at Osp1 99
Figure 5.82: Distribution of the fitted C_{vz} with mean wind speed at Osp2 100
Figure 5.83: Distribution of the fitted C_{vz} with sectors at Osp1 and Osp2. 100
Figure 5.84: Distribution of the fitted C_{wz} with mean wind speed at Osp1. 101
Figure 5.85: Distribution of the fitted C_{wz} with mean wind speed at Osp2. 102
Figure 5.86: Distribution of the fitted C_{wz} with sectors at Osp1 and Osp2. 102
Figure 5.87: Distribution of the fitted C_{uy} with stability at Osp1. 103
Figure 5.88: Distribution of the fitted C_{uy} with stability at Osp2. 104
Figure 5.89: Distribution of the fitted C_{uy} with stability in the eastern sector at Osp1 and Osp2. 105

List of Tables

Table 2.1: Terrain categories and roughness lengths in Eurocode 1. [6].....6
Table 3.1: Information for sonic anemometers. [3].....28
Table 5.1: Overview of considered mean wind speed ranges and number of 1-hour events in each range. 35
Table 5.2: Overview of considered directional sectors and number of 1-hour events in each sector. 35
Table 5.3: Overview of considered stability cases and number of 1-hour events in each case.. S1-S2, S3-S4 and S5-S6 represent unstable, near-neutral, and stable conditions respectively. 35
Table 5.4: Percentage of disregarded data according to selected criteria. 35

1. Introduction

This thesis evaluates the environmental conditions at a Norwegian fjord where a 5 km long floating bridge is planned to be built, based on the wind measurements from 2015 to 2020.

1.1 Background

As of today, travelling from Kristiansand in the south of Norway to Trondheim using the coastal highway route E39 takes approximately 21 hours to complete and requires the use of seven different ferry connections to be able to cross the fjords. The route covers a distance of approximately 1100 km and connects cities such as Stavanger, Bergen and Ålesund, among others. The project "Ferry-free E39" aims to improve the route through the construction of bridges and tunnels on the fjord crossings to remove the need for travel by ferry, and is Norway's largest and most ambitious infrastructure project with an estimated cost of NOK 340 billion. As a result, the travel time could potentially be cut by half and the total distance reduced by 50 km. [1, 2]

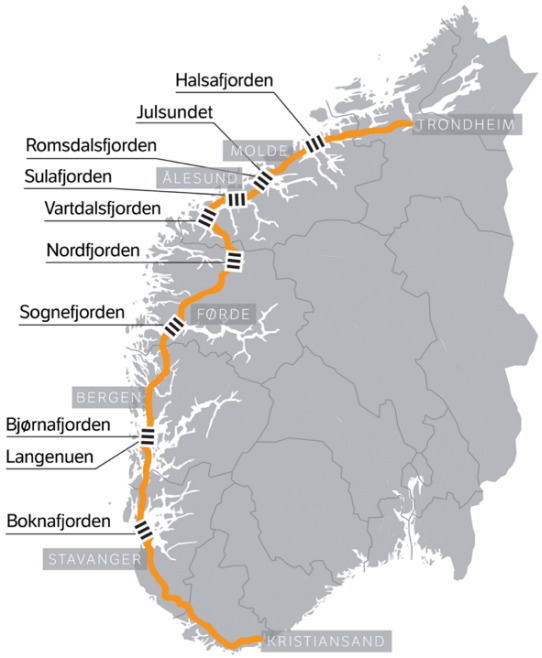

Figure 1.1: The coastal highway route E39 on the western coast of Norway from Kristiansand to Trondheim. [1]

Crossing the fjords without the use of ferries presents challenges due to their length and depth, requiring the development of new technologies to meet the demanding nature of the fjords and to obtain an adequate solution. One of these fjords, Bjørnafjorden, is located 30 km south of Bergen and has a length of approximately 5 km from Svarvhelleholmen in the south to Synnøytangen in the north, with a depth of 550 meters at its deepest. A proposed solution for the fjord crossing is an end-anchored, curved floating bridge with a straight, cable-stayed part at the southern end. Until now most floating bridges have been constructed in locations that are sheltered from extreme environmental conditions. Floating bridges are particularly sensitive to excitations from wind and waves, and the stability of the structure under strong winds is important. Therefore, the dynamic response of the structure under the environmental conditions of the fjord must be predicted.

1.2 Objectives

This thesis will build on the results in appendix G of the Metocean Specification report for the planned bridge in Bjørnafjorden, where raw measurement data from four wind masts in the area were used to identify strong wind events and to fit the spectral and coherence parameters. A threshold of 19 m/s for the mean wind speed was selected to identify the strong wind events over a five year period [3]; therefore in this thesis the effect of lowering the threshold of the mean wind speed will be investigated. In addition, the effect of atmospheric stability on the wind characteristics will also be investigated, as this was not included in the report.

The main work will be done through data processing of the wind measurements using the programming language Python. The Python scripts used in this thesis were originally made by associate professor Jungao Wang and modified for the objectives of this thesis accordingly.

The objectives of this thesis are:

1. Identify wind events by selecting a lower threshold limit for the mean wind speed.
2. Understand the frequency and spatial distribution of the wind measurements using different spectral models.
3. Compare the fitted spectral and coherence model parameters with the suggested parameters in the standard N400.
4. Study the effect of mean wind speed, mean wind direction and atmospheric stability on the wind characteristics at Bjørnafjorden.

Some limitations of the work done in this thesis are the location of the wind masts and the number of wind events. The wind masts are installed on hills and the local terrain would affect the wind measurements; the upstream terrain details for the masts are different from that of the bridge. In the design of bridges, the number of wind events from 5 years' worth of data are not fully representative for the design life at 100 years.

2. Theory

2.1 Wind

The following sections describing the basic wind theory is based on [4] and [5].

Wind is the three-dimensional flow of air; this motion being caused by a combination of factors relating to pressure differences in the atmosphere and the earth's rotation. The temperature of the earth's surface is not uniform, varying daily and seasonally, as well as geographically. The differences in solar radiation between the equator and the poles generates pressure and temperature differences, and wind motion is the result of trying to restore equilibrium between high- and low-pressure regions. As global winds move from the high-pressure regions to low-pressure regions, the rotation of the earth produces a force (known as the Coriolis force) that diverts the air masses eastwards in the northern hemisphere and westwards in the southern hemisphere.

When describing wind speed in statistical terms, the flow is divided into a mean wind component and a fluctuating, time-varying turbulence component. Figure 2.1 shows the instantaneous wind profile and the variation of both the mean wind speed $U(z)$ and turbulence $u(x, y, z, t)$ with height z. The total wind speed is the sum of the mean wind speed and the fluctuations.

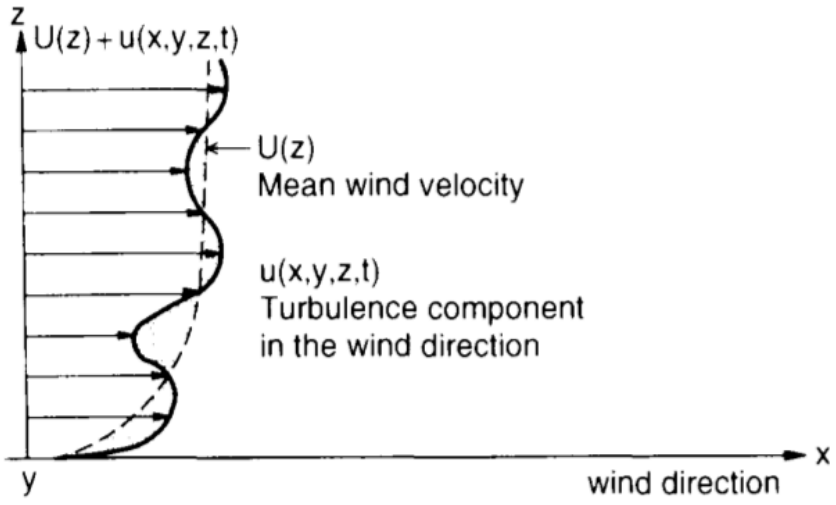

Figure 2.1: Mean wind speed and turbulence profile along the mean wind direction. [4]

Applying a cartesian coordinate system to the three-dimensional wind flow, the x-axis is the longitudinal component along the mean wind direction, the y-axis is the lateral component, and the z-axis is the vertical component, positive upwards. The wind flow can then be described as:

$$U(x, y, z) + u(x, y, z, t) \tag{2.1}$$

$$v(x, y, z, t) \tag{2.2}$$

$$z(x, y, z, t) \tag{2.3}$$

where $U(x, y, z)$ is the mean wind component, and $u(x, y, z, t)$, $v(x, y, z, t)$ and $w(x, y, z, t)$ are the turbulence components in the longitudinal, lateral, and vertical directions respectively.

2.1.1 Mean wind

As shown in Figure 2.1, the mean wind speed only varies with height above the ground at any given time. Close to the ground the wind is influenced by the uneven surface of the earth, but this influence decreases with increased height. As height above the ground increases, so does the mean wind speed, and the height at which the mean wind speed is unaffected by the terrain is called the gradient height.

To measure the roughness of the different types of terrain that the wind passes over, roughness length z_0 is introduced. The region in which the roughness length influences the wind speed profile is called the atmospheric boundary layer, and depending on the roughness length and intensity of the wind (e.g. a storm), the height of the atmospheric boundary layer can vary from 100 meters to 1 km.

Recommendations for values of z_0 varies in different codes and standards. Table 2.1, taken from Eurocode 1, gives a description of terrain types and their corresponding values for roughness length. The lower range of roughness length values corresponds to open sea or areas without significant buildings or vegetation. As the roughness length increases, it indicates more complex terrain conditions.

Table 2.1: Terrain categories and roughness lengths in Eurocode 1. [6]

Terrain category		z_0 m	z_{min} m
0	Sea or coastal area exposed to the open sea	0,003	1
I	Lakes or flat and horizontal area with negligible vegetation and without obstacles	0,01	1
II	Area with low vegetation such as grass and isolated obstacles (trees, buildings) with separations of at least 20 obstacle heights	0,05	2
III	Area with regular cover of vegetation or buildings or with isolated obstacles with separations of maximum 20 obstacle heights (such as villages, suburban terrain, permanent forest)	0,3	5
IV	Area in which at least 15 % of the surface is covered with buildings and their average height exceeds 15 m	1,0	10
The terrain categories are illustrated in Annex A.1.			

In the short term, e.g. 10 minutes or 1 hour, the mean wind speed is assumed to be an ergodic process, i.e. the mean wind speed at any given time can be reasonably represented by the mean wind speed for the entire time period. Two models are commonly used to represent the mean wind profile in the atmospheric boundary layer: the theoretical logarithmic profile and the empirical power law.

2.1.1.1 Logarithmic profile

Originally derived for the turbulent boundary layer on a flat plate by Prandtl, it was found that the logarithmic profile was also valid in the atmospheric boundary layer near the surface in strong wind conditions. The logarithmic profile is derived based on height above the ground z, surface shear stress τ_o and air density ρ_a. Since the height of the boundary layer is not taken into account, the profile is only valid near the ground (50-100 meters). This gives the common form:

$$U(z) = u_* \frac{1}{k} \ln\left(\frac{z}{z_0}\right) \tag{2.4}$$

where:

$u_* = \sqrt{\tau_o/\rho_a}$ is the friction velocity;

k is von Karman's constant, which is non-dimensional and experimentally found to have a value of about 0.4;
z_0 is the roughness length.

Thus, knowing the mean wind speed at a height z_1, the mean wind speed at different height z_2 can be found by the relation:

$$\frac{U(z_2)}{U(z_1)} = \frac{\ln(z_2/z_0)}{\ln(z_1/z_0)} \tag{2.5}$$

The logarithmic profile is well suited for fully developed wind flow over a homogenous terrain; however, this is rarely the case in reality. In addition to this, the logarithmic component of the profile makes it difficult to integrate and invalid for heights below $z = 0$.

2.1.1.2 Power law

Compared to the logarithmic profile, the power law has an empirical basis and is easier to integrate. The power law assumes that the wind speed varies exponentially with height, and relates the wind speed at any height to a reference wind speed:

$$U(z) = U(z_{ref})\left(\frac{z}{z_{ref}}\right)^{\alpha} \tag{2.6}$$

where:
$U(z_{ref})$ is the reference wind speed at a given height z_{ref}, usually 10 meters;
α is a non-dimensional exponent that changes with the roughness length, and is related to z_0 by:

$$\alpha = \frac{1}{\ln(z_{ref}/z_0)} \tag{2.7}$$

Figure 2.2 shows a comparison between the logarithmic profile and the power law. In general, there is good agreement between the two mean wind profiles.

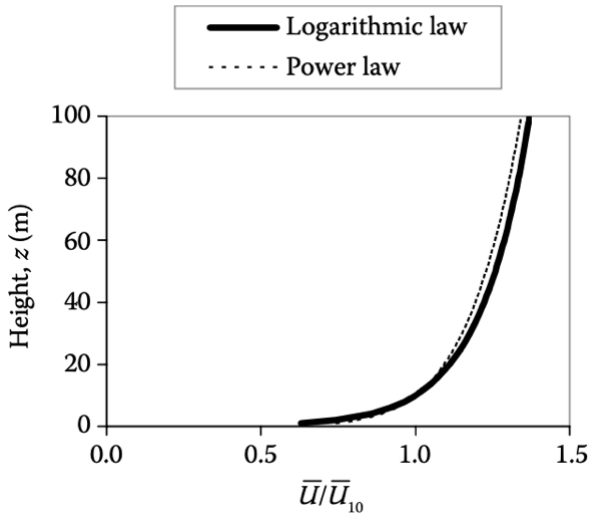

Figure 2.2: Comparison between the logarithmic profile and the power law (z_0 = 0.02, z_{ref} = 50 m and α = 0.128). [4]

2.1.1.3 Mean wind profile in Eurocode 1

In Eurocode 1, the variation of the mean wind speed with height is based on the logarithmic profile but differs in that the profile is only valid above a minimum height. It is given by the expression [6]:

$$v_m(z) = c_r(z) \cdot c_o(z) \cdot v_b \tag{2.8}$$

where:

$v_m(z)$ is the mean wind speed at height z;

$c_r(z)$ is the roughness factor;

$c_o(z)$ is the orography factor;

v_b is the basic wind velocity at 10 meters above ground in terrain category II with an annual probability of exceedance of 0.02, i.e. a return period of 50 years.

The roughness factor $c_r(z)$ is given by:

$$\begin{aligned} c_r(z) &= k_r \ln\left(\frac{z}{z_0}\right) \quad \text{for } z_{min} \leq z \leq z_{max} \\ c_r(z) &= c_r(z_{min}) \quad \text{for } z \leq z_{min} \end{aligned} \tag{2.9}$$

where $k_r = 0.19\left(\frac{z}{z_{0,II}}\right)^{0.07}$ is the terrain factor. $z_{0,II}$ is the roughness length corresponding to terrain category II from Table 2.1, and z_{min} is the minimum height. z_{max} is the maximum height and can be taken as 200 meters. [6]

2.1.2 Turbulence

In the case of wind flow, turbulence occurs because of the generation of vortices by the friction of the uneven surface terrain as the wind passes over, which is why wind flow in the atmospheric boundary layer has random periods with significant variations in length.

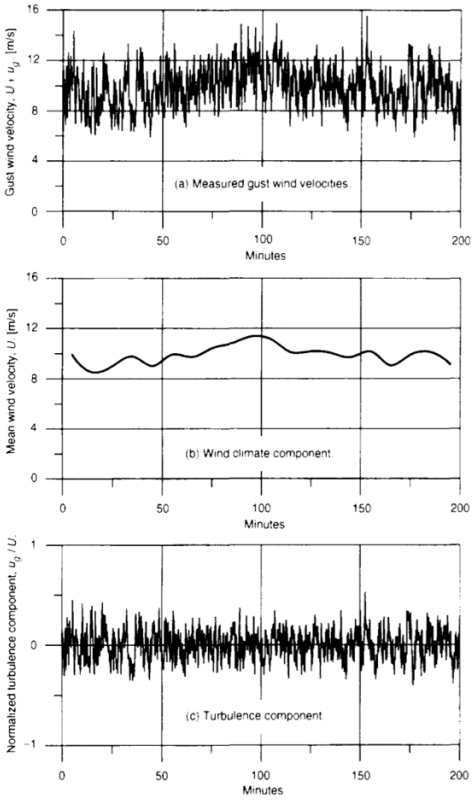

Figure 2.3: A time series of wind measurements showing the separation of the fluctuations from the mean wind. [4]

The turbulence components u, v and w are assumed to be stationary, stochastic functions of position and time with a zero mean value during a considered time period. In other words, the fluctuations are assumed to follow a Gaussian distribution, and their statistical properties do not change over time. Figure 2.3 shows a time series of the random variation of wind and its separated mean and turbulence component.

Assuming all turbulence components have zero mean values, the standard deviations are as follows:

$$\sigma_u = \sqrt{\frac{1}{T}\int_0^T (U(t) - \overline{U})dt} = \sqrt{\frac{1}{T}\int_0^T u(t)^2 dt} \tag{2.10}$$

$$\sigma_v = \sqrt{\frac{1}{T}\int_0^T v(t)^2 dt} \tag{2.11}$$

$$\sigma_w = \sqrt{\frac{1}{T}\int_0^T w(t)^2 dt} \tag{2.12}$$

where:

$\overline{U}$ is the average wind speed;

$u(t)$, $v(t)$ and $w(t)$ are the turbulence components in the longitudinal, lateral, and vertical directions respectively;

T is the length of the considered time period.

A common way to describe turbulence is through the turbulence intensity, defined as the ratio of the standard deviation of the turbulence component to the mean wind speed:

$$I_u(z) = \frac{\sigma_u(z)}{U(z)} \tag{2.13}$$

$$I_v(z) = \frac{\sigma_v(z)}{U(z)} \tag{2.14}$$

$$I_w(z) = \frac{\sigma_w(z)}{U(z)} \tag{2.15}$$

Turbulence intensity decreases with increasing height above the ground and increases with surface roughness. For a given turbulence intensity in the longitudinal wind direction, the corresponding lateral and vertical turbulence intensity components generally have a lower magnitude. N400 gives the following relation of the lateral and vertical turbulence intensities to the longitudinal turbulence intensity:

$$\begin{bmatrix} I_v \\ I_w \end{bmatrix} = \begin{bmatrix} 3/4 \\ 1/2 \end{bmatrix} I_u \tag{2.16}$$

where Eurocode 1 is referred to for the calculation of I_u, given as [6]:

$$I_u = \frac{1}{\ln(z/z_0)} \tag{2.17}$$

2.1.3 Friction velocity

As mentioned in section 2.1.1.1, friction velocity is defined using the surface shear stress τ_0 and the density of the fluid, in this case air, ρ_a. The friction velocity can be estimated by the following expression:

$$u_* = [(\overline{uw})^2 + (\overline{vw})^2]^{1/4} \tag{2.18}$$

where $\overline{uv}$ and $\overline{vw}$ are the covariances of the turbulence components, also known as components of the Reynolds stress tensor. [7] Equation 2.18 is an approximation when the $\overline{uv}$ term, or the horizontal shear stress, is negligible, however this is not always the case for complex terrain such as in a fjord. [8] The friction velocity is a measure of the surface shear stress and an important scaling variable when the wind characteristics are dependent on the atmospheric stability. [9] The relationship between the standard deviation of the longitudinal turbulence component to the friction velocity is given by:

$$\sigma_u = \beta u_* \tag{2.19}$$

where β is a non-dimensional parameter referred to as the turbulence ratio. [10] Based on a study of three different fjords in western Norway, β was found to range from approximately 1.74 to 4.12 for wind speeds above 12 m/s, showing large variation in complex terrain such as a fjord. [8]

2.1.4 Integral length scales

The spatial characteristics of wind turbulence can be described by integral length scales, which are a measure of the sizes of the vortices in wind. Length scales are commonly notated as L_i^s,

where $i = u, v, w$ is the turbulence component and $s = x, y, z$ is the direction. For example, L_u^x is the integral length scale for the turbulence component u along the longitudinal direction x.

Each integral length scale is defined as:

$$L_i^s = \int_0^\infty R_i(s)ds \tag{2.20}$$

where $R_i(s)$ is the autocorrelation function for the turbulence component at two points separated by distance s. The autocorrelation function and its properties will be presented in section 2.5.3.

Assuming Taylor's hypothesis of "frozen turbulence", which states that temporal turbulence variations can be based on spatial wind speed field characteristics [4], the integral length scales in the longitudinal direction can be found by:

$$L_i^x = \overline{U} \int_0^\infty R_i(\tau)d\tau = \overline{U}T_i; i = u, v, w \tag{2.21}$$

where:
$\overline{U}$ is the mean wind speed;
$R_i(\tau)$ is the autocorrelation function for the i^{th} turbulence component at two points in time separated by τ;
T_i is the integral time scale for the i^{th} turbulence component, which may be thought of as a characteristic time of memory.

An example of the autocorrelation function for the longitudinal turbulence component is shown in Figure 2.4. When estimating the integral length scale, the autocorrelation function is integrated only up to the first zero crossing. [11]

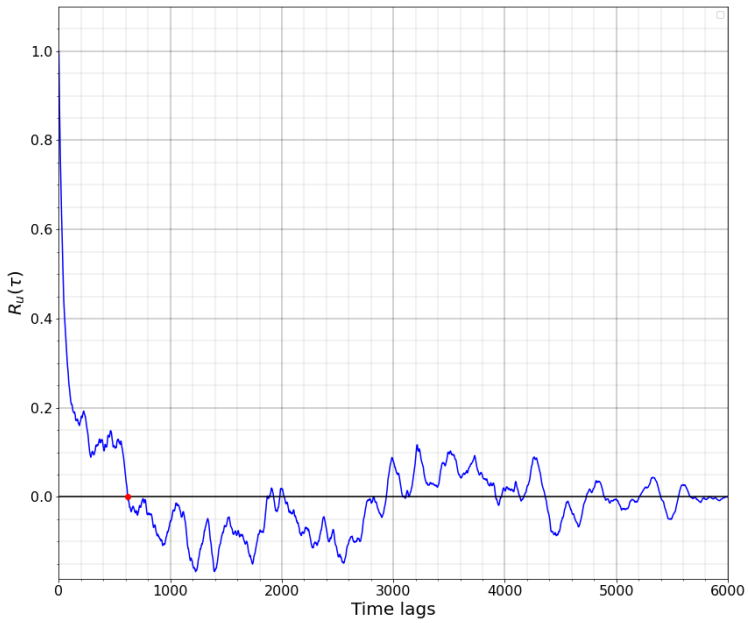

Figure 2.4: The autocorrelation function for a time series of the longitudinal turbulence component. The red marker indicates the first zero crossing, which in this case occurs at a time lag of 617.

Due to its dependence on factors such as height above the ground z and surface roughness z_0, estimation of the integral length scales based on measured wind data can show large scatter. [12] In N400, L_u^x can be estimated by:

$$L_u^x(z) = \begin{cases} L_1(z/z_1)^{0.3}, & z > z_{min} \\ L_1(z_{min}/z_1)^{0.3}, & z \leq z_{min} \end{cases} \tag{2.22}$$

where:

$L_1 = 100$ is a reference length scale;

$z_1 = 10$ is reference height;

z_{min} is a minimum height defined in Table 2.1 from Eurocode 1. [6]

For homogenous flow conditions, the remaining integral length scales are expressed as a fraction of $L_u^x(z)$:

$$\begin{bmatrix} L_u^y \\ L_u^z \\ L_v^x \\ L_v^y \\ L_v^z \\ L_w^x \\ L_w^y \\ L_w^z \end{bmatrix} = \begin{bmatrix} 1/3 \\ 1/5 \\ 1/4 \\ 1/4 \\ 1/12 \\ 1/12 \\ 1/18 \\ 1/18 \end{bmatrix} L_u^x \tag{2.23}$$

2.2 Atmospheric stability

Atmospheric stability refers to the vertical motion of an elementary air mass relative to the surrounding temperature, and its tendency to resist this motion. Depending on this vertical motion, the stability of the atmosphere is classified as stable, neutral, or unstable. Two forces act on the elementary air mass: gravity and a buoyancy force. In stable conditions, the temperature of the air mass is lower than its surroundings, causing it to return to its original position if displaced. Conversely, in unstable conditions the air mass continues to rise when displaced. If the air mass remains in its new position after being displaced, the stability is classed as neutral. [13]

The atmospheric stability depends on the lapse rate, which is defined as the rate of change in temperature with height. Assuming a system in which there is no water vapor in the air and no heat transfer occurs, the dry adiabatic lapse rate is given as $-\left(\frac{dT}{dz}\right)_{adiabtic} = \frac{1^{\circ}C}{100\ m}$. The negative sign is introduced because the temperature of the atmosphere decreases with increasing height. In determining the atmospheric stability, the actual lapse rate of the system (which is affected by heat transfer and the amount of water vapor in the air) is compared with the dry adiabatic lapse rate. If the actual lapse rate is smaller than the dry adiabatic lapse rate, the atmosphere is stable. Unstable conditions occur when the opposite is true, and an actual lapse rate equal to the dry adiabatic lapse rate corresponds to neutral conditions. [13]

The Monin-Obukhov length is L is commonly used to classify atmospheric stability through the following expression [14]:

$$\zeta = \frac{-gkz\overline{w\theta_v}}{\overline{\theta_v}u_*^3} \tag{2.24}$$

where:

$\zeta = \frac{z}{L}$ is the stability parameter;

g is the gravitational constant;

k is the von Karman constant;

$\overline{\theta_v}$ is the virtual potential temperature;

$\overline{w\theta_v}$ is the surface flux.

Negative and positive values of the stability parameter ζ correspond to unstable and stable conditions respectively.

2.3 Wind spectra

The power spectral density shows how the turbulence is distributed over different frequencies, i.e. the energy in wind as a function of frequency. Knowing the autocorrelation function for the turbulence components, the Fourier transform and its inverse are:

$$S_x(\omega) = \frac{1}{2\pi}\int_{-\infty}^{\infty} R_x(\tau)e^{-i\omega t}\,d\tau \tag{2.25}$$

$$R_x(\tau) = \int_{-\infty}^{\infty} S_x(\omega)e^{i\omega t}\,d\omega \tag{2.26}$$

where $S_x(\omega)$ is the spectral density of the random process $x(t)$ and ω is the angular frequency. Fourier analysis will be presented in detail in section 2.5.

The variance of the turbulence components is given as the area under graph of the spectral density:

$$\sigma_i^2 = \int_0^{\infty} S_i(f)\,df; i = u, v, w \tag{2.27}$$

where:

σ_i^2 is the variance for the i^{th} turbulence component;

f is the frequency in Hz;

$S_i(f)$ is the spectral density for the i^{th} turbulence component.

A general form of the normalized power spectral density for wind turbulence can be given as [9]:

$$\frac{fS_i(f)}{u_*^2} = \frac{AR^2\hat{f}^\gamma}{\left(C + B\hat{f}^\alpha\right)^\beta}; i = u, v, w \tag{2.28}$$

where $\hat{f} = \frac{f\Lambda}{U}$ is the non-dimensional frequency, and Λ can be either the height above the ground z or the integral length scale in the longitudinal direction L_i^x.

In Equation 2.25, R is the turbulence ratio as described in section 2.1.3, while A, C, B, γ, α and β are six parameters that control the shape of the spectrum. Figure 2.5 gives an overview of each parameter's influence. Parameter A controls the total energy distribution of the wind flow, parameters B and C control the energy distribution in the high and low frequency range respectively, and the last three parameters adjust the slope. [9] Some of the most common forms of spectral models used in wind engineering, which use $C = \gamma = 1$, are presented in the following sections.

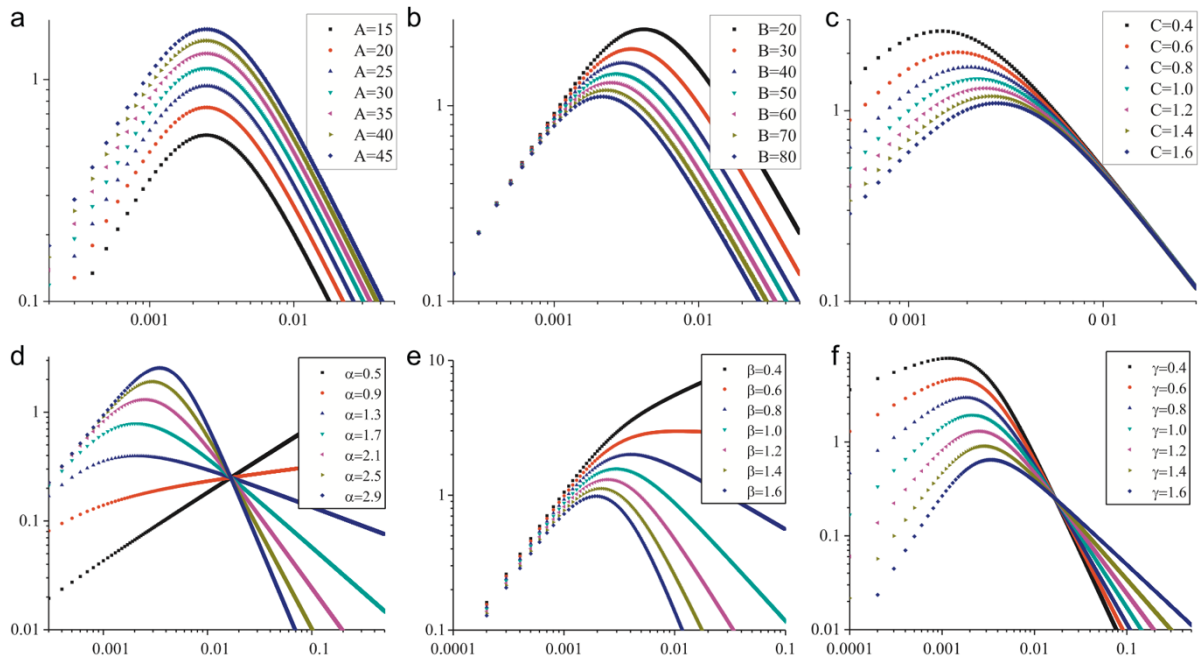

Figure 2.5: Influence of each parameter on the shape of the wind spectrum when all other parameters are fixed at their median value. The horizontal axis represents the frequency, and the vertical axis represents the spectrum normalized by the standard deviation. [9]

2.3.1 Kaimal spectrum

The Kaimal spectrum is a commonly used form of the wind spectrum, developed for wind flow over a flat, homogenous terrain in neutral conditions. The normalized spectra for the wind turbulence components are given in [15] as:

$$\frac{fS_u(f)}{u_*^2} = \frac{105\hat{f}}{\left(1+33\hat{f}\right)^{5/3}} \tag{2.29}$$

$$\frac{fS_v(f)}{u_*^2} = \frac{17\hat{f}}{\left(1+9.5\hat{f}\right)^{5/3}} \tag{2.30}$$

$$\frac{fS_w(f)}{u_*^2} = \frac{2\hat{f}}{1+5.3\hat{f}^{5/3}} \tag{2.31}$$

where $\hat{f} = \frac{fz}{\bar{U}}$ is the non-dimensional frequency. The longitudinal and lateral spectra are so-called "blunt models" with $\alpha = 1$ and $\beta = \frac{5}{3}$, while the vertical spectrum is a "pointed model" with $\alpha = \frac{5}{3}$ and $\beta = 1$. The latter type of spectral model has a sharper peak compared to the former, as can be seen from Figure 2.5. [14]

2.3.2 N400

The N400 wind spectrum is based on the original Kaimal spectrum but uses the integral length scale as a scaling variable instead of z. It is normalized with the standard deviation rather than friction velocity, and parameter B is related to parameter A by a factor of 1.5 [16]:

$$\frac{fS_i(f)}{\sigma_i^2} = \frac{A_i\hat{f}_i}{\left(1+1.5A_i\hat{f}_i\right)^{5/3}}; i = u, v, w \tag{2.32}$$

where $\hat{f}_i = \frac{fL_i^x(z)}{\bar{U}(z)}$ is the reduced frequency and $L_i^x(z)$ is the integral length scale for the i^{th} wind component in the longitudinal direction.

Since there is only one parameter, A_i, it controls the location of the spectral peak on the horizontal axis of the normalized wind spectra. A higher value for A_i leads to more energy in at lower frequencies. Suggested values in N400 are $A_u = 6.8$, $A_v = 9.4$ and $A_w = 9.4$. [16]

2.3.3 Harris spectrum

The Harris wind spectrum is based on the von Karman spectrum, the latter having been developed for laboratory turbulence. Harris adapted the longitudinal von Karman spectrum for use in wind engineering [5]:

$$\frac{fS_u(f)}{\sigma_u^2} = \frac{4\left(\frac{L_u^x f}{\overline{U}}\right)}{\left[1 + 70.8\left(\frac{L_u^x f}{\overline{U}}\right)^2\right]^{5/6}} \tag{2.33}$$

The maximum of Equation 2.33 occurs at $f = 0.1456\frac{z}{L_u^x}$, meaning the energy content of the spectrum on the frequency axis is controlled by the integral length scale. [12]

2.3.4 Busch-Panofsky spectrum

The Busch-Panofsky spectrum is a common form used for the vertical wind spectrum, given by the expression [5]:

$$\frac{fS_w(f)}{\sigma_w^2} = \frac{2.15\left(\frac{fz}{\overline{U}}\right)}{1 + 11.16\left(\frac{fz}{\overline{U}}\right)^{5/3}} \tag{2.34}$$

Equation 2.34 is similar to the original Kaimal spectrum with the "pointed" model type, only with different suggested values for parameters A and B.

2.3.5 Frøya spectrum

According to [10], few models of the wind spectrum have been designed for offshore conditions. The Frøya spectrum is an empirical model based on offshore wind measurements at an island off the Norwegian coast in the North Sea. Developed by Andersen and Løvseth in 1992, the aim of the model, which is a generalization of the Harris and Kaimal spectra, was to better capture the energy content in the low frequency range of the wind spectrum [17], as this is of significance for offshore structures with lower eigen-frequencies compared to onshore structures.

The Frøya spectrum only considers the longitudinal turbulence component and is not normalized with respect to the standard deviation or the friction velocity, thereby directly representing the actual energy content at varying frequencies. The spectrum is given as [18]:

$$S_u(f) = 320 \frac{\left(\frac{U_0}{10}\right)^2 \left(\frac{z}{10}\right)^{0.45}}{\left(1 + \tilde{f}_n\right)^{\frac{5}{3n}}} \tag{2.35}$$

where:

U_0 is the 1-hour mean wind speed at 10 meters above sea level;

z is the height above sea level;

$n = 0.468$;

$\tilde{f}_n = 172f \left(\frac{z}{10}\right)^{2/3} \left(\frac{U_0}{10}\right)^{-0.75}$

2.4 Coherence

While the power spectral density gives information on the frequency distribution of the energy in wind at a single point, the spatial correlation of wind flow at different frequencies is an important factor to consider, particularly when it comes to slender structures such as long-span bridges.

S_{uu} is the cross-spectral density, which describes the frequency distribution of the spatial relation between turbulence components at two points. The cross-spectral density is a complex variable, and its real part is called the co-spectral density. Normalizing the co-spectral density gives the coherence [4]:

$$Coh_{uu} = \frac{Re[S_{uu}(P_1, P_2, f)]}{\sqrt{S_u(P_1, f) S_u(P_2, f)}} \tag{2.36}$$

where:

$S_{uu}(P_1, P_2, f)$ is the cross-spectral density at points P_1 and P_2;

$S_u(P_1, f)$ and $S_u(P_2, f)$ are the spectral densities at points P_1 and P_2 respectively.

The coherence depends on the distance Δr between the two considered points; when $\Delta r = 0$ there is full coherence, and when $\Delta r \to \infty$, i.e. the distance between the points is very large, the coherence is zero.

Davenport's (1962) empirical model for the longitudinal turbulence is commonly used to model the coherence [4]:

$$Coh_u = \exp\left[-\left(\frac{C \cdot f \cdot \Delta r}{\overline{U}}\right)\right] \tag{2.37}$$

where:
C is a non-dimensional, empirical constant that determines the degree of spatial correlation;
Δr is the separation length between two points in the horizontal plane;
$\overline{U} = \frac{1}{2}\big(U(z_1) + U(z_2)\big)$ is the mean value of the wind speed at the two points.

Some disadvantages of this empirical model are that it does not allow for negative values for the coherence, and that it implies full correlation at very low frequencies regardless of how large the separation length is. Typical values for the parameter C range between from 10 to 20. [5] A small value for C in Equation 2.44 indicates more coherence, and conversely, a larger value for C indicates less coherence.

In N400, the normalized co-spectrum is given in the same form as Davenport's empirical model:

$$\frac{Re\big[S_{i_1}S_{i_2}(f, \Delta s_j)\big]}{\sqrt{S_{i_1}(f) \cdot S_{i_2}(f)}} = \exp\left(-C_{ij}\frac{f\Delta s_j}{v_m(z)}\right) \tag{2.38}$$

where:
i_1, i_2 are the turbulence components u, v or w;
j is the longitudinal direction x or the lateral direction y;
C_{ij} is the coherence parameter;
Δs_j is the horizontal or vertical distance between the considered points.

Suggested values for the coherence parameters in N400 are $C_{uy} = C_{uz} = 10$, $C_{vy} = C_{vz} = C_{wy} = 6.5$ and $C_{wz} = 3$. Values for the longitudinal direction are not specified in the standard but can be assumed to be $C_{ux} = C_{wx} = 3$ and $C_{vx} = 6$. [3]

The wind coherence for the horizontal and vertical separation can be described by Equation 2.39 and 2.40 respectively:

$$Coh_{i,hor} = e^{\left(\frac{f}{0.5(\bar{U}(z_1)+\bar{U}(z_2))}\sqrt{\left((C_{ix}dx)^2+(C_{iy}dy)^2\right)}\right)}\cos\left(\frac{2\pi f}{0.5(\bar{U}(z_1)+\bar{U}(z_2))}dx\right); \quad i = u, v, w \tag{2.39}$$

$$Coh_{i,ver} = e^{-\left(\frac{f}{0.5(\bar{U}(z_1)+\bar{U}(z_2))}C_{iz}dz\right)};\ i = u, v, w \tag{2.40}$$

where the separation lengths in each direction is represented by dx, dy and dz. The cosine term in Equation 2.39 is introduced to account for the time-lag in the longitudinal direction. [3]

2.5 Signal processing

As wind flow varies in a random, unpredictable way due to the effect of turbulence, it is necessary to describe it in statistical terms. [4] The following sections are based mainly on [19] and present statistical concepts that are important to adequately describe wind flow.

2.5.1 Fourier analysis

Knowing the frequency composition of an aperiodic function, such as turbulent wind, is important in determining loads and fatigue on a structure. This can be done through Fourier analysis.

Fourier analysis is the process through which a complex function or a signal can be decomposed into a sum of trigonometric functions. Assuming $x(t)$ is a random periodic process as a function of time t, with a period of T and a mean value of zero, the process can be represented through a Fourier integral as:

$$x(t) = 2\int_0^\infty A(\omega)\cos\omega t\, d\omega + 2\int_0^\infty B(\omega)\sin\omega t\, d\omega \tag{2.41}$$

$$A(\omega) = \frac{1}{2\pi}\int_{-\infty}^{\infty} x(t)\cos\omega t\, dt \tag{2.42}$$

$$B(\omega) = \frac{1}{2\pi}\int_{-\infty}^{\infty} x(t)\sin\omega t\, dt \tag{2.43}$$

where $A(\omega)$ and $B(\omega)$ are the components of the Fourier transform of $x(t)$ and $\omega = \frac{2\pi}{T}$ is the angular frequency. It is common to represent the Fourier transform in complex form, using Euler's formula $e^{i\theta} = \cos\theta + i\sin\theta$:

$$X(\omega) = \frac{1}{2\pi}\int_{-\infty}^{\infty} x(t)e^{-i\omega t}\, dt \tag{2.44}$$

$$x(t) = \int_{-\infty}^{\infty} X(\omega)e^{i\omega t}\, d\omega \tag{2.45}$$

Equations 2.44 and 2.45 are referred to as a Fourier transform pair, where $X(\omega)$ is the complex Fourier transform of $x(t)$. A condition for this theory to apply is that the function decays to zero when $|t| \to 0$:

$$\int_{-\infty}^{\infty} |x(t)|\, dt < \infty \tag{2.46}$$

However, in reality a random process such as wind turbulence is not periodic and does not satisfy the above condition. This can be overcome by analyzing the autocorrelation function instead, which will be discussed in section 2.5.3.

For non-continuous time series, such as wind measurements, a discrete Fourier transform (DFT) is given as:

$$X_k = \frac{1}{N}\sum_{r=0}^{N-1} x_r e^{-i(2\pi kr/N)}\,; k = 0, 1, 2, \dots, (N-1) \tag{2.47}$$

where x_r is the discrete series and $r = 0, 1, 2, \dots, (N-1)$. For a time series with r number of samples, using DFT will require r^2 number of operations, which can be computationally demanding for large time series. The Fast Fourier transform (FFT) algorithm commonly used

reduces the complexity from r^2 to $r \log_2 r$, thereby significantly reducing computer processing time, and even improving accuracy due to less round-off errors.

2.5.2 Welch's method

Although FFT analysis of a time series can yield accurate power spectral densities, the resulting spectrum may be noisy and somewhat difficult to interpret. Welch's method is an approach to estimate the spectral density through the following steps, described in [20]:

1. The time series is divided into segments of length L.
2. Each segment is windowed, meaning the signal is tapered at each end of the segment, thus reducing the noisiness of the resulting spectrum.
3. The segments may also overlap each other (up to 50%) to reduce loss of information from windowing.
4. The Fourier transform of each segment is calculated and the value is squared, resulting in a modified periodogram.
5. The final estimation of the power spectral density is calculated from the average of all the modified periodograms.

2.5.3 Correlation and autocorrelation

Correlation is a measure of how two variables are related to each other by degree of linear dependency. Considering two variables x and y, the correlation coefficient is defined as:

$$\rho_{xy} = \frac{E\left[(x - m_x)(y - m_y)\right]}{\sigma_x \sigma_y} \tag{2.48}$$

where m_x, m_y and σ_x, σ_y are the mean values and standard deviations of x and y respectively. If $\rho_{xy} = \pm 1$, there is perfect correlation between the two variables.

The autocorrelation function gives information about the degree of correlation between two values of a signal as a function of separation, which can be either spatial or temporal. For a random process $x(t)$, the autocorrelation function $R_x(t, \tau)$ is defined as the expected value of the product $x(t)x(t + \tau)$, see Figure 2.6. When assuming that $x(t)$ is stationary process, as is the case for wind turbulence, the expected value will only depend on the time separation τ:

$$R_x(\tau) = E[x(t)x(t+\tau)] \tag{2.49}$$

Similarly, the mean and standard deviation will also be independent of t:

$$E[x(t)] = E[x(t+\tau)] = m \tag{2.50}$$

$$\sigma_{x(t)} = \sigma_{x(t+\tau)} = \sigma \tag{2.51}$$

The correlation coefficient for $x(t)$ and $x(t+\tau)$ is then:

$$\rho = \frac{E[\{x(t)-m\}\{x(t+\tau)-m\}]}{\sigma^2} = \frac{R_x(\tau)-m^2}{\sigma^2} \in [-1,1] \tag{2.52}$$

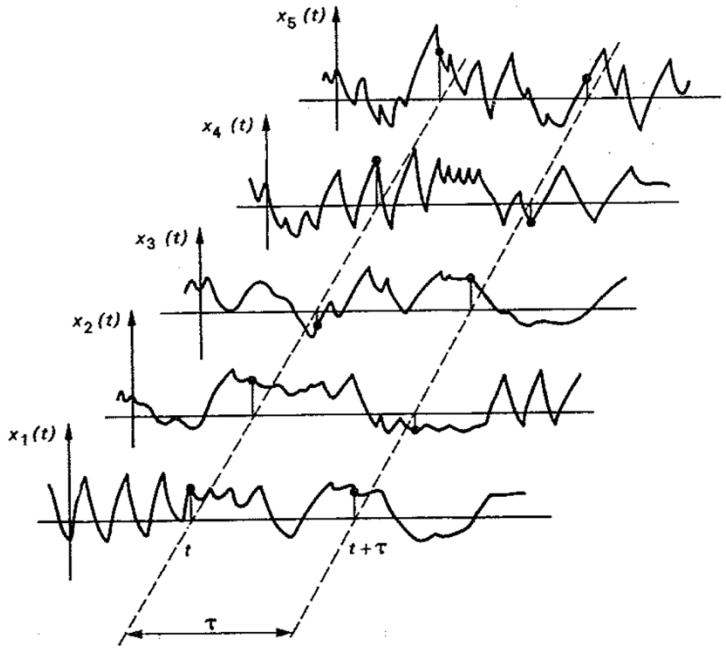

Figure 2.6: An example of different realizations of the same stochastic process. [19]

The following are the properties of the autocorrelation function:

1. $R_x(\tau)$ is symmetric; $R_x(\tau) = R_x(-\tau)$
2. $R_x(\tau)$ is a decaying function; $R_x(\tau) \leq R_x(0)$
3. $-\sigma^2 + m^2 \leq R_x(\tau) \leq \sigma^2 + m^2$
4. When $\tau = 0$, $R_x(\tau = 0) = E[x(t)^2] = E[x^2]$
5. When $\tau \to \infty$, $\rho \to 0$, i.e. $R_x(\tau \to \infty) \to m^2$

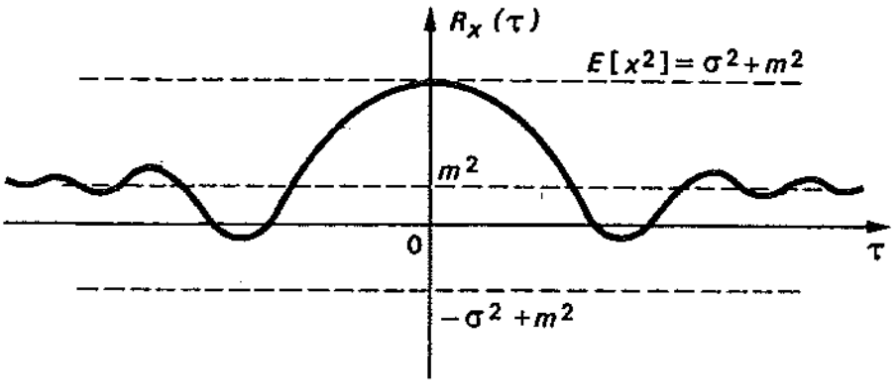

Figure 2.7: Properties of the autocorrelation function. [19]

In the case of the turbulence components of wind flow, as stated earlier, each component is regarded as a stationary, stochastic process with a zero mean value. Therefore, the autocorrelation function of the turbulence component decays to zero as $\tau \to \infty$:

$$\int_{-\infty}^{\infty} |R_i(\tau)|\, dt < \infty;\ i = u, v, w \tag{2.53}$$

Thus, the Fourier transform of the turbulence components can be calculated using the autocorrelation function, as it satisfies the condition for the theory to apply.

2.5.4 Transformation matrix

When using sonic anemometers to measure wind speed, it is given as three axis velocities in a UVW coordinate system, where V is the axis 90˚ anti-clockwise from U in the horizontal plane and W is the vertical axis perpendicular to U and V. [21] If assuming that the U-axis is pointing towards the east and the V-axis north, this coordinate system will be referred to as the global coordinate system $\boldsymbol{G}$, with the reference frame given as:

$$\boldsymbol{G} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.54}$$

The local coordinate system of the wind is as described in section 2.1, in which the mean wind direction is aligned with mean wind speed. Measured from the global U-axis, the longitudinal

wind direction is θ, and if the z-axis has the same direction as the W-axis, the reference frame for the local coordinate system $\boldsymbol{L}$ in relation to $\boldsymbol{G}$ is given as:

$$\boldsymbol{L} = \begin{bmatrix} \sin\theta & \cos\theta & 0 \\ \sin\left(\theta - \frac{\pi}{2}\right) & \cos\left(\theta - \frac{\pi}{2}\right) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.55}$$

In other words, the x-axis of the wind flow's local coordinate system is oriented at an angle θ anti-clockwise from the global U-axis. To transform the three axis velocities measured by the sonic anemometer from the global to the local coordinate system, a transformation matrix is used, given by:

$$\boldsymbol{T} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.56}$$

The three axis velocities in the global coordinate system $\boldsymbol{G}$ are represented as $\boldsymbol{V_G} = [U_G \quad V_G \quad W_G]^T$, and the three axis velocities in the wind flow's local coordinate system $\boldsymbol{L}$ are represented by $\boldsymbol{V_L} = [U_L \quad V_L \quad W_L]^T$. By using Equation 2.56, we can find $\boldsymbol{V_L}$ by:

$$\boldsymbol{V_L} = \boldsymbol{T}\boldsymbol{V_G} \tag{2.57}$$

3. Wind measurements at Bjørnafjorden

3.1 Topography

Bjørnafjorden is located 30 km south of Bergen and is approximately 25 km wide and 5 km long, running into the mainland in the east and the North Sea in the west. Figure 3.1 shows the topography surrounding the fjord, the location of wind masts and the potential location of the planned floating bridge. A group of islands separate the fjord from the open sea, ranging in height from 0-250 meters. The mainland on the eastern side is characterized by mountainous regions with heights up to 800 meters.

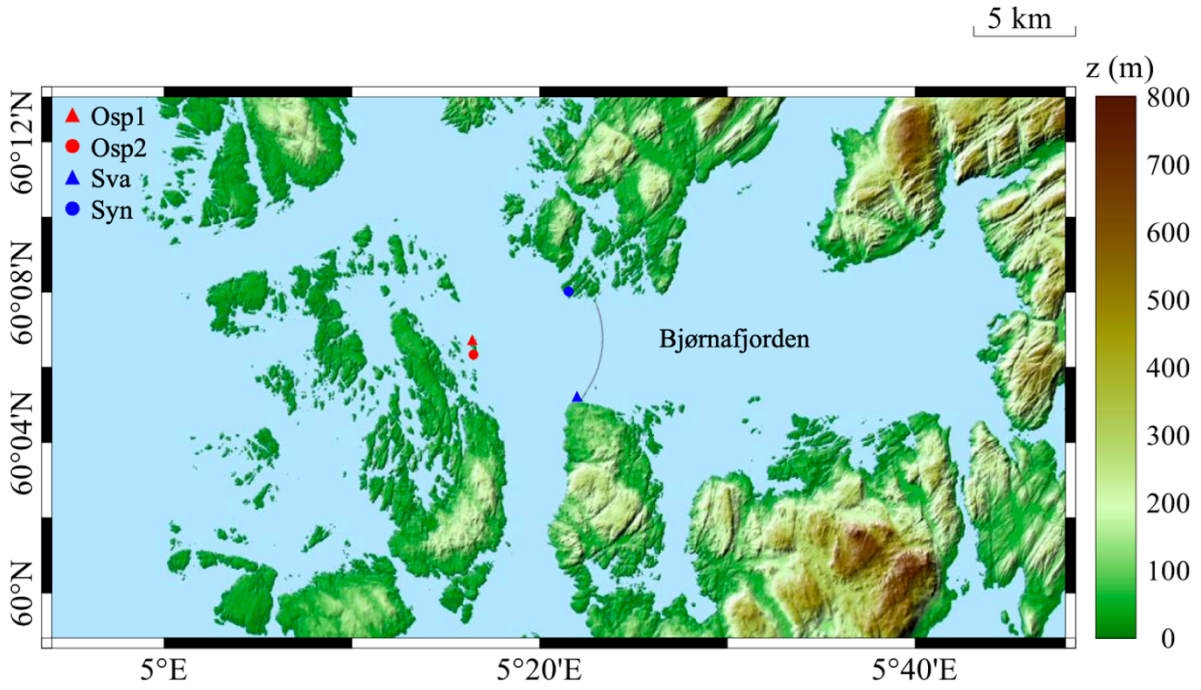

Figure 3.1: Overview of wind mast locations and topography in Bjørnafjorden. [3]

To better understand the wind characteristics at the fjord, four 50-meter-high wind masts were installed in 2015; one near the northern end of the planned bridge at Synnøytangen, one near the southern end at Svarvhelleholmen and two at the island Ospøya to the west of the bridge. [22] The terrain surrounding the wind masts is relatively flat for winds coming from west, around 0-70 meters, while for southern and northern winds there is some significant variation in elevation. For the masts at Ospøya and Svarvhelleholmen there is a long fetch over water for wind coming from the east. Similar open-fetch conditions are present for Svarvhelleholmen for north-western and north-eastern winds. For southern and northern winds at

Svarvhelleholmen and Synnøytangen respectively, the terrain is steep and complex, varying from 100-300 meters.

3.2 Wind masts

Each mast is outfitted with three Gill WindMaster Pro 3-Axis Anemometers [21], with a sampling frequency of 10 Hz. The sonic anemometers determine the wind speed by measuring the time delay of a pulse between an upper and a lower transducer. Each anemometer has three pairs of transducers, and the times are compared between each pair. The sensors are installed such that the U and V-axis are parallel to the global east and north respectively for ease of calculations. [22] In addition to wind speed, mean wind direction and sonic temperature is measured as well. Table 3.1 gives an overview at the heights the sensors are installed.

Table 3.1: Information for sonic anemometers. [3]

Location	Ground height	Sensor height above ground			Installation date
		A	B	C	
Ospøya 1	23 m	48.8 m	48.8 m	33.1 m	2015.12.03
Ospøya 2	34 m	48.8 m	48.8 m	33.1 m	2015.12.17
Svarvhelleholmen	5 m	48.3 m	31.9 m	12.7 m	2015.03.18
Synnøytangen	26 m	48.3 m	31.9 m	12.7 m	2015.02.23

4. Methodology

4.1 Identification of wind events

In the Metocean Specification report, the raw data from the wind masts in Bjørnafjorden was used to identify strong wind events from 2015 to 2020 based on a threshold for the mean wind speed. The selected threshold was 19 m/s and resulted in 52 strong wind events over a five-year period. [3] The new threshold is selected to be 6 m/s in order to increase the number of events, and are hereby referred to simply as wind events rather than strong wind events. This is done to study the evolution of the wind characteristics with different wind speed thresholds.

The identification of wind events was done through the following steps:

- The mean wind speed and mean wind direction for each sensor was loaded from files containing the raw measurement data.
- The function 'signal.find_peaks' was used to find the time at which the top sensor recorded a mean wind speed larger than or equal to 6 m/s. A minimum distance between each peak of 24 hours was specified to prevent overlap. The same was done at a reference station, Slåtterøy Fyr, located 30 kilometers south-west of Bjørnafjorden.
- Time tags of the wind event (date and time), mean wind speed and mean wind direction for each sensor were exported into an Excel-file for reference.

Two instances where the wind events overlapped were discovered on 2016.12.26 and 2017.04.24; this was corrected so that duplicates would not affect the results. After the correction, a total of 305 wind events were identified from 2015 to 2020 using the new threshold of 6 m/s.

4.2 Data processing and visualization

With the time tag of each wind event identified, the raw measurement data from the sensors was extracted using a time window of ±24 hours around the time tag, i.e. each event was assumed to last 2 days. Sixteen of the wind events were missing some data in the selected 48-hour window, therefore these events were not considered further in the study, reducing the total number of wind events from 305 to 289.

The raw data from each 48-hour wind event was then further processed into 1-hour events for each sensor, containing one mean wind speed and one mean wind direction. The components of the mean wind vector were transformed from the global coordinate system to the wind flow's local coordinate system using Equation 2.57. The sonic temperature from sensor A at each wind mast was also extracted for use in the stability calculations. A 1-hour time averaging was selected because the atmospheric stability in offshore environments is dominated by unstable conditions and thus the turbulence time scales are larger when compared to neutral conditions. [10] Additionally, the accuracy of the estimated turbulence statistics are increased with a longer time averaging. [10] Time averaging over 1 hour is associated with non-stationarity [8], however getting stationary time series is still a challenge when using a 1-hour time averaging.

Similarly, as for the 1-hour events, 10-minute events were also processed for estimation of the length scale. This was done for better comparison between the reference length scales in N400, which are based on the standard 10-minute time-averaging.

In addition to the data processing, each wind event was visualized in a time history plot to see its evolution over a 48-hour period. Figure 4.1 shows a time history from a wind event on October 22, 2018, with the evolution of wind speed and direction, and a comparison of the mean wind speeds recorded by each of the four wind masts. This event was selected to be further studied in detail for the spectral and coherence parameter fittings in Chapter 5.

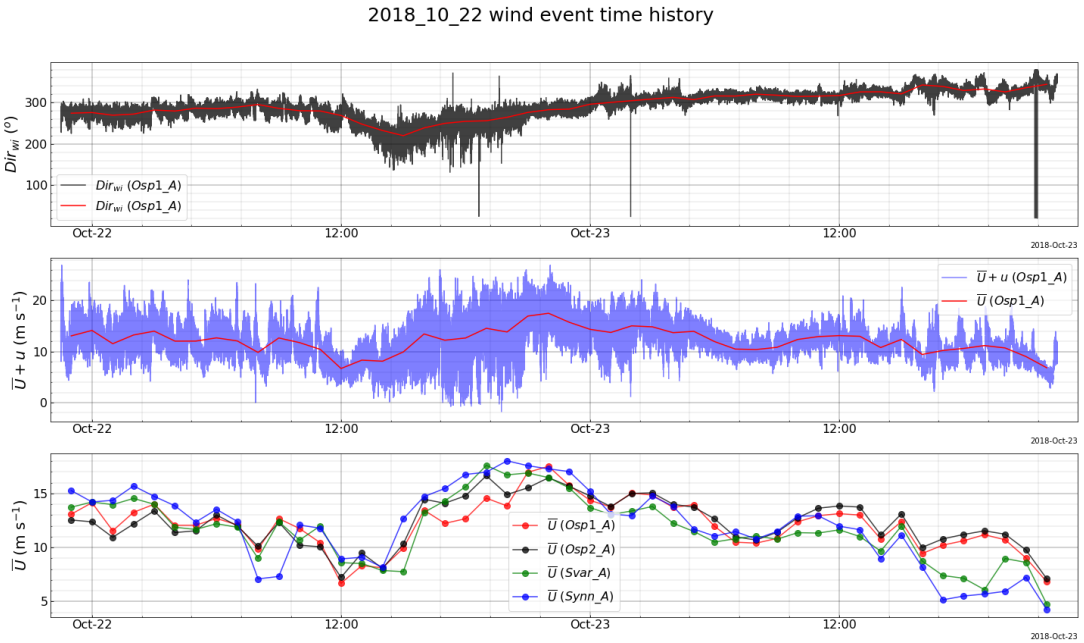

Figure 4.1: Evolution of a wind event over 48 hours on October 22, 2018.

4.3 Spectrum fitting

The spectrum fitting was done largely following the method described in the Metocean Specification report. [3] The main points will be summarized in this section.

Four criteria were specified and needed to be fulfilled to perform the fitting:

- The available data for a 1-hour event should be more than 95%, and any missing data was estimated using linear interpolation.
- The range of the moving average mean wind speed (using 10-minute averaging) should be no more than 30% of the mean wind speed to ensure stationarity of the 1-hour event.
- The range of the moving average mean wind direction (using 10-minute averaging) should be no more than 20 degrees to ensure stationarity of the 1-hour event.
- The mean wind speed recorded by sensors A, B and C should be higher than or equal to 6 m/s.

The function 'scipy.signal.welch' was used to numerically estimate the power spectral density using Welch's method from the measured wind turbulence at each wind mast. [23] Sensitivity studies were done in [3] relating to the choice of different parameters in using Welch's method; results showed that the fitted spectral parameters were sensitive to the range of frequency selected to perform the fitting, more specifically the low frequency range. Since, the eigen-frequencies of the floating bridge are above the low frequency range, it was decided to perform the fitting between 0.01 to 1 Hz for the longitudinal and lateral turbulence, and 0.1 to 1 Hz for the vertical turbulence. The data was divided into six segments of 10 minutes each using a Hann window, with 50% overlap.

Four spectral models were chosen to be fitted to the data; the N400 spectrum (fitted using 1 and 2 parameters), the original Kaimal spectrum and the Harris spectrum. The Busch-Panofsky spectrum was not fitted to the data as the shape of the spectrum is identical to the Kaimal spectrum of the vertical component. Instead, a reference Busch-Panofsky spectrum was used for comparison, in addition to a reference Frøya spectrum for the longitudinal component. Equations 4.1-4.5 give the form of the non-normalized spectra to be fitted:

N400 (1 parameter): $$S_i(f) = \frac{A_i \sigma_i^2 \left(\frac{L_i^x}{\overline{U}}\right)}{\left[1+1.5A_i\left(\frac{fL_i^x}{\overline{U}}\right)\right]^{5/3}}; i = u, v, w \quad (4.1)$$

N400 (2 parameters): $$S_i(f) = \frac{A_i \sigma_i^2 \left(\frac{L_i^x}{\overline{U}}\right)}{\left[1+1.5B_i\left(\frac{fL_i^x}{\overline{U}}\right)\right]^{5/3}}; i = u, v, w \quad (4.2)$$

Kaimal spectrum: $$S_{u,v} = \frac{A_{u,v} u_*^2 \left(\frac{z}{U}\right)}{\left[1+B_{u,v}\left(\frac{fz}{U}\right)\right]^{5/3}} \quad (4.3)$$

$$S_w = \frac{A_w u_*^2 \left(\frac{z}{U}\right)}{1+B_w\left(\frac{fz}{U}\right)^{5/3}} \quad (4.4)$$

Harris spectrum: $$S_u(f) = \frac{A_u \sigma_u^2 \left(\frac{L_u^x}{\overline{U}}\right)}{\left[1+B_u\left(\frac{fL_u^x}{\overline{U}}\right)^2\right]^{5/6}} \quad (4.5)$$

The spectrum fitting for each turbulence component was done using the 'curve_fit' function (in the frequency range specified above), which uses the least-squares method to fit the parameters to the numerically estimated spectrum. [24] The results are presented in section 5.6.

4.4 Coherence fitting

To investigate the coherence characteristics of the wind, sensors A and B in Ospøya 1 and Ospøya 2 were installed at the same height with a horizontal distance of 8 meters, while sensor C was installed on the same side as sensor A with a vertical distance of 15.7 meters. Sensors A and B were used to investigate the horizontal coherence and sensors A and C were used for the vertical coherence. The criteria for performing the fitting were the same as described in the previous section.

The numerical estimation of the coherence based on the wind measurements were done using the function 'scipy.signal.csd', which estimates the cross-spectral density using Welch's method. [25] The real part of the estimated cross-spectral density was divided by the spectral density of the turbulence components at the two points considered to obtain the coherence parameter. The data was divided into eighteen segments using a Hann window, with 50% overlap, and fitted to the same frequency range as described in the previous section.

The coherence functions to be fitted were the same as Equations 2.39 and 2.40 in section 2.4. However, for the horizontal coherence, instead of fitting two parameters C_{ix} and C_{iy}, the longitudinal coherence parameter was chosen to be a fixed fraction of the lateral coherence parameter to have numerical stability in the curve fitting. The relationship between C_{ix} and C_{iy} is based on [26].

The shape of the coherence function is given by the coherence parameters under the square root in Equation 2.39. For one realization of the function, there are an infinite number of combinations of C_{ix} and C_{iy} that would give the same result, and the fitting relies on the initial guesses of the parameters. Therefore, by fitting only one parameter, C_{iy}, instead of two, the fitting is more robust. The coherence function to be fitted for the vertical component remains the same as Equation 2.40, and the new expressions for the horizontal coherence become:

$$Coh_{u,hor} = e^{\left(\frac{f}{0.5(\bar{U}(z_1)+\bar{U}(z_2))}\sqrt{\left(\left(\frac{1}{3}C_{uy}dx\right)^2+\left(C_{uy}dy\right)^2\right)}\right)}\cos\left(\frac{2\pi f}{0.5\left(\bar{U}(z_1)+\bar{U}(z_2)\right)}dx\right) \quad (4.6)$$

$$Coh_{v,hor} = e^{\left(\frac{f}{0.5(\bar{U}(z_1)+\bar{U}(z_2))}\sqrt{\left(\left(\frac{1}{4}C_{vy}dx\right)^2+\left(C_{vy}dy\right)^2\right)}\right)}\cos\left(\frac{2\pi f}{0.5\left(\bar{U}(z_1)+\bar{U}(z_2)\right)}dx\right) \quad (4.7)$$

$$Coh_{w,hor} = e^{\left(\frac{f}{0.5(\bar{U}(z_1)+\bar{U}(z_2))}\sqrt{\left(\left(\frac{1}{6}C_{wy}dx\right)^2+\left(C_{wy}dy\right)^2\right)}\right)}\cos\left(\frac{2\pi f}{0.5\left(\bar{U}(z_1)+\bar{U}(z_2)\right)}dx\right) \quad (4.8)$$

The 'curve_fit' function was used on Equations 4.6-4.8 and 2.40, with an initial guess of each coherence parameter set to be equal to the suggested values in N400. The results are presented in section 5.7.

5. Results and discussion

The results presented in this chapter are discussed by considering the distribution of the data with mean wind speed, mean wind direction and atmospheric stability. Figure 5.1 gives an overview of the directional distribution of the recorded mean wind speed for each available 1-hour event from 2015 to 2020. The left side shows the distribution for all wind speeds while the right side only shows wind speeds over 15 m/s. Based on the observations here, the mean wind speed is divided into four ranges, and for the fitted spectral and coherence parameters the data is then further divided into three directional sectors. Only three directional sectors were chosen as the amount data from the north-eastern sector is very low compared to the other sectors, as seen in Figure 5.1. To investigate the effect of atmospheric stability, the data is divided based on near-neutral, unstable, and stable conditions, and further divided into wind speeds above and under 15 m/s.

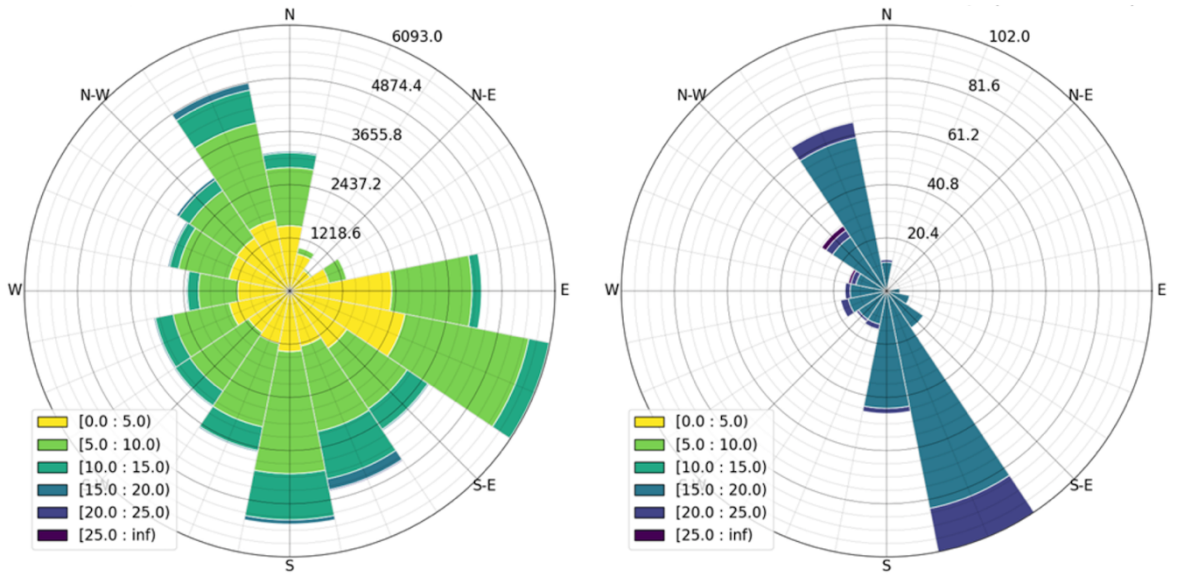

Figure 5.1: Directional distribution of measured mean wind speed with number of 1-hour events for each range.

Tables 5.1-5.3 give an overview of the requirements for the different cases, and the number of 1-hour events in each case for sensor A after checking against the criteria listed in section 4.3. Table 5.4 gives an overview of how much data was disregarded for sensor A according to the selected criteria. A total of 13 872 1-hour events were available, and the amount of disregarded data is given as a percentage of this. Most of the disregarded data was due to non-stationarity, followed by the criteria for the measured mean wind speed. In the following sections, unless stated otherwise, the results presented are from sensor A only.

Table 5.1: Overview of considered mean wind speed ranges and number of 1-hour events in each range.

Mean wind speed ranges		No. of 1-hour events			
		Osp1	**Osp2**	**Svar**	**Synn**
U1	6-10 m/s	4581	3801	3195	3617
U2	10-15 m/s	2487	2206	1044	1399
U3	15-20 m/s	508	483	133	278
U4	> 20 m/s	66	80	24	28

Table 5.2: Overview of considered directional sectors and number of 1-hour events in each sector.

Directional sectors		No. of 1-hour events			
		Osp1	**Osp2**	**Svar**	**Synn**
Sector 1	90˚-180˚	3373	2829	1222	2209
Sector 2	180˚-270˚	1863	1505	1101	1346
Sector 3	270˚-360˚	2131	1797	1656	1462

Table 5.3: Overview of considered stability cases and number of 1-hour events in each case.. S1-S2, S3-S4 and S5-S6 represent unstable, near-neutral, and stable conditions respectively.

Stability case			No. of 1-hour events			
			Osp1	**Osp2**	**Svar**	**Synn**
S1	$\zeta < -0.1$	U < 15 m/s	4570	2988	3710	3245
S2	$\zeta < -0.1$	U > 15 m/s	42	85	22	84
S3	$-0.1 < \zeta < 0.1$	U < 15 m/s	4593	4830	5822	3606
S4	$-0.1 < \zeta < 0.1$	U > 15 m/s	508	438	126	129
S5	$\zeta > 0.1$	U < 15 m/s	2836	2142	3301	3544
S6	$\zeta > 0.1$	U > 15 m/s	24	40	9	93

Table 5.4: Percentage of disregarded data according to selected criteria.

	Osp1	**Osp2**	**Svar**	**Synn**
Missing data > 5%	7%	21.9%	4.2%	20.7%
Non-stationarity of mean wind speed	54.7%	45.8%	74.1%	51.7%
Non-stationarity of mean wind direction	42.4%	34.6%	61.3%	61.9%
Mean wind speed < 6 m/s	35.5%	28.5%	61.9%	38.8%

5.1 Angle of attack

It is interesting to see the variation of the angle of attack at the different wind masts to understand the effect of the surrounding topography. The angle of attack is determined by equation:

$$\alpha = \tan^{-1}\left(\frac{W}{U}\right) \tag{5.1}$$

where W and U are the vertical and longitudinal components of the mean wind vector respectively. U is always positive along the mean wind direction, so the sign of α depends on the direction of W.

Figure 5.2 shows the distribution of α at Ospøya 1 with the local topography and Figure 5.3 shows the distribution for different wind speed ranges. The color of each data point represents the corresponding 1-hour mean wind speed. As the wind speed range increases, the dominant directions are for winds coming from the north-west and south. For U1, positive values of α are seen from around 290˚ to 350˚ and 10˚ to 160˚. On the north-western side of the wind mast's location, there is a drop in the terrain from approximately 24 to 11 meters, so the vertical component of the wind vector is positive. Similar steep terrain is present on the eastern side of the wind mast, resulting in positive values of α. From around 160˚ to 250˚, the terrain adjacent to the wind mast is relatively flat. The vertical component of the wind vector tends to be negative in this sector, resulting in negative values of α. The lowest values occur for winds coming directly from the south, due to the presence of a hill with a height of approximately 50 meters. The negative values for winds coming from the west were not expected as there is no drop in the terrain on this side, however these values may be due to the islands west of Ospøya which range in height from 0-70 meters.

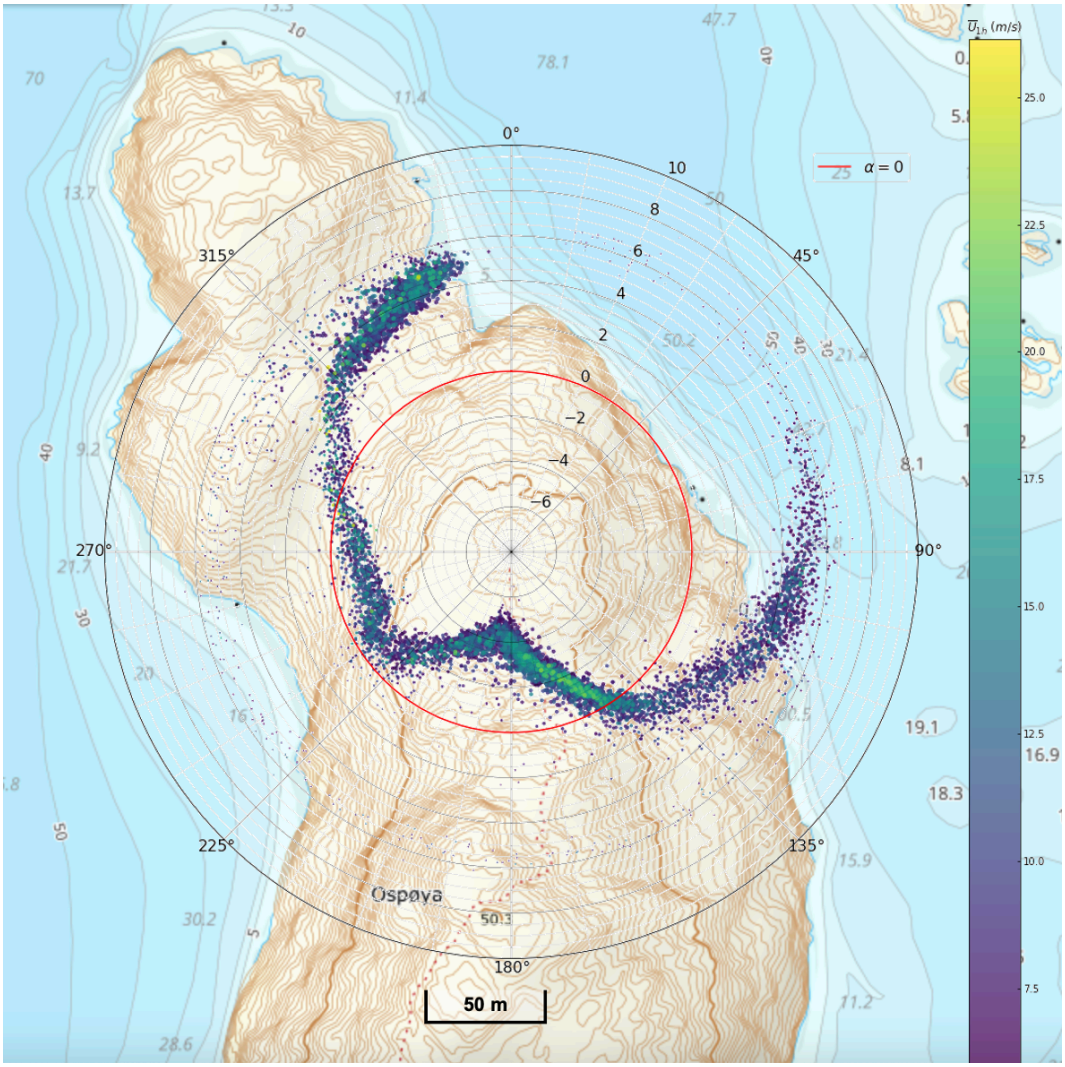

Figure 5.2: Angle of attack at Ospøya 1. Contour plot is taken from Norgeskart. [27]

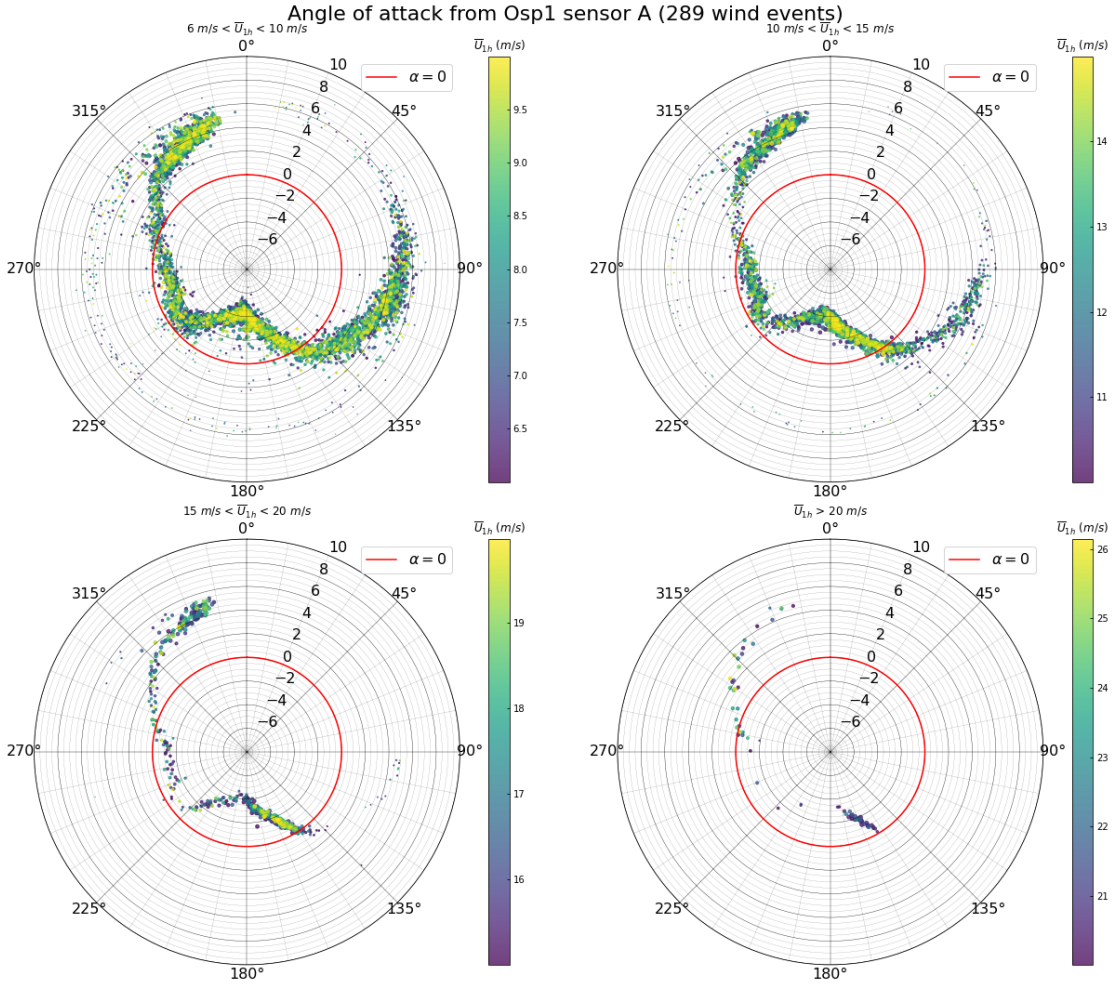

Figure 5.3: Distribution of angle of attack with direction and mean wind speed at Osp1.

Figure 5.4 shows the distribution of α at Ospøya 2 with the local topography and Figure 5.5 shows the distribution for different wind speed ranges. At Ospøya 2, the mountain is located north of the wind mast, so values for α are negative from around 270˚ to 350˚. Since Ospøya 2 is located higher than Ospøya 1, the negative values are smaller in comparison. For the remaining wind directions, the wind mast is located higher than the surrounding terrain, resulting in a positive vertical component of the wind vector and thus positive values for α. The terrain increases in steepness when moving from west to south, and south to east, giving the results as shown in the figure below. The values for α range from -4˚ to 8˚, with the highest values coming from south-east. With increasing wind speed range, the dominant wind directions are from the north-west and south-east, similar to Ospøya 1.

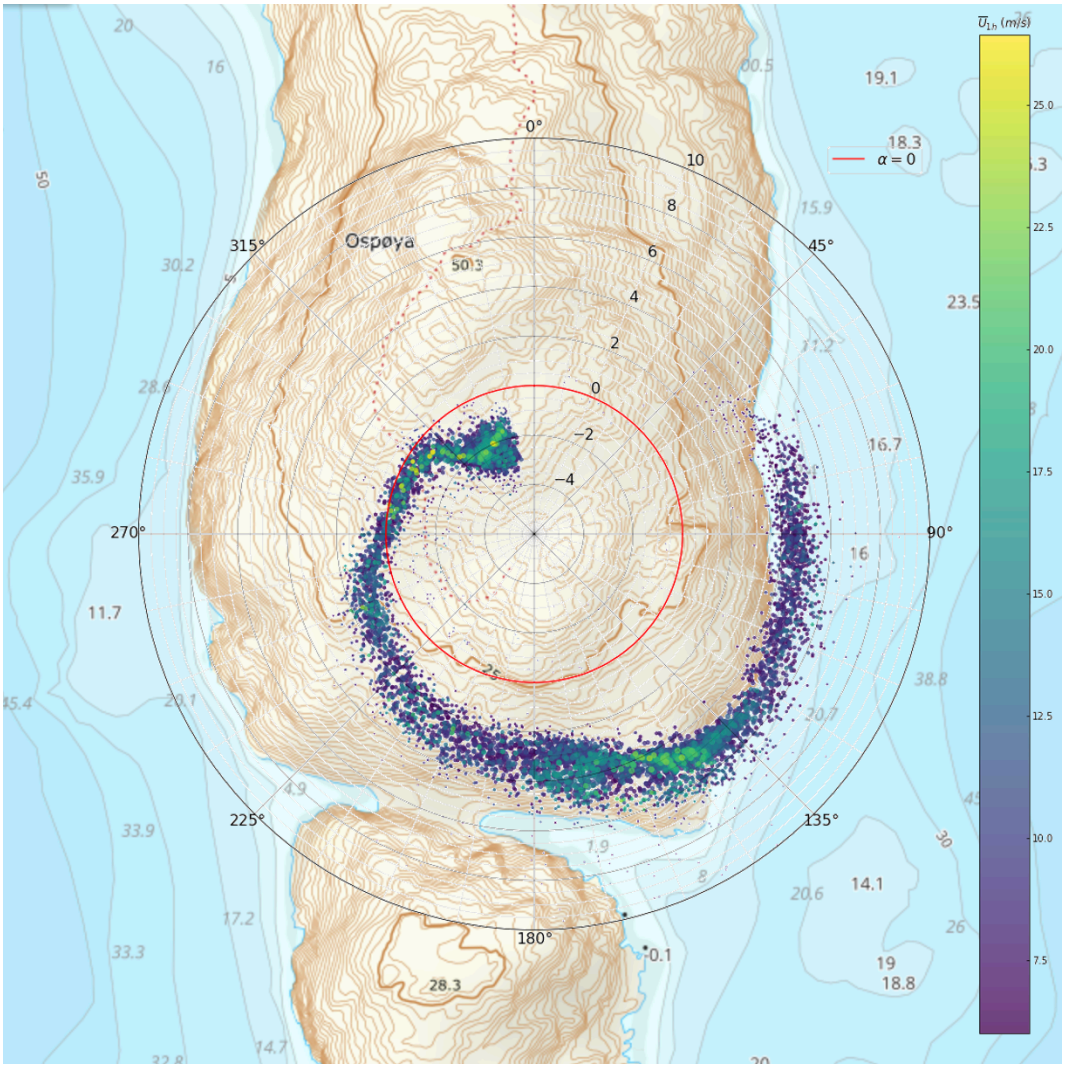

Figure 5.4: Angle of attack at Ospøya 2. Contour plot taken from Norgeskart. [27]

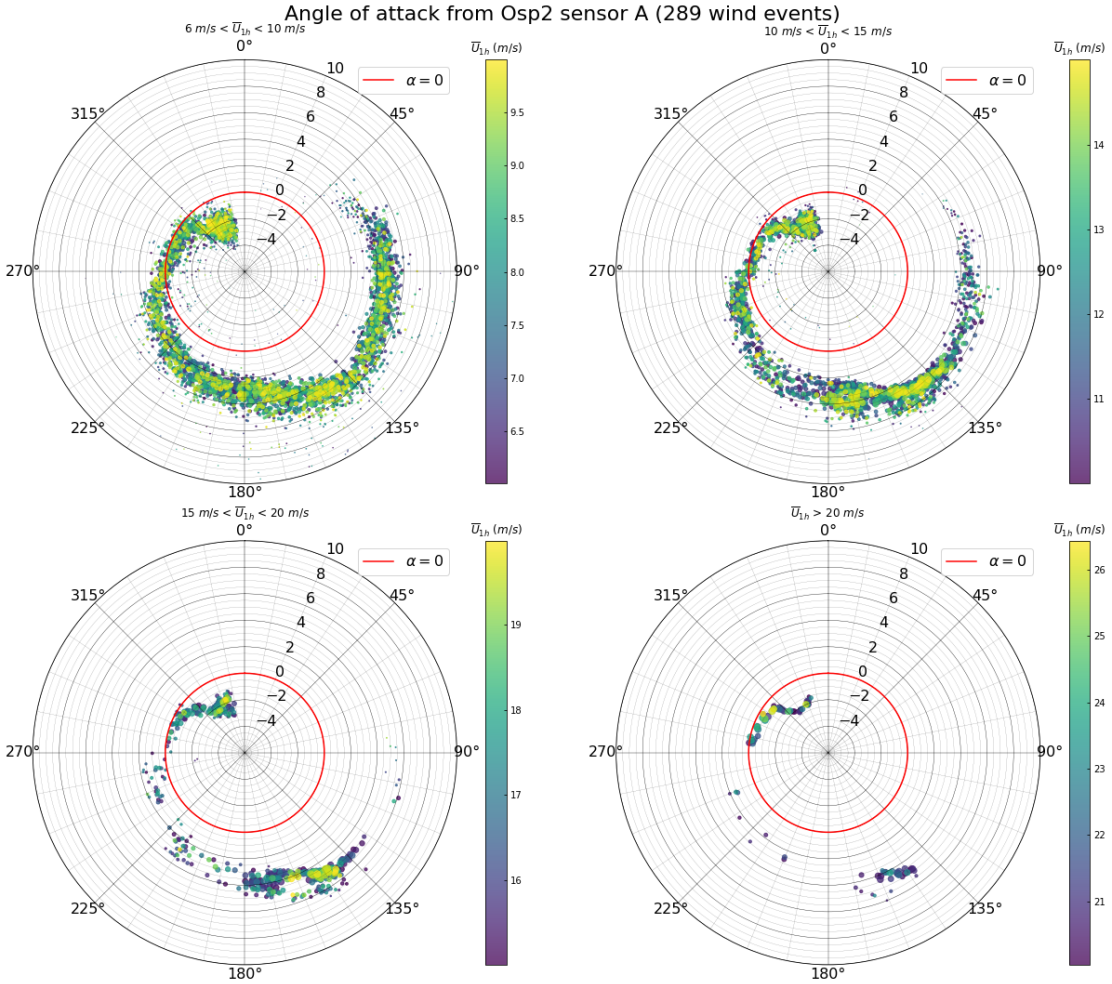

Figure 5.5: Distribution of angle of attack with direction and mean wind speed at Osp2.

Figure 5.6 shows the distribution of α at Svarvhelleholmen with the local topography and Figure 5.7 shows the distribution for different wind speed ranges. Southern winds, around 110˚ to 250˚, travel from terrain as high as 300 meters when reaching the wind mast, resulting in large negative values for α. On the north-western and north-eastern side, the terrain height does not vary as greatly but the magnitude of α is nearly similar to those from the south. The dominant wind directions at Svarvhelleholmen are mainly from the west and north-west.

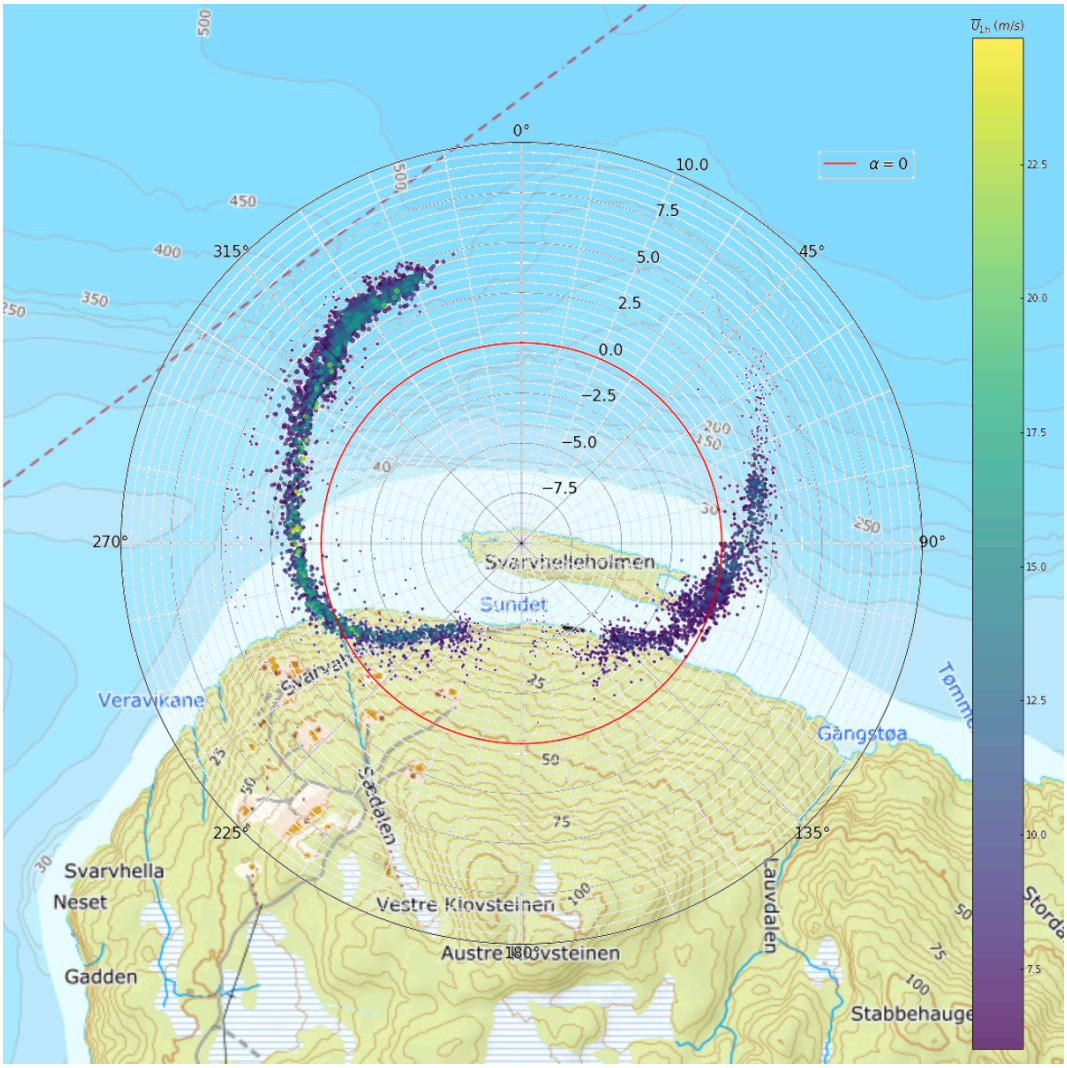

Figure 5.6: Angle of attack at Svarvhelleholmen. Contour plot taken from Norgeskart. [27]

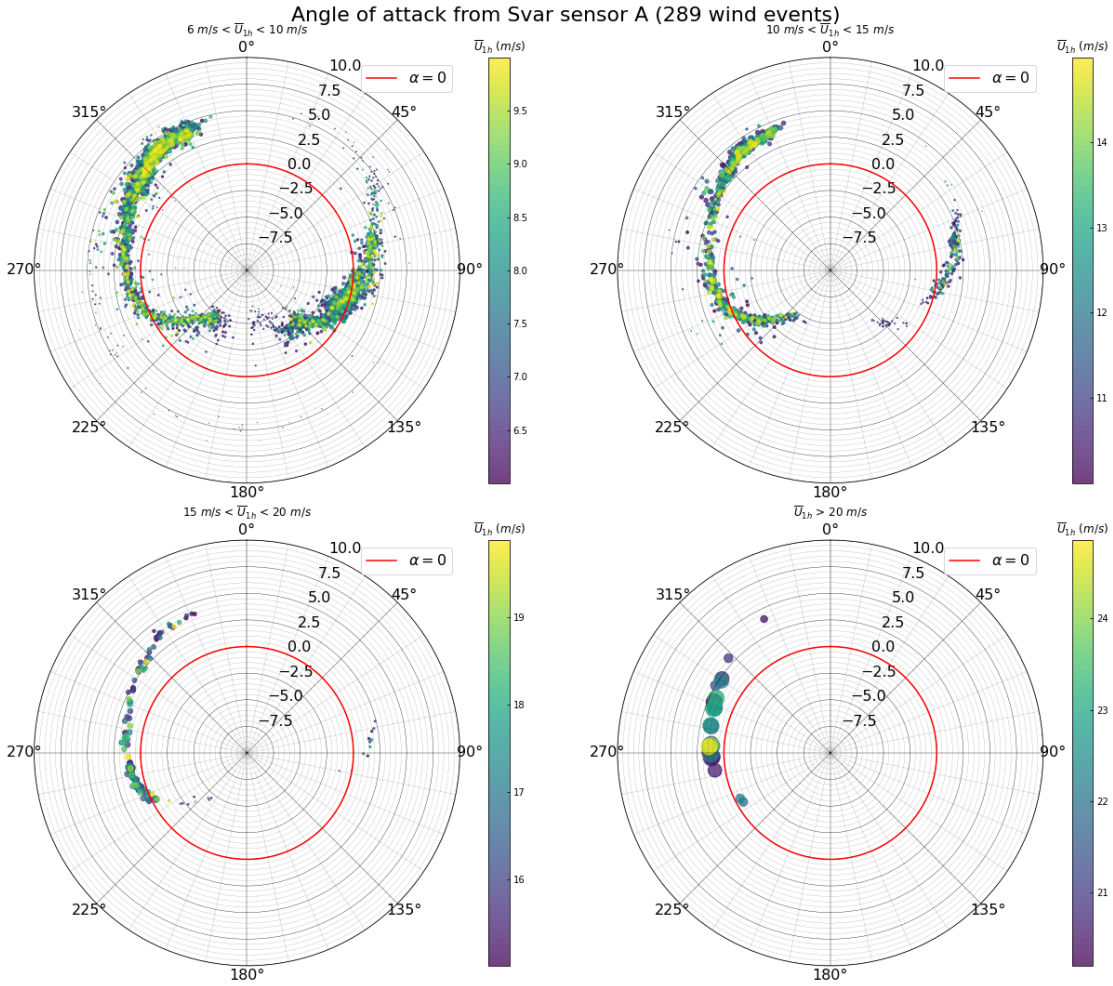

Figure 5.7: Distribution of angle of attack with direction and mean wind speed at Svar.

Figure 5.8 shows the distribution of α at Synnøytangen with the local topography and Figure 5.9 shows the distribution for different wind speed ranges. Similar to the other wind masts, the immediate surrounding terrain affects the magnitude of α and whether it is positive or negative relative to the horizontal plane. From 160˚ to 320˚ the wind is travelling up the terrain to reach the wind mast, therefore the values for α are positive. The negative values are however harder to explain, as the wind mast is located the highest relative to its surroundings. A possible explanation may be that the mainland to the east which varies in height up to 100 meters. The highest wind speeds come from the north-west and south-west, as seen for U4.

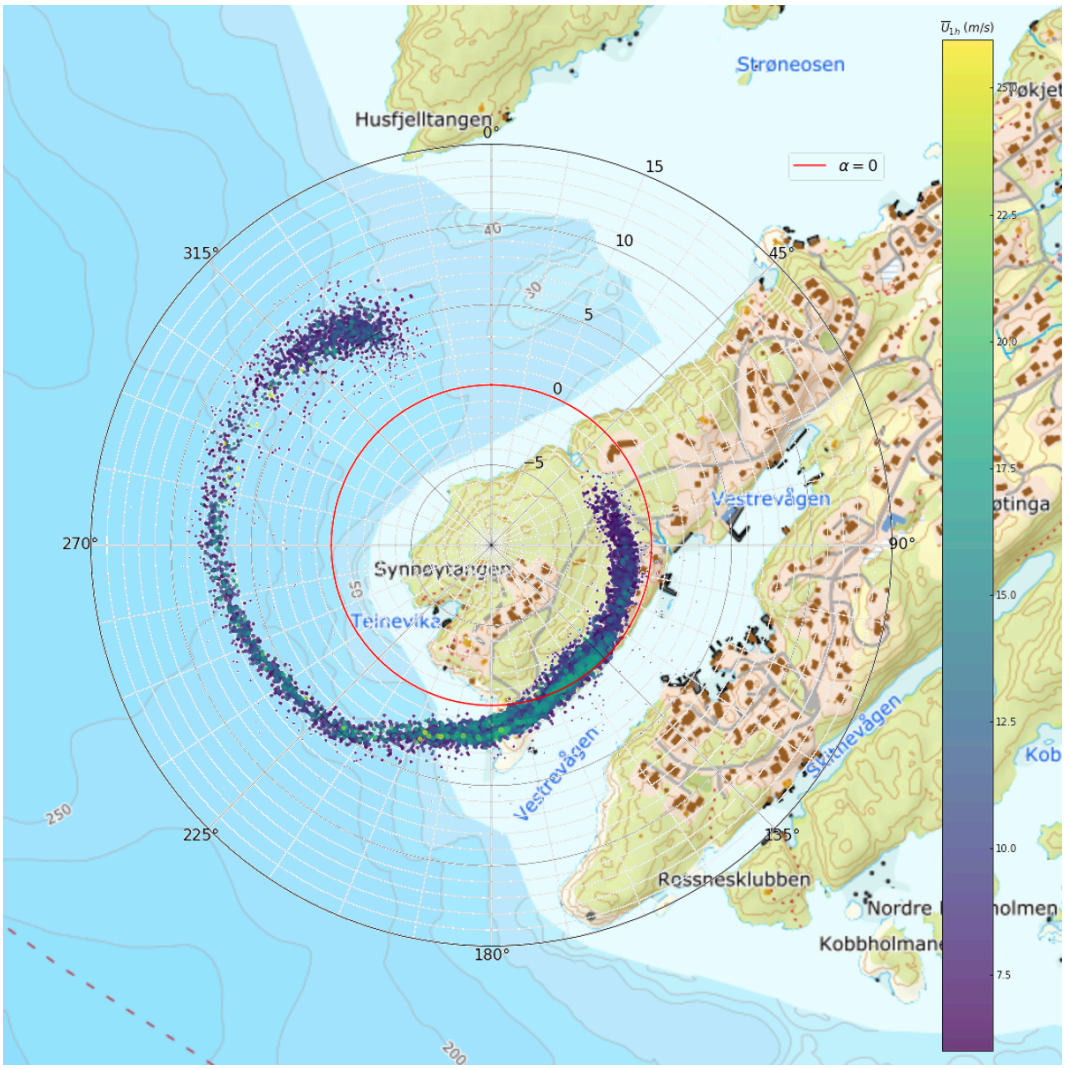

Figure 5.8: Angle of attack at Synnøytangen. Contour plot taken from Norgeskart. [27]

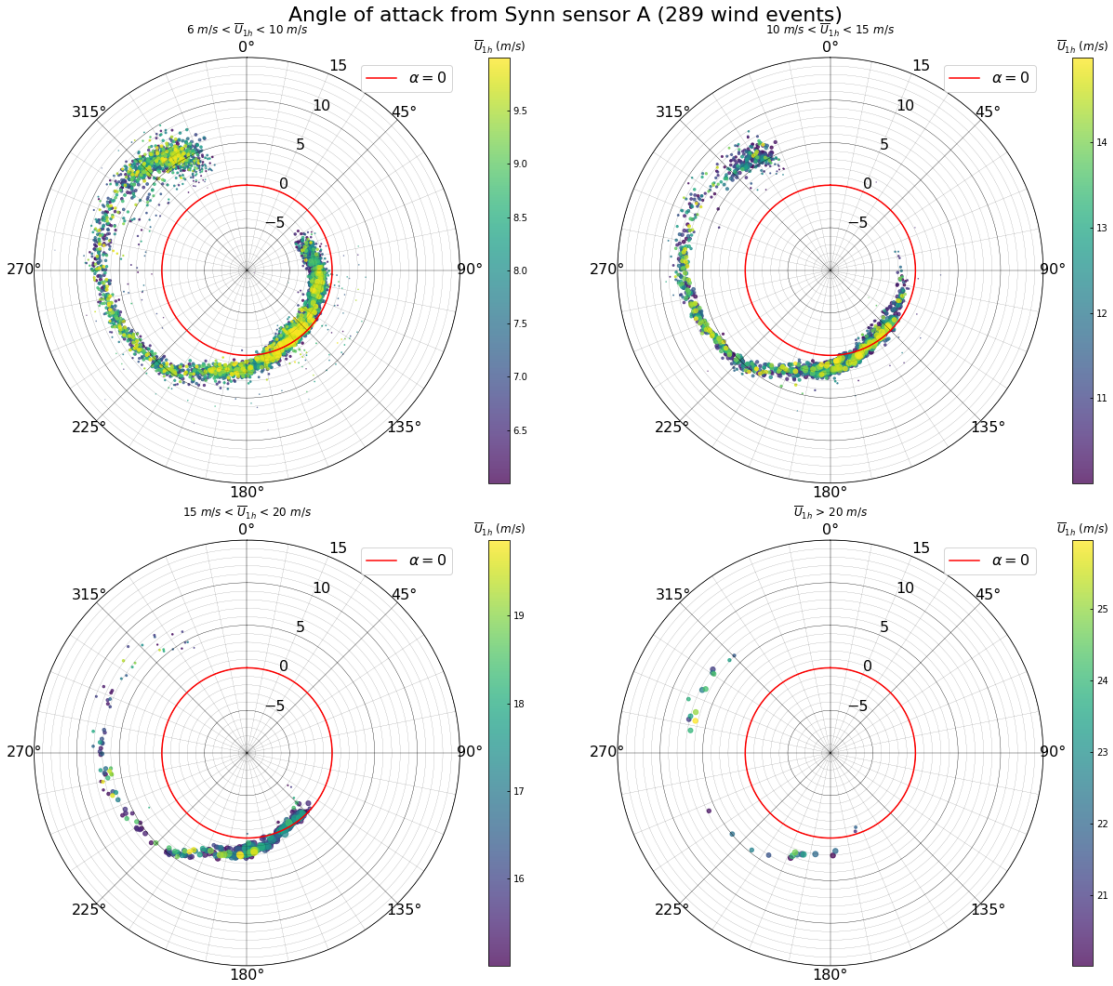

Figure 5.9: Distribution of angle of attack with direction and mean wind speed at Synn.

To verify that the values for α presented above are not due to other sources such as installation errors, each of the three sensors are plotted against each other for all the wind masts in Figures 5.10-5.13. The figures show generally consistent values, with good agreement between all sensors. This indicates that there were no installation issues with the sensors. The exceptions are the lowest sensors at Ospøya 1 and Synnøytangen. Sensor C at Synnøytangen is affected by the canopy in the area, which influences the incoming wind, however sensor C at Ospøya 1 is not affected by the same issue, making the reason for the deviations unclear.

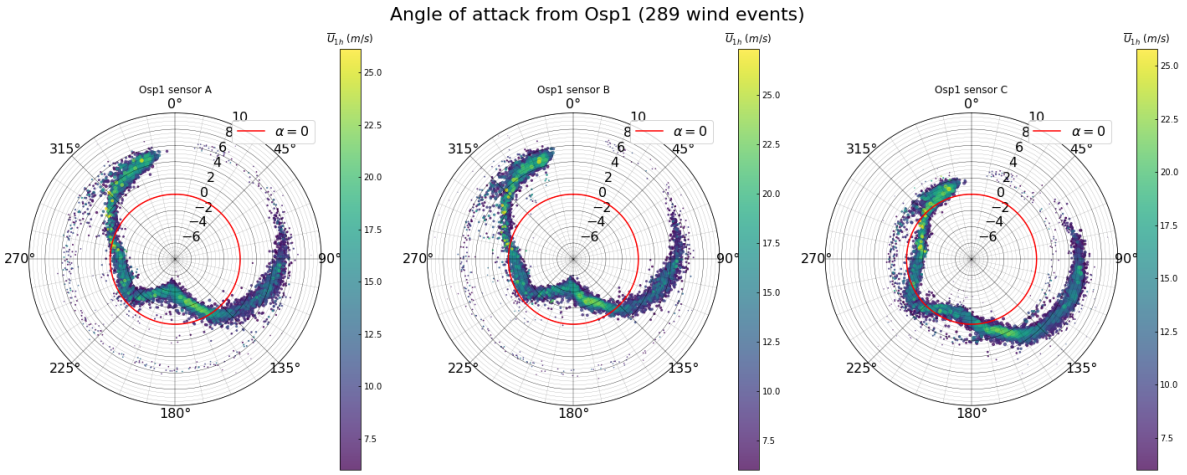

Figure 5.10: Comparison of angle of attack between all sensors at Osp1.

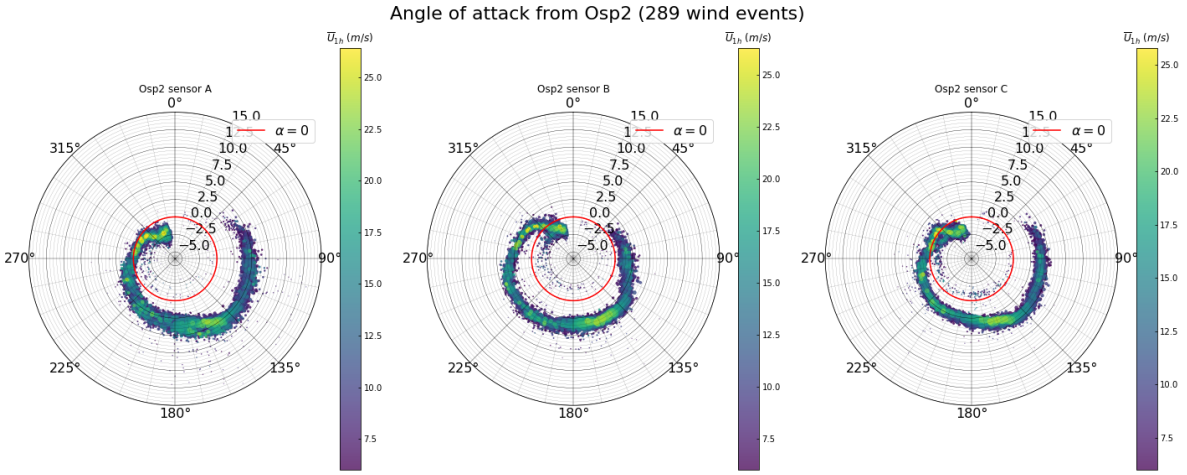

Figure 5.11: Comparison of angle of attack between all sensors at Osp2.

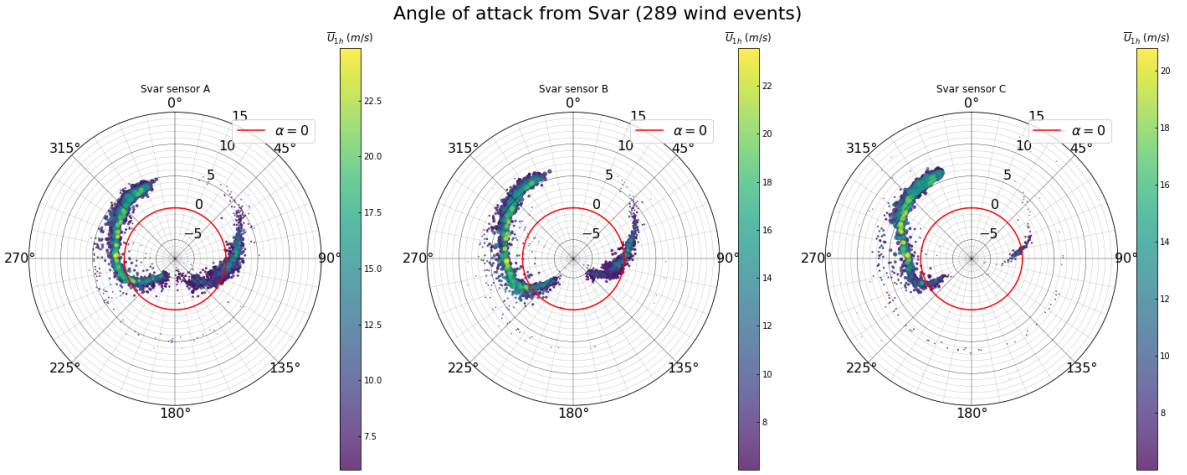

Figure 5.12: Comparison of angle of attack between all sensors at Svar.

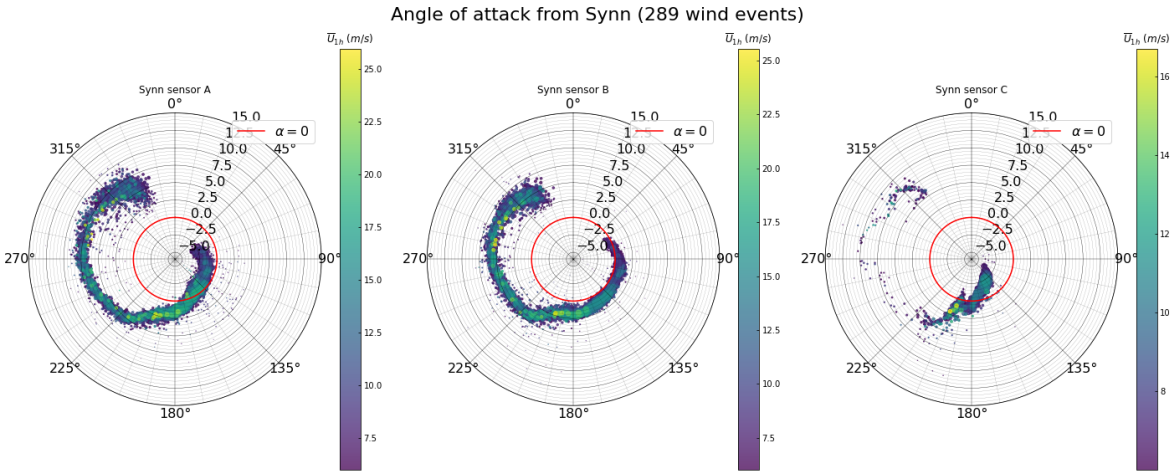

Figure 5.13: Comparison of angle of attack between all sensors at Synn.

5.2 Turbulence intensity

5.2.1 Directional distribution

Figures 5.14-5.16 show the distribution of the longitudinal, lateral, and vertical turbulence intensities calculated from Equations 2.13-2.15, based on wind measurements at the fjord. Results from the top sensors at each of the four wind masts are presented in the figures. Sensor B at Ospøya 1 and 2 were omitted as they showed a similar result as sensor A. The turbulence intensities calculated based on N400 are also shown for reference; at a height of 48.8 meters, which is the height of the top sensors, $I_u = 0.12$, $I_v = 0.09$ and $I_w = 0.06$.

At Ospøya 1 and 2, the highest turbulence intensities come from southern and western winds, ranging from around 0.14 to 0.26 for the longitudinal component. The wind coming from east is significantly less turbulent in comparison, with values lower than the reference. This is expected as Ospøya is exposed to an open fetch of water several kilometers long on its eastern side, while on its western and southern side the wind is travelling over a combination of water and land comprised of islands with varying size and height.

The wind coming from south is most turbulent at Svarvhelleholmen for U1, due to the presence of steep terrain with heights up to over 300 meters. However, because the turbulence intensity is inversely proportional with the mean wind speed, the values are most likely overestimated at the low wind speed ranges. [8] For north-western and eastern winds, I_u ranges up to around 0.2 to 0.3, which is higher compared to the eastern turbulence intensity at Ospøya. Similarly, at Synnøytangen the most turbulent wind comes from the east, south-east and north-west,

where the wind must travel over land, while from the south-west the wind is less turbulent when travelling over water.

Values for I_v are larger than the N400 reference regardless of wind direction. The reference value for I_w captures most of the observations from the south-west at Synnøytangen and east at Ospøya 1 and 2, where the wind flow is over water in both cases. However, for wind flow over land, I_w generally has a higher value than what is suggested by N400.

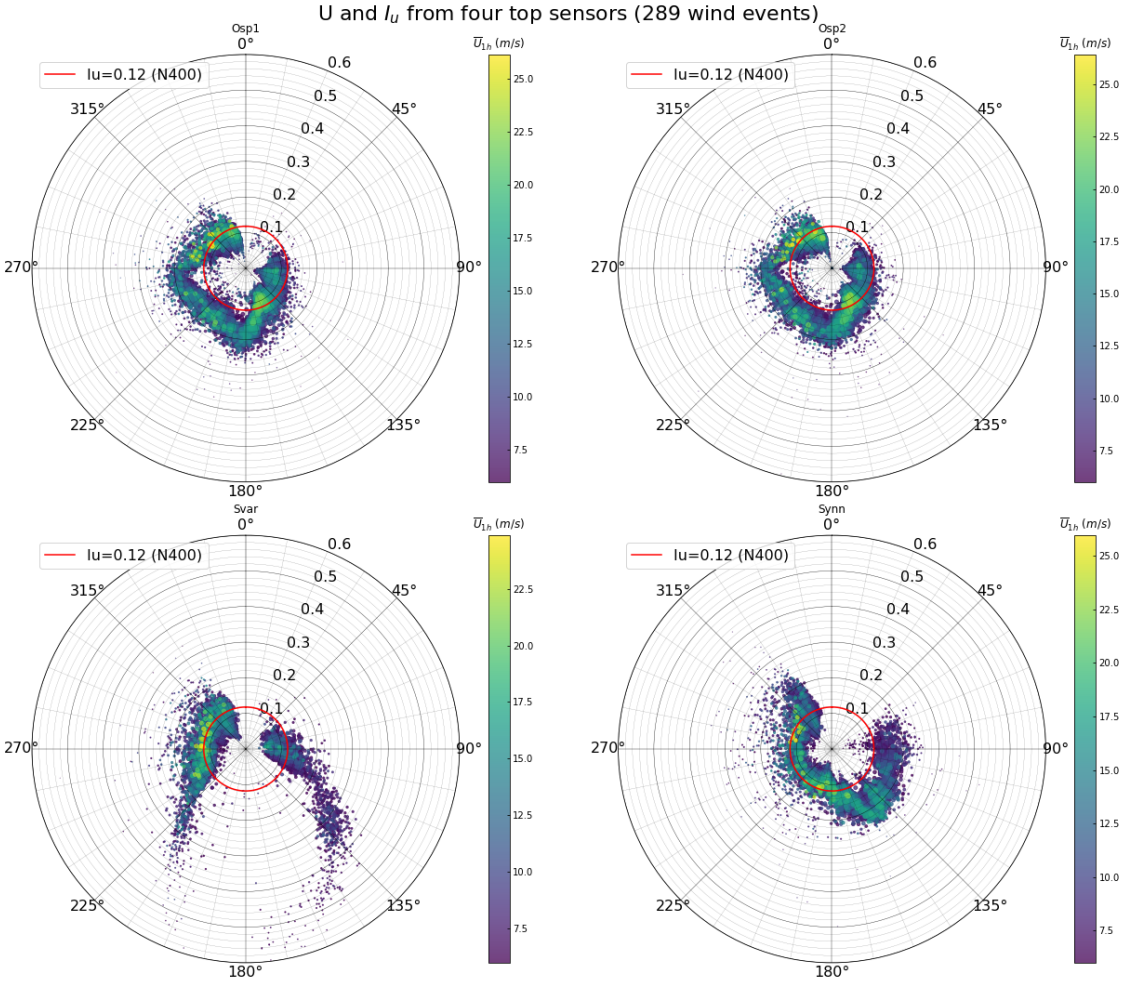

Figure 5.14: Directional distribution of the longitudinal turbulence intensity for the four top sensors.

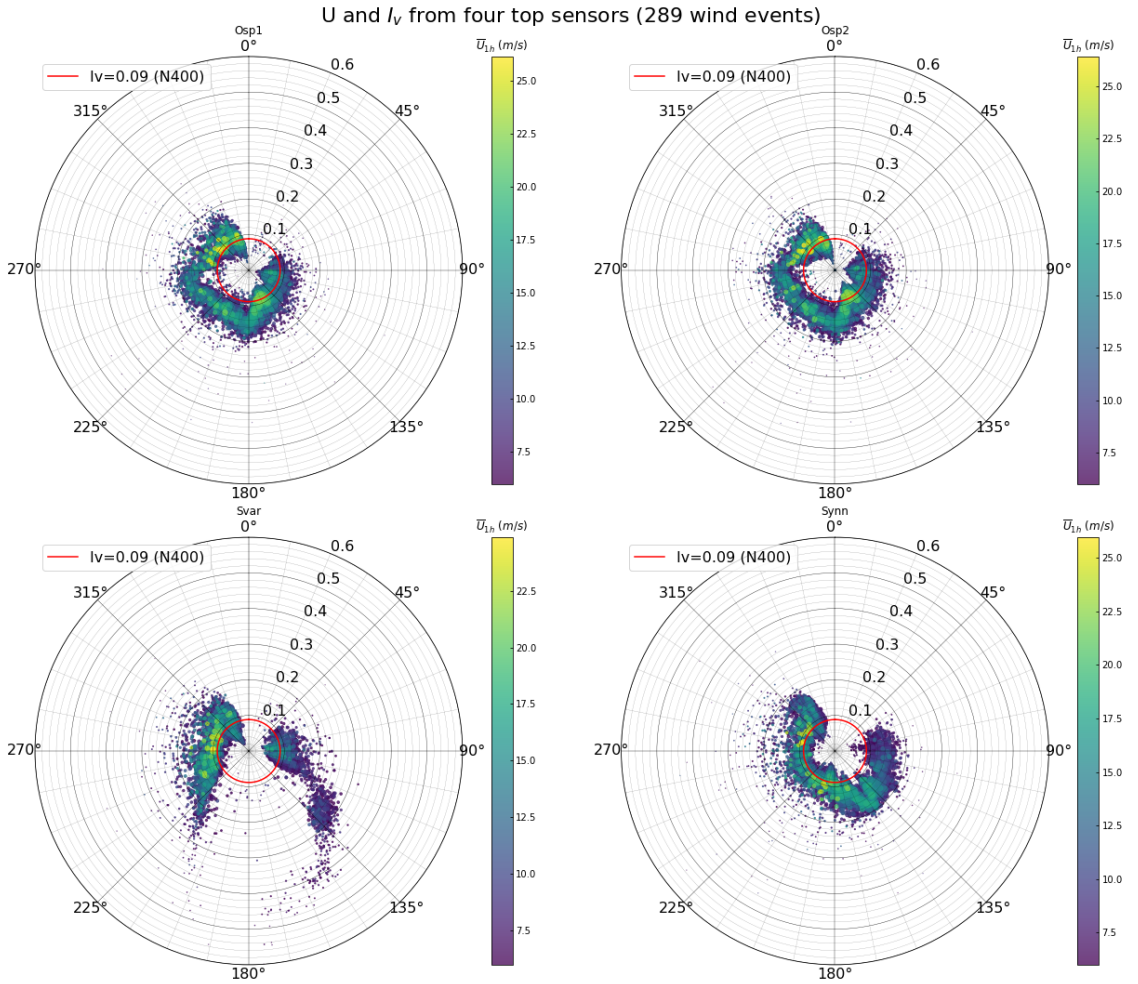

Figure 5.15: Directional distribution of the lateral turbulence intensity for the four top sensors.

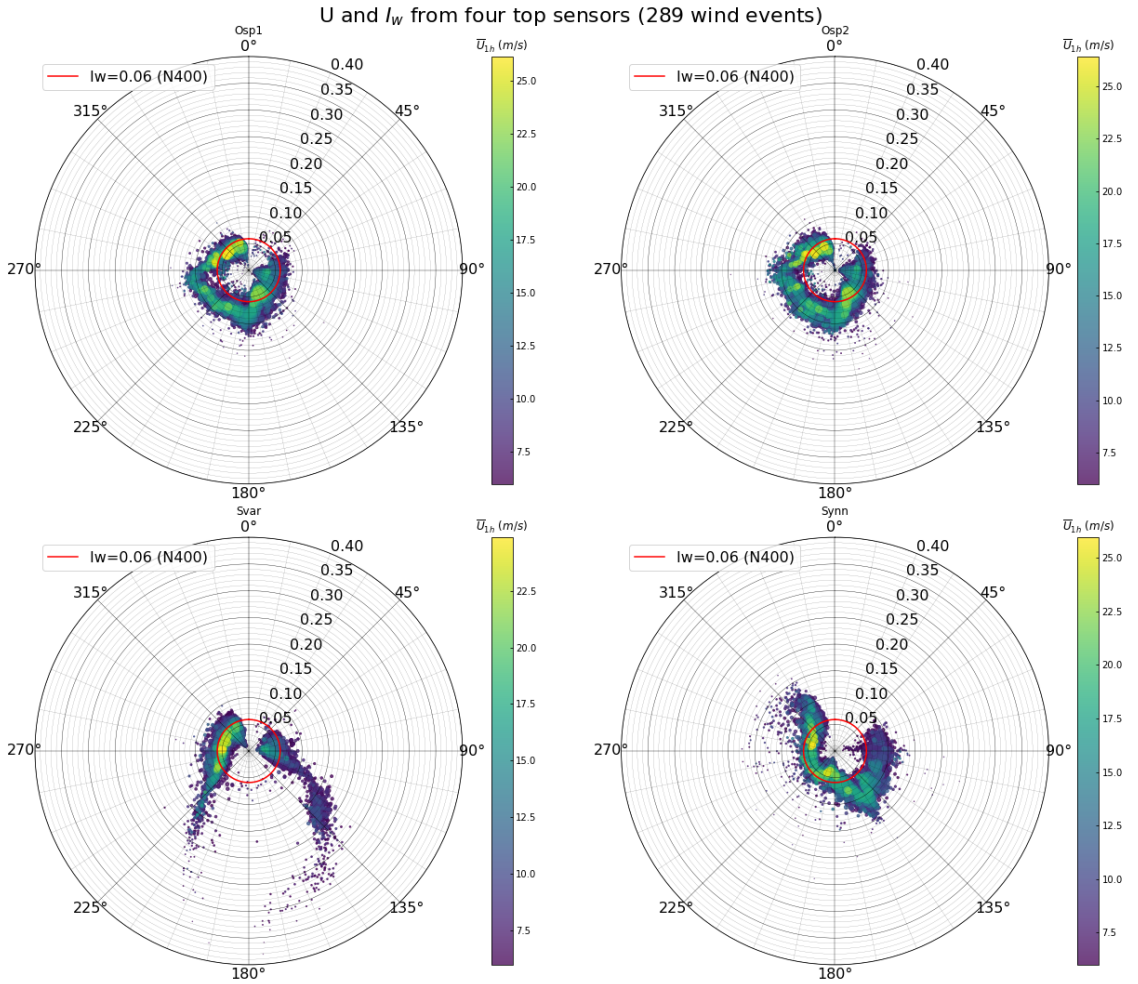

Figure 5.16: Directional distribution of the vertical turbulence intensity for the four top sensors.

5.2.2 Distribution with mean wind speed

In this section, results from Ospøya 2 were omitted as they were almost identical to the results from Ospøya 1. The remaining three wind masts are compared with each other.

Figures 5.17-5.19 show the distribution of the longitudinal turbulence intensity with the four chosen wind speed ranges. At Ospøya in ranges U1 and U2, most of the values from the south-east and north-west fall under the reference value from N400. Wind from the south-west at Ospøya consistently has turbulence intensities larger than the reference value for all wind speed ranges. At the same time, as the wind speed increases, the values for the turbulence intensity seem to fall closer to the reference value. North-western and south-eastern winds dominate for the highest wind speeds.

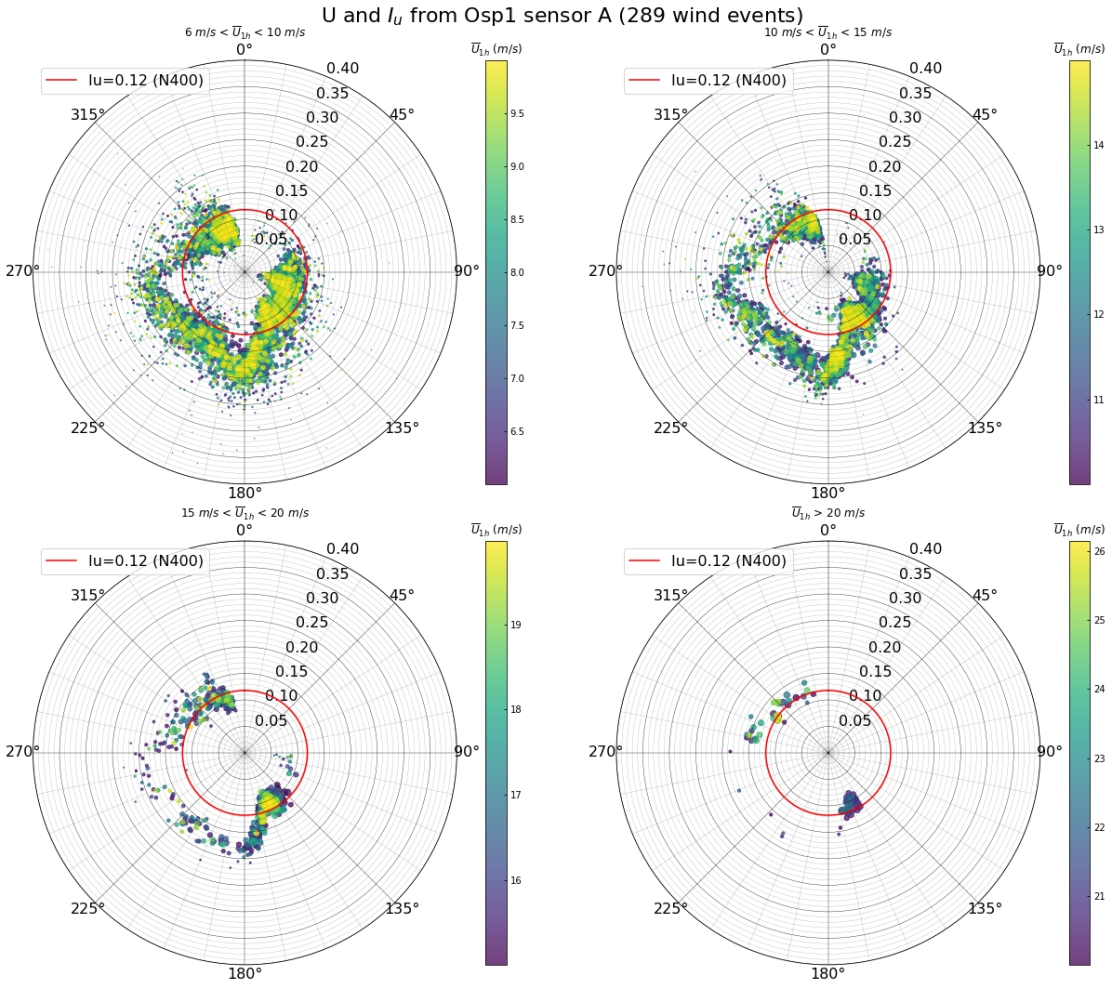

Figure 5.17: Distribution of the longitudinal turbulence intensity with mean wind speed at Osp1.

Svarvhelleholmen shows the largest turbulence intensities for U1, with the lowest wind speeds coming from the south giving the highest turbulence intensity. There is a clear decrease in turbulence intensity from U1 to U2. Similar to Ospøya, as the mean wind speed increases, the turbulence intensity tends to be closer to the reference value, but still underestimated. The dominant wind direction is west for the ranges U3 and U4.

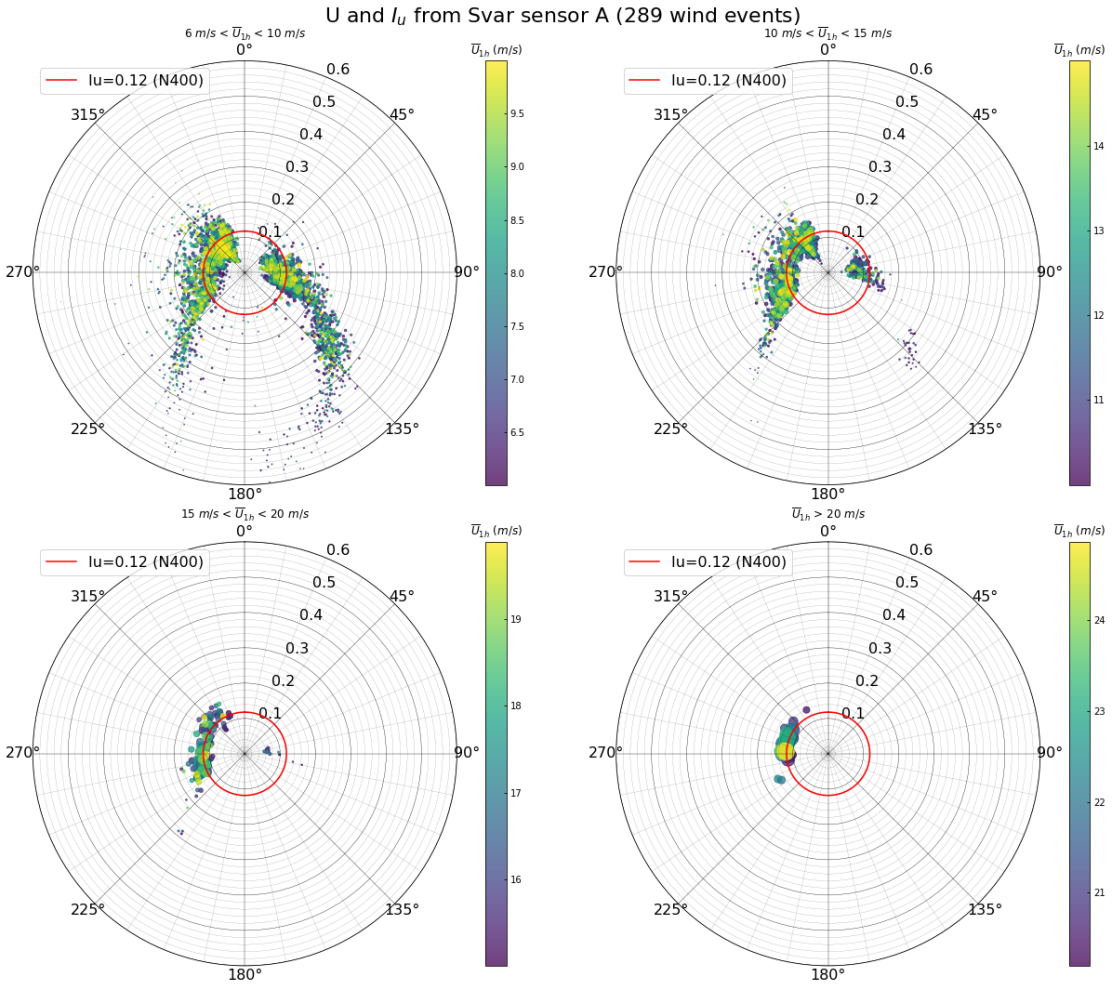

Figure 5.18: Distribution of the longitudinal turbulence intensity with mean wind speed at Svar.

At Synnøytangen for winds coming from the south-west, most of the turbulence intensity values are smaller than the reference value, regardless of wind speed range. The opposite is the case for winds from the north-west and south-east. The highest wind speeds come from the south and west, which correspond to wind flow over land and water respectively. The turbulence intensity is closest to the reference value for the wind speed range U4.

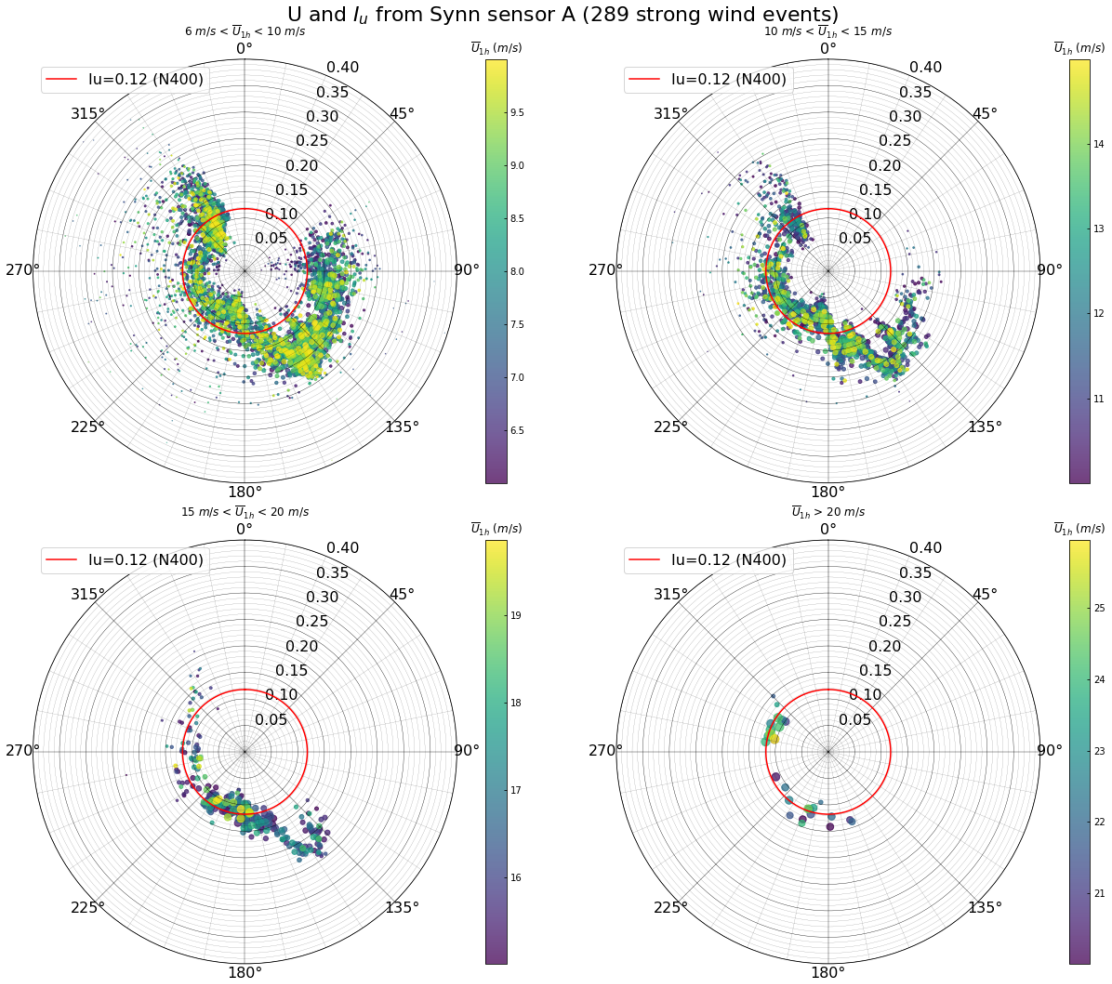

Figure 5.19: Distribution of the longitudinal turbulence intensity with mean wind speed at Synn.

Figures 5.20-5.22 show the distribution of the lateral turbulence intensity with the four chosen wind speed ranges. The results for the lateral turbulence intensity show similar distributions to the longitudinal turbulence intensity, the main difference being that the N400 reference value in general does not cover much of the actual measured turbulence intensities, as was concluded in section 5.2.1. This is the case for all wind masts, regardless of wind speed range or direction.

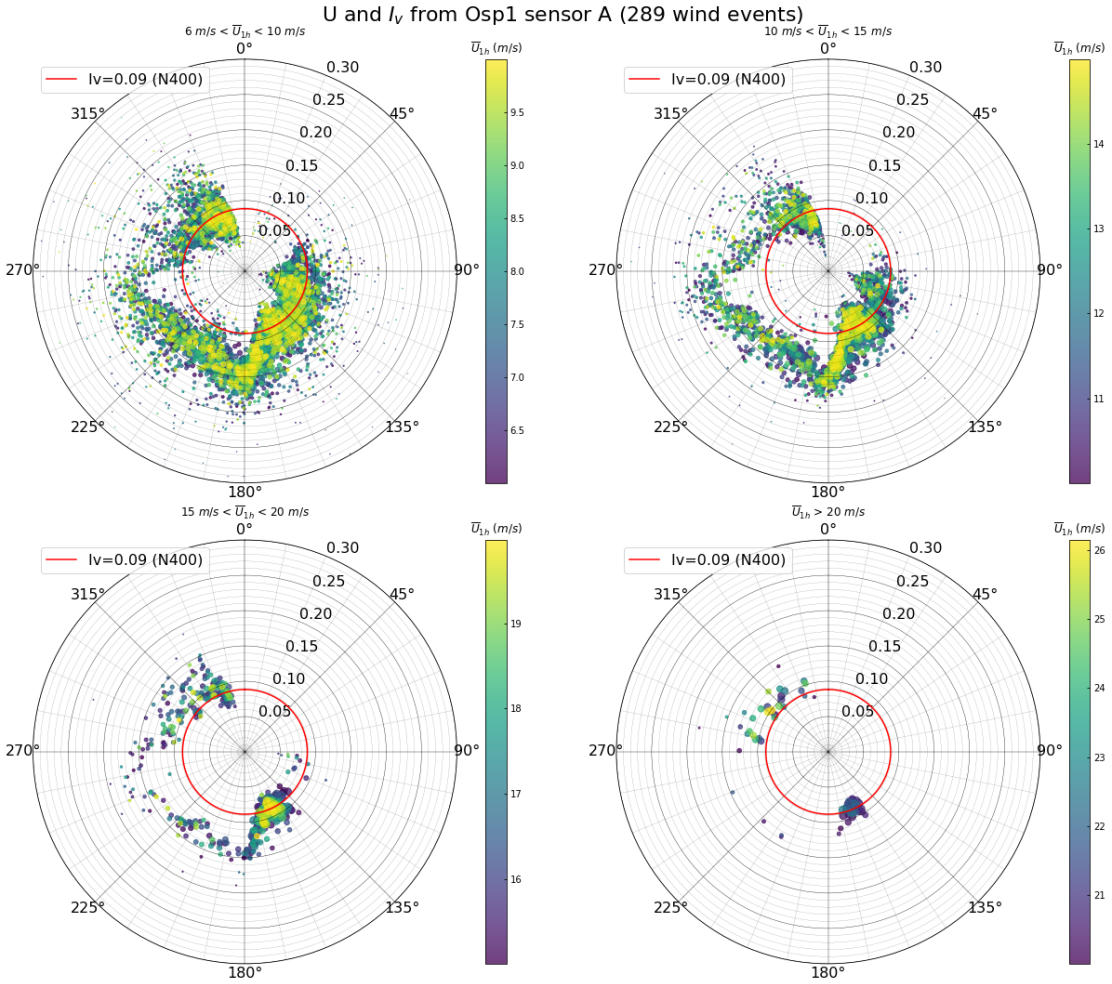

Figure 5.20: Distribution of the lateral turbulence intensity with mean wind speed at Osp1.

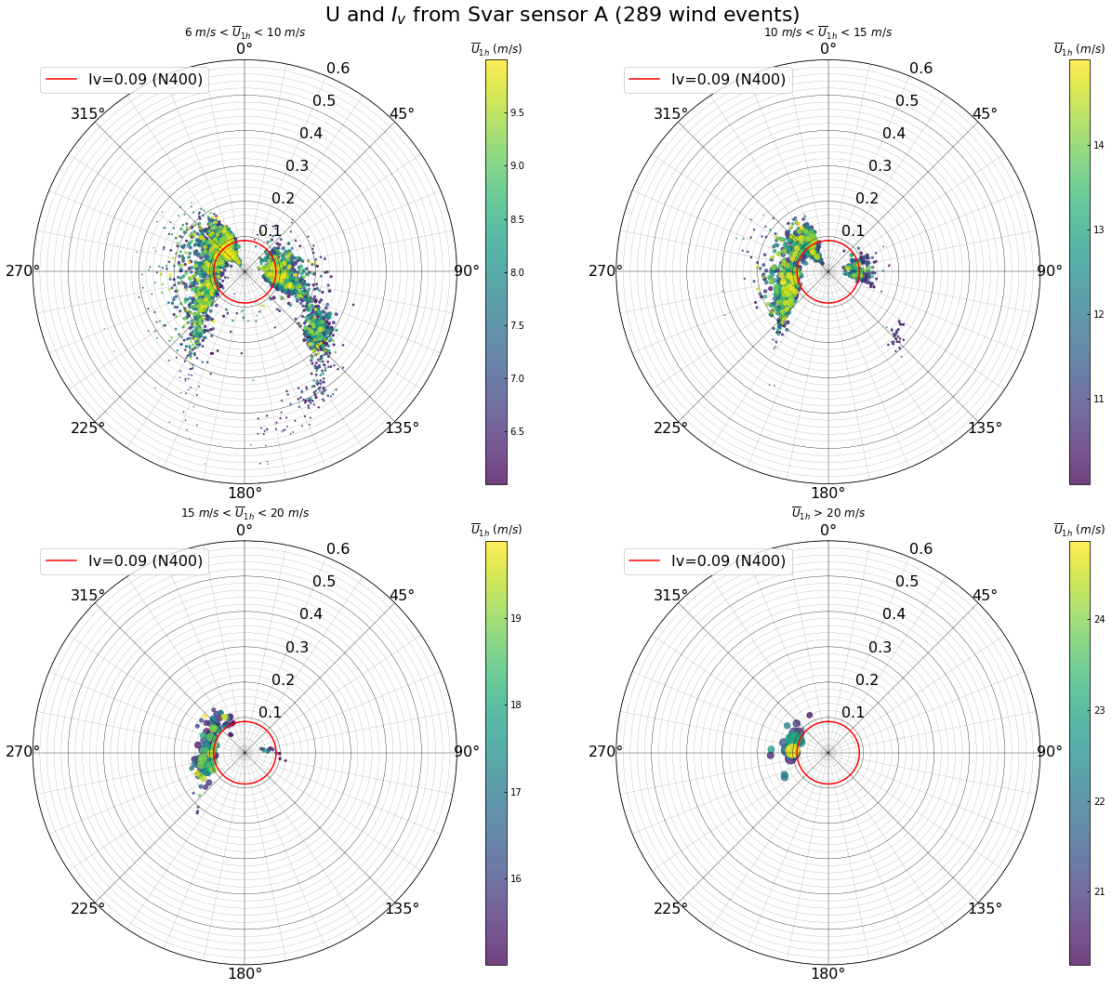

Figure 5.21: Distribution of the lateral turbulence intensity with mean wind speed at Svar.

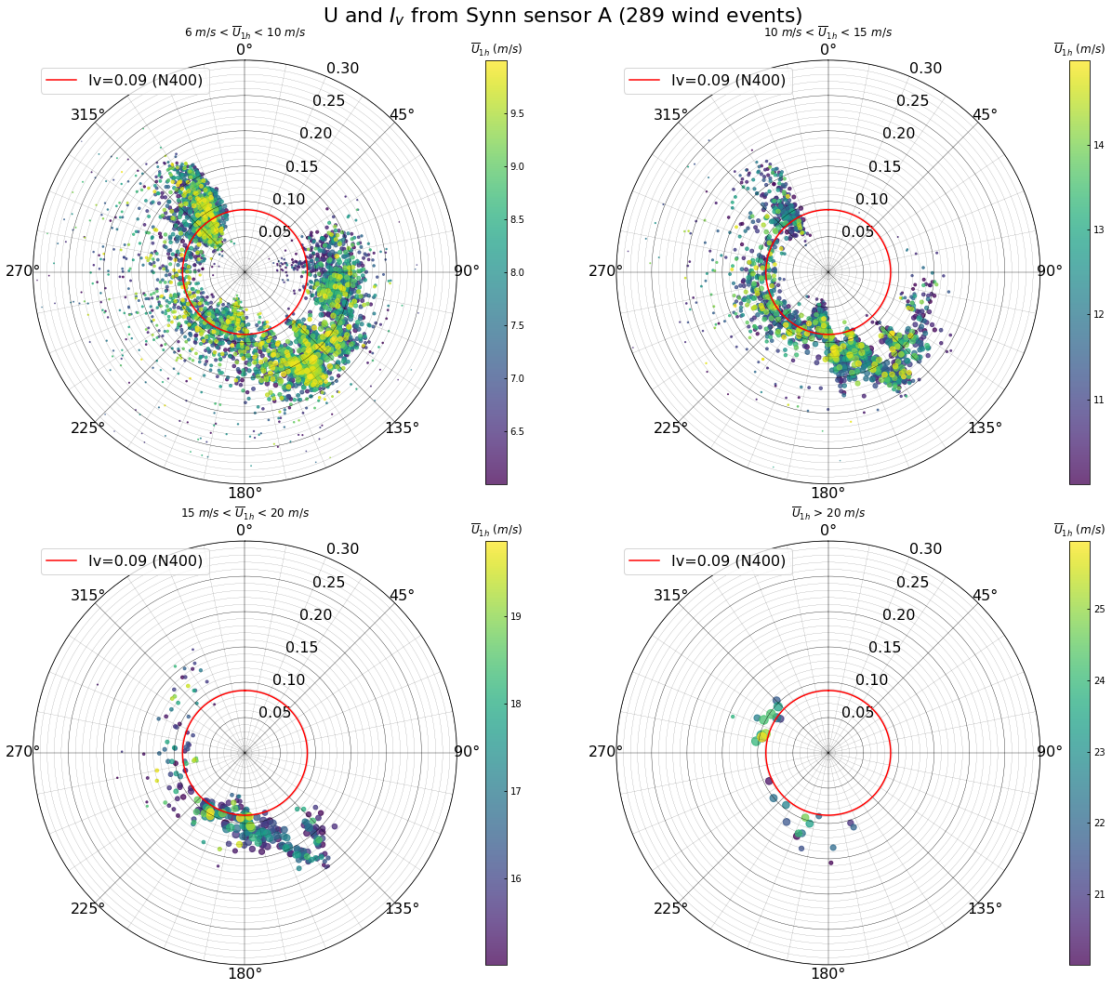

Figure 5.22: Distribution of the lateral turbulence intensity with mean wind speed at Synn.

Figures 5.23-5.25 show the distribution of the vertical turbulence intensity with the four chosen wind speed ranges. Overall, I_w has the lowest values compared to the other turbulence components. Svarvhelleholmen shows the highest turbulence intensities for the U1 range, but the value decreases significantly as the wind speed range increases. The N400 reference value underestimates the turbulence intensities in ranges U1 and U2 for the dominant wind directions mentioned in section 5.2.

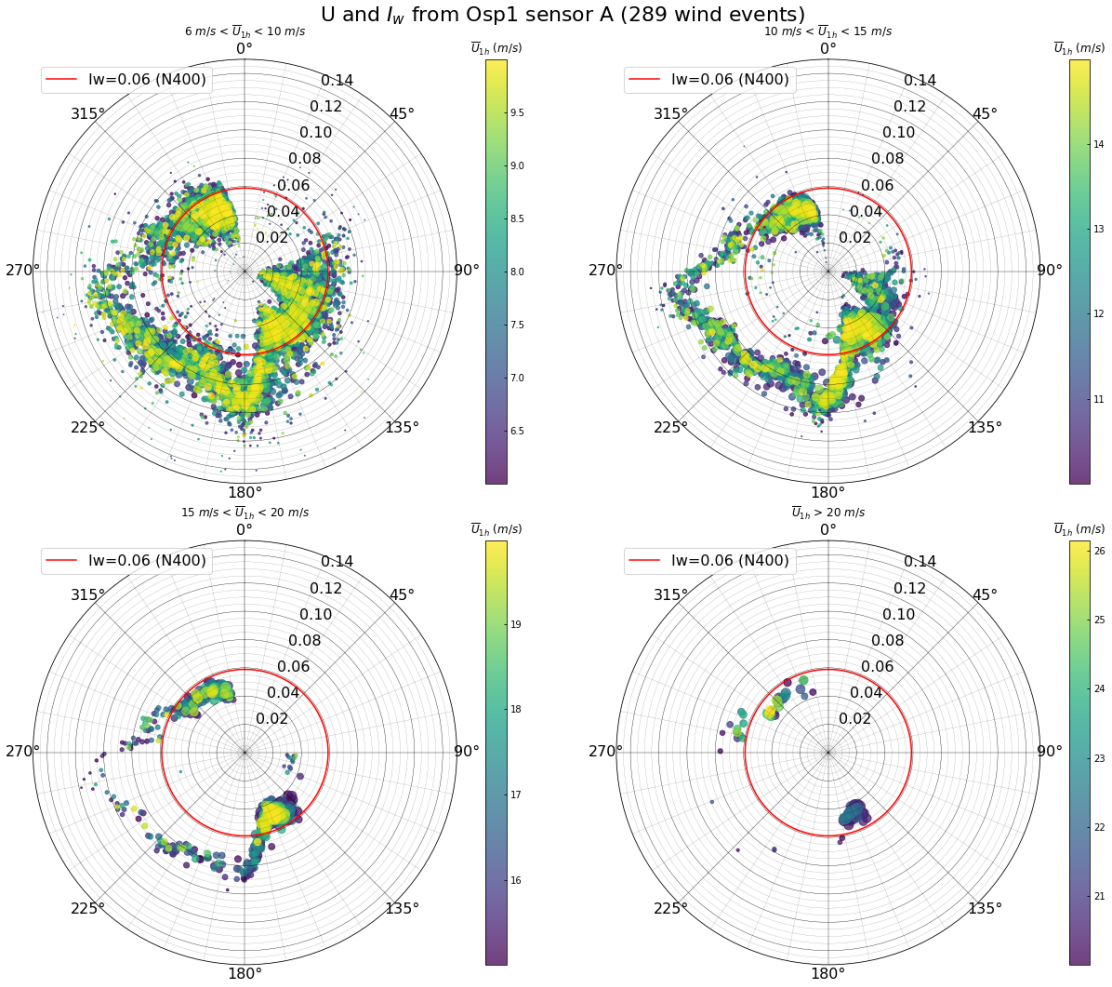

Figure 5.23: Distribution of the vertical turbulence intensity with mean wind speed at Osp1.

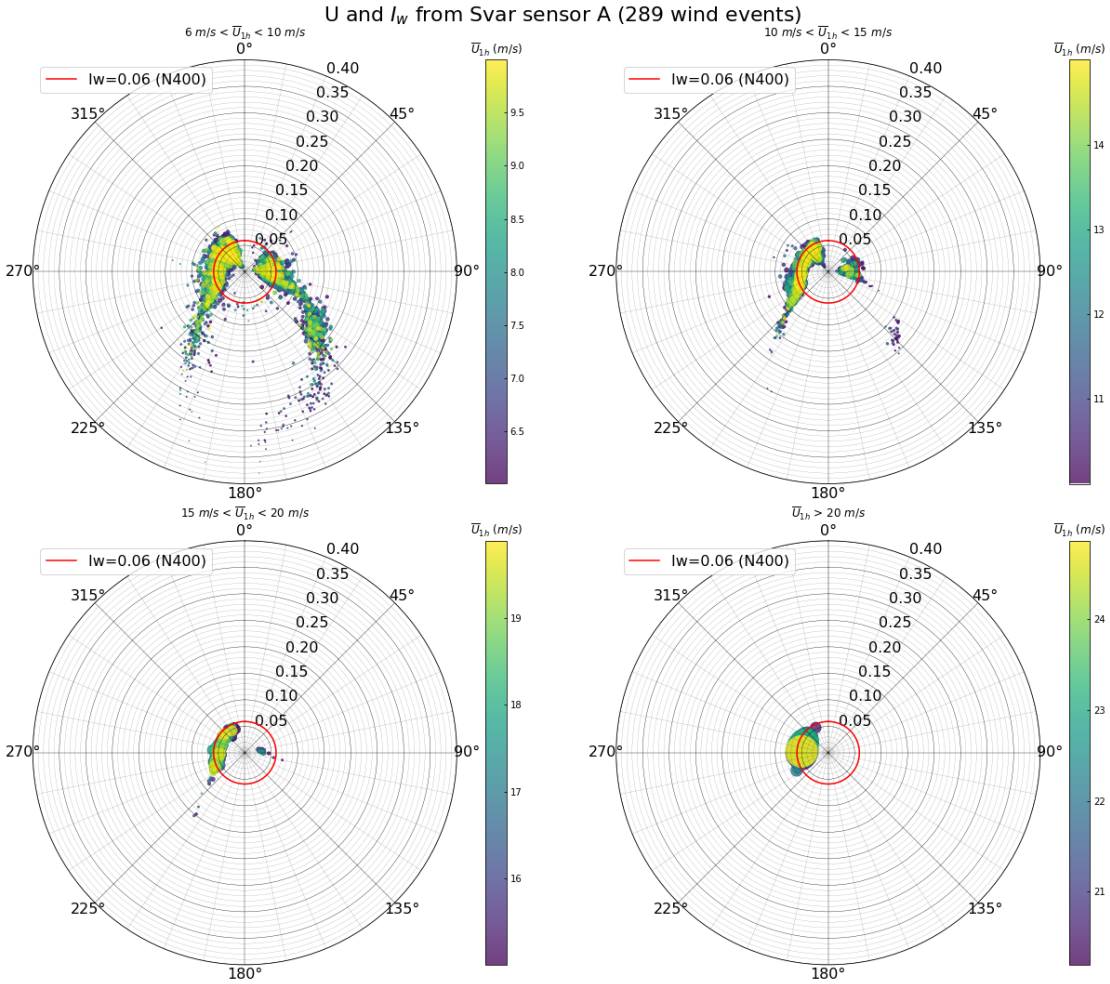

Figure 5.24: Distribution of the vertical turbulence intensity with mean wind speed at Svar.

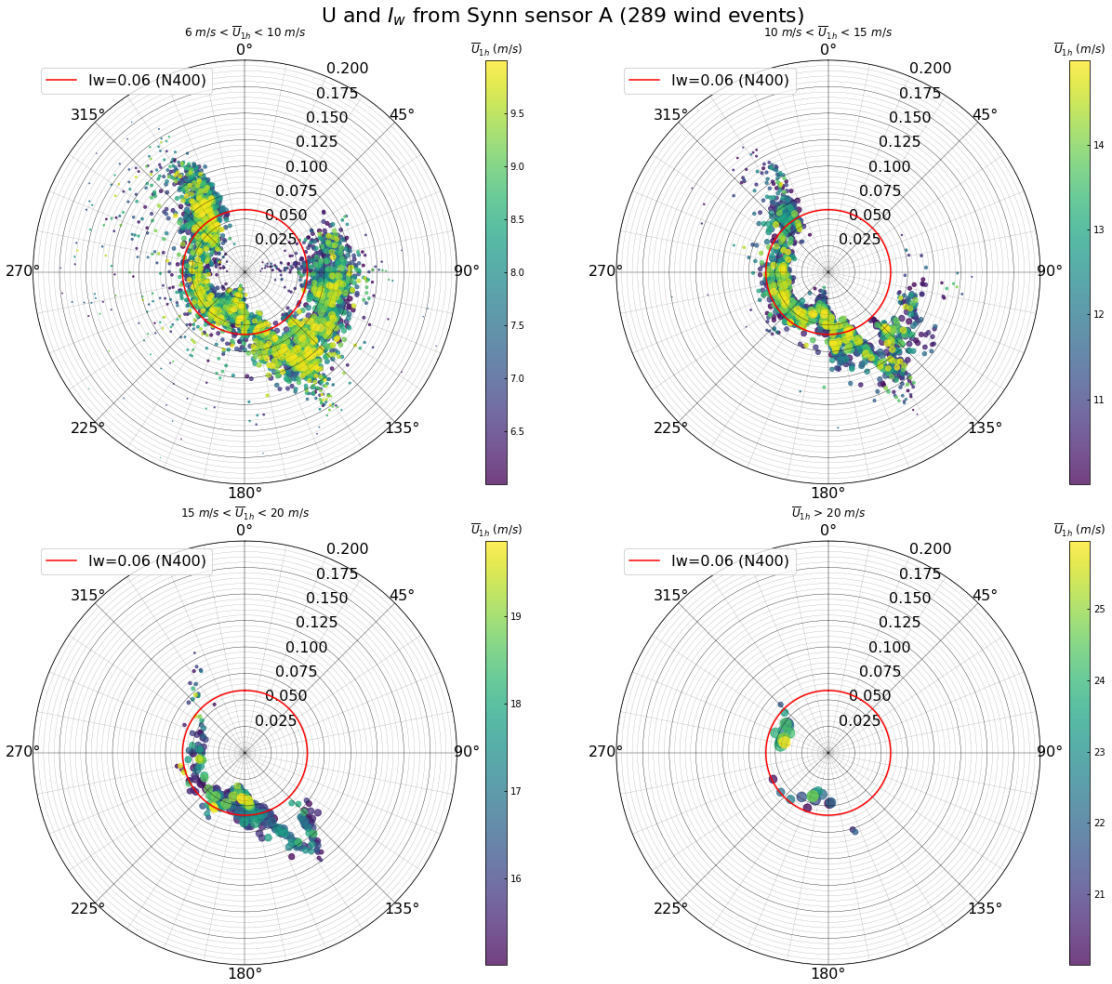

Figure 5.25: Distribution of the vertical turbulence intensity with mean wind speed at Synn.

5.3 Atmospheric stability

As shown in Table 5.3, the greatest number of observations give neutral conditions, followed by unstable and lastly stable. Figure 5.26 shows the variation of stability with the mean wind speed and mean wind direction at Ospøya 1, Svarvhelleholmen and Synnøytangen. Once again, Ospøya 2 was omitted due to having similar results as Ospøya 1. From the figure, as the mean wind speed increases, the atmosphere is increasingly characterized by near-neutral conditions, same as was found in [28]. Unstable and stable conditions are almost equally represented at the lower wind speeds for Ospøya and Svarvhelleholmen. Synnøytangen shows the largest scatter, indicating significant fluctuations in the atmospheric stability in that location. This may be due to the wind mast being located at the crossing between strictly onshore and offshore conditions, leading to more variability. It should also be noted that the results obtained for the stability parameter using Equation 2.24 depend on the vertical turbulence component w, which has not been corrected for any misalignment due to the tilt of the sonic anemometers. Therefore, the

magnitude and direction of w may be slightly different than what is actually the case, affecting the values for the stability parameter.

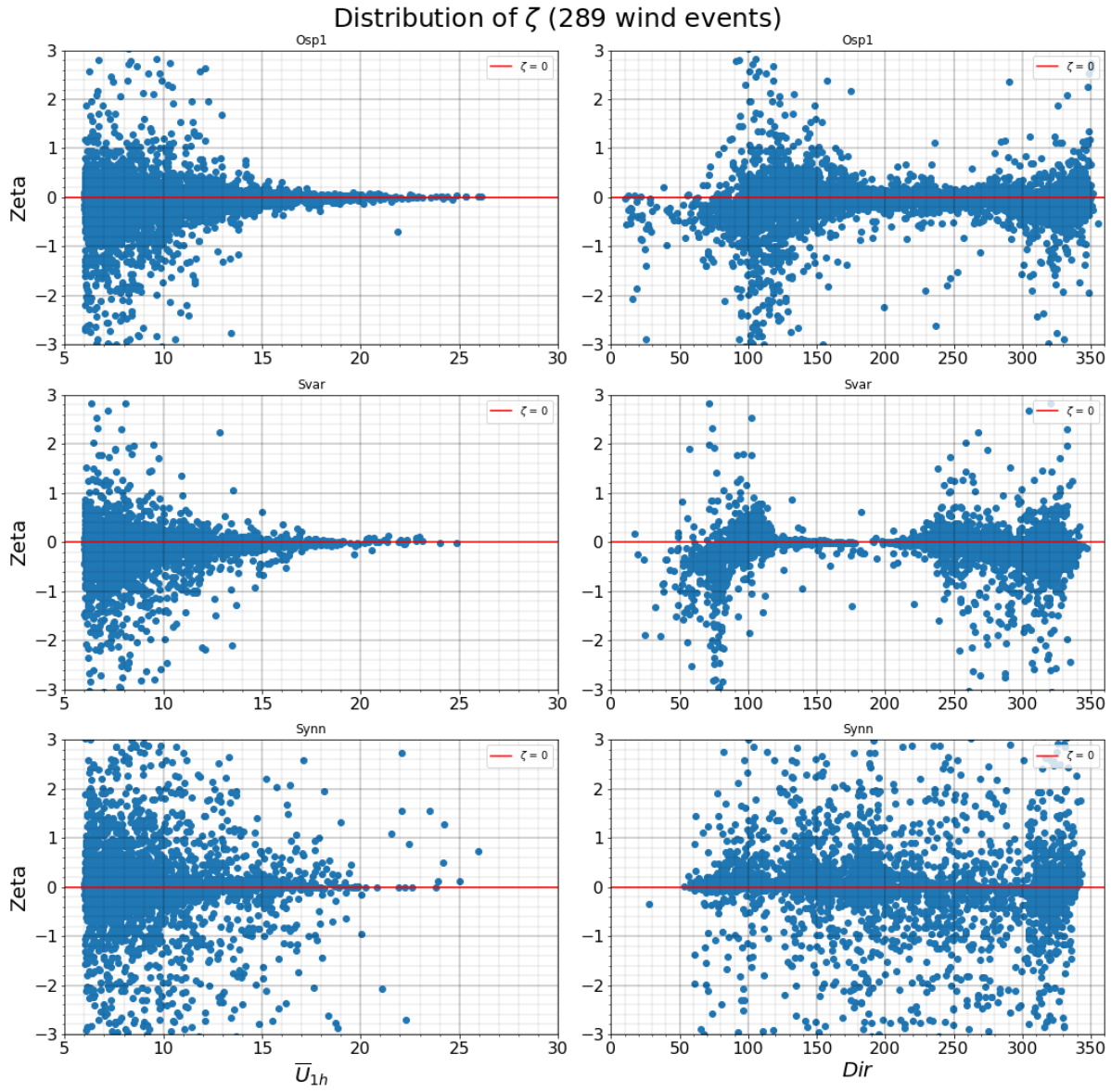

Figure 5.26: Variation of atmospheric stability with mean wind speed and direction at Osp1, Svar and Synn,

Figures 5.27-5.29 show the distribution of the atmospheric stability with different wind speed ranges. For the U3 and U4 ranges, the stability at Ospøya and Svarvhelleholmen tends to be at near-neutral conditions. For directions corresponding to wind flow over water, unstable conditions at Ospøya and Svarvhelleholmen are slightly more prevalent for U1 and U2, similar to what was found in [14]. It is difficult to see any trends for Synnøytangen, but as the mean wind speed increases, the observations are mostly concentrated in the south-east and north-

west. The results for the distribution of the atmospheric stability may be significantly affected by the amount of disregarded data, indicating that the use of non-stationary analysis for more representative results may be necessary.

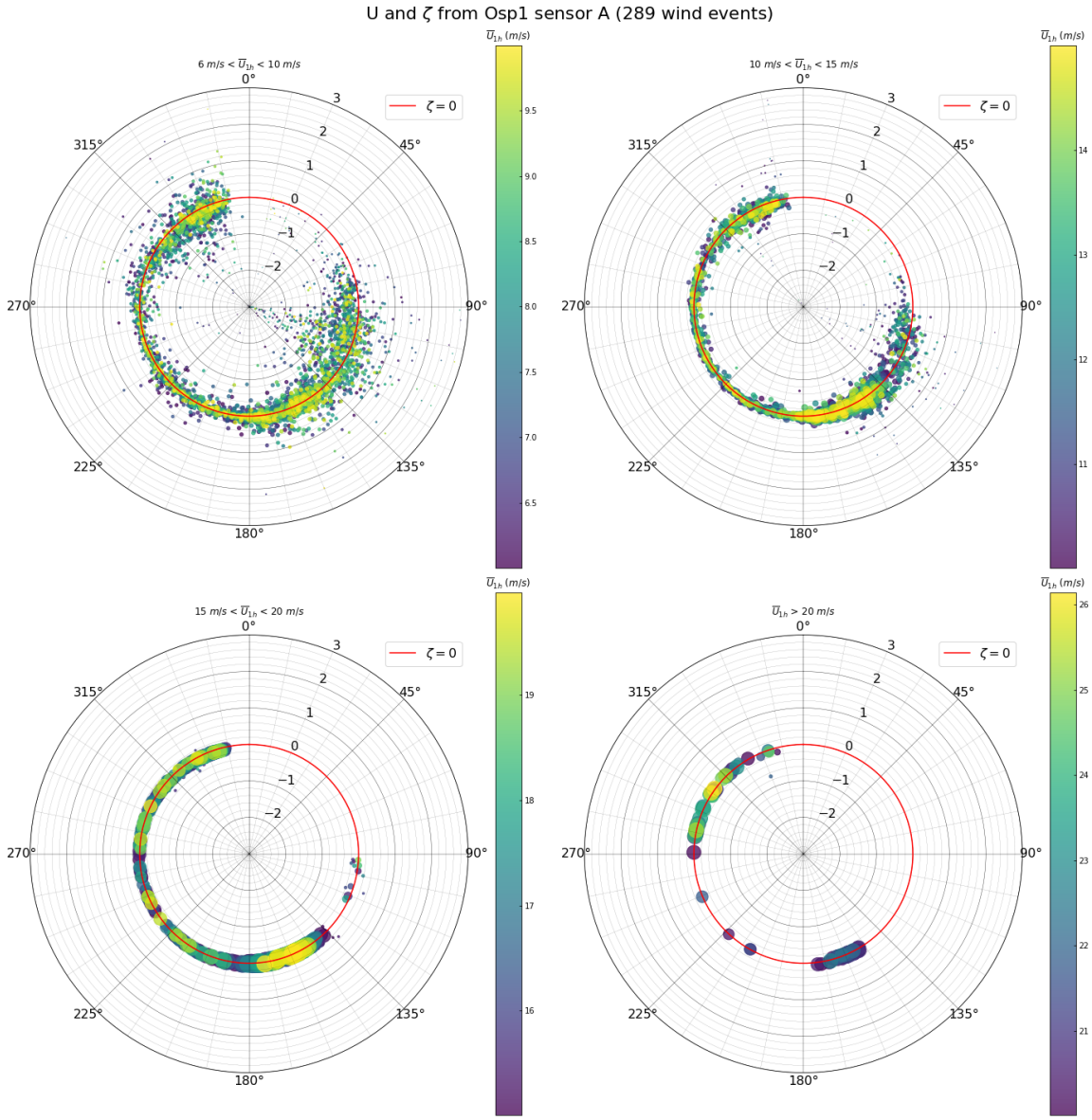

Figure 5.27: Distribution of atmospheric stability with mean wind speed at Osp1.

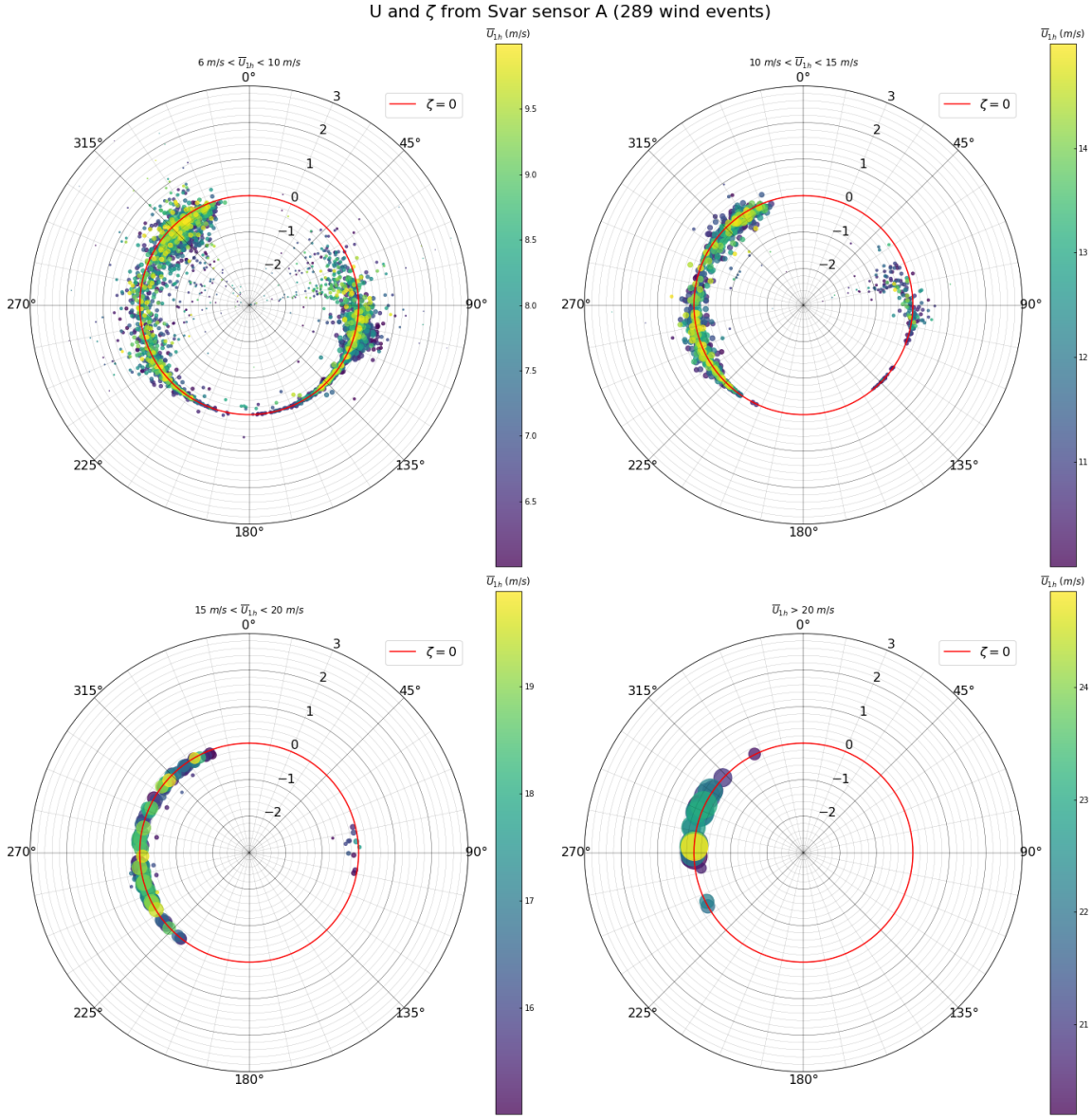

Figure 5.28: Distribution of atmospheric stability with mean wind speed at Svar.

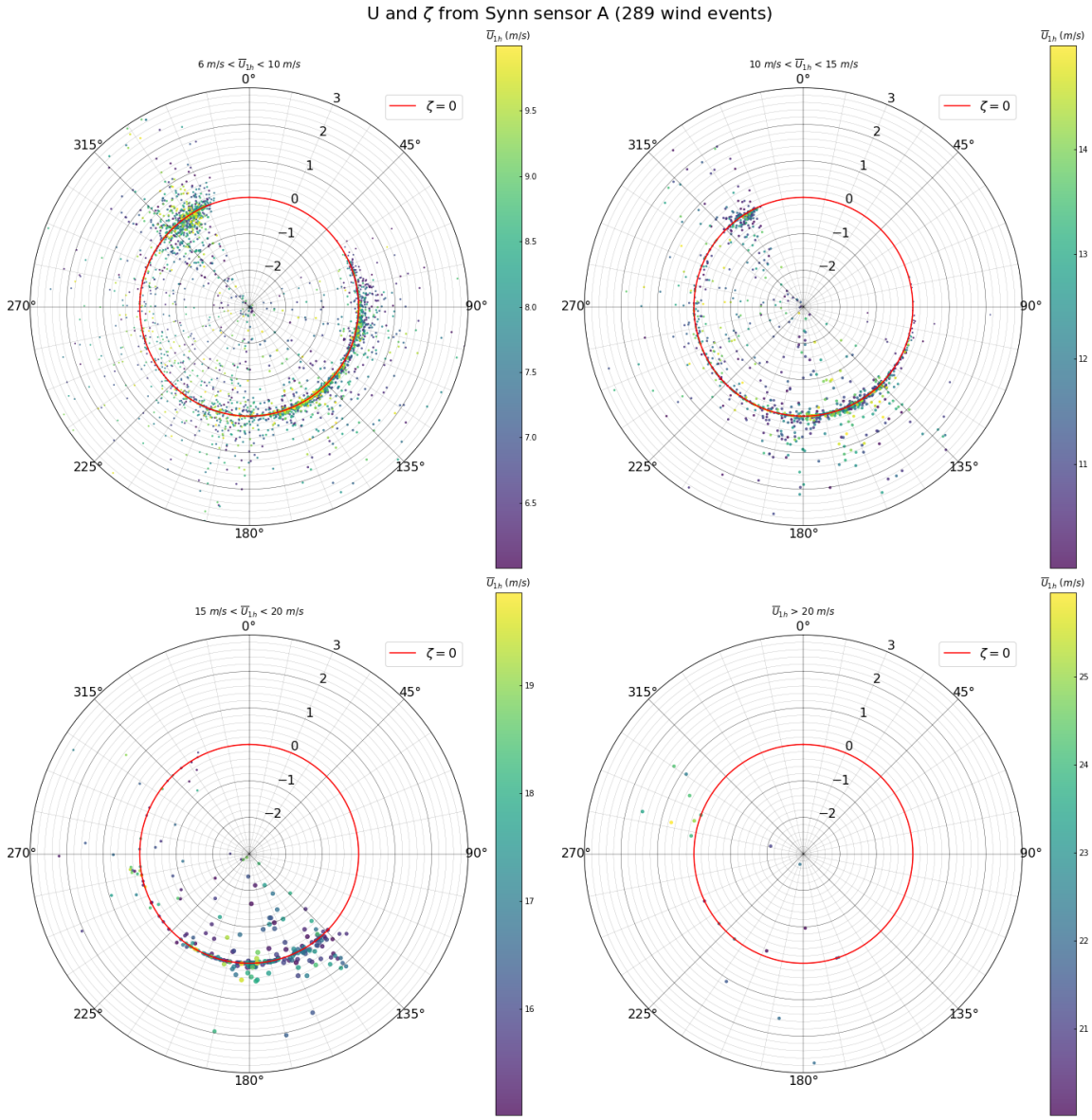

Figure 5.29: Distribution of atmospheric stability with mean wind speed at Synn.

5.4 Friction velocity

Figure 5.30 shows the variation of the standard deviation of the longitudinal turbulence with the friction velocity estimated using Equation 2.18. A reference value of the turbulence ratio β was chosen to be 2.5, which corresponds to neutral conditions over open, flat terrain. [9] As the wind speed range increases, shown by the different colored markers, the friction velocity tends to become higher compared to the smaller wind speed ranges, which show significantly more scatter. However, this may also be due to there being more observations for the lower ranges. In the U3 and U4 ranges, β is consistently higher than the reference value of 2.5.

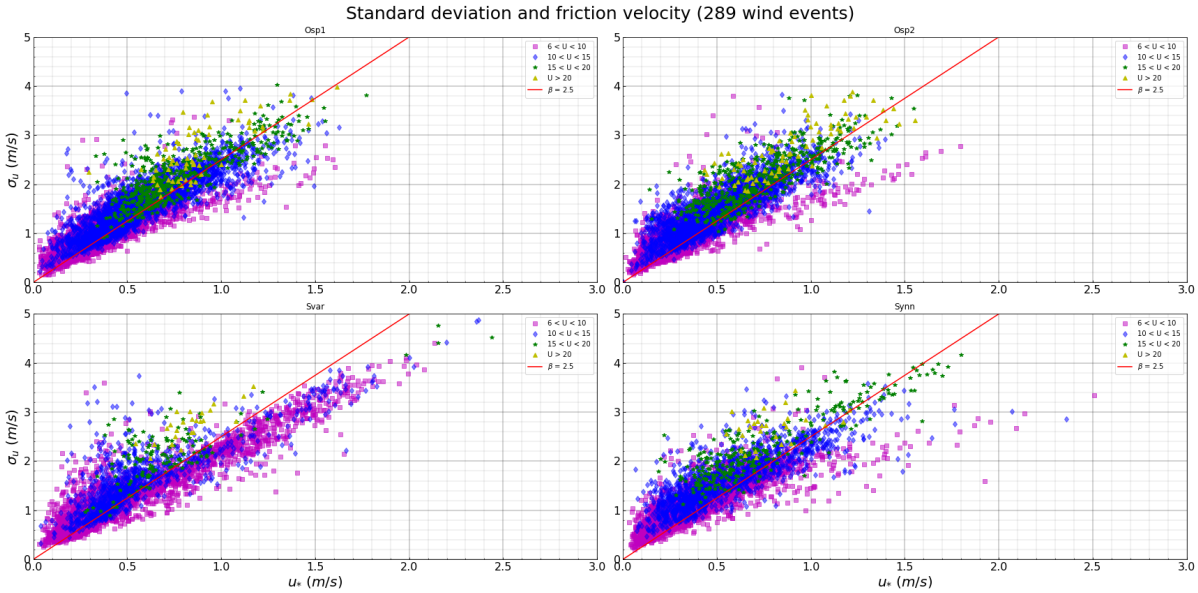

Figure 5.30: Variation of friction velocity in relation to the longitudinal standard deviation.

Figure 5.31 and Figure 5.32 show the distribution of β from the four top sensors. According to Equation 2.19, β increases when u_* decreases, indicating that the lower values for the friction velocity occur from the east at Ospøya 1 and 2, from the east and west at Svarvhelleholmen and from the west at Synnøytangen. All these directions correspond to wind flow over water. General values for β lie between 2 and 4, with some higher values from the south-east at Ospøya 2. These values agree with results in [8], as β is expected to increase with rougher terrain. The large scatter of β could be due to changes in nearby roughness length, as mentioned in [12].

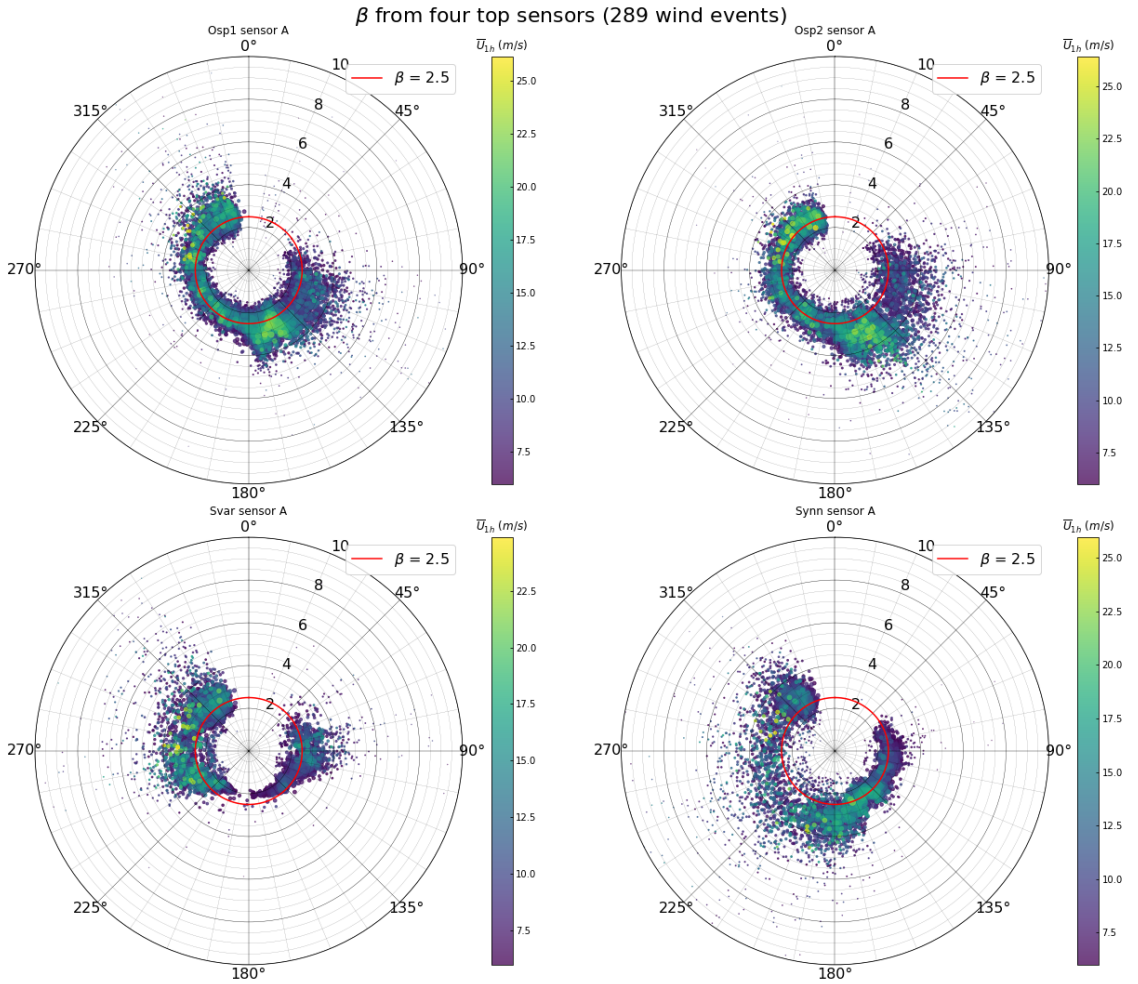

Figure 5.31: Distribution of the turbulence ratio with mean wind speed from the four top sensors.

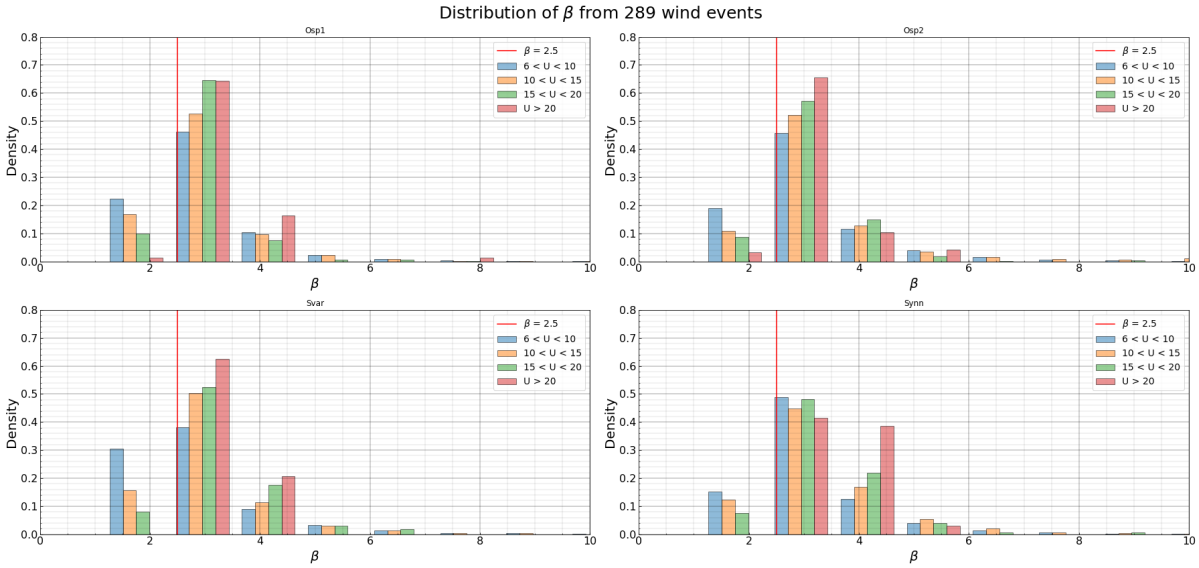

Figure 5.32: Distribution of the turbulence ratio from the four top sensors.

Figure 5.33 shows the variation of β with stability at Osp1. The other wind masts are not shown as they show similar results. β is found to decrease gradually from unstable to neutral conditions, and increase from neutral to stable conditions. [9, 14] Although the scatter of β is large, the same trend can be observed in the figure below. In unstable wind flows there is more turbulent mixing, and β is found to be higher in these conditions. [9]

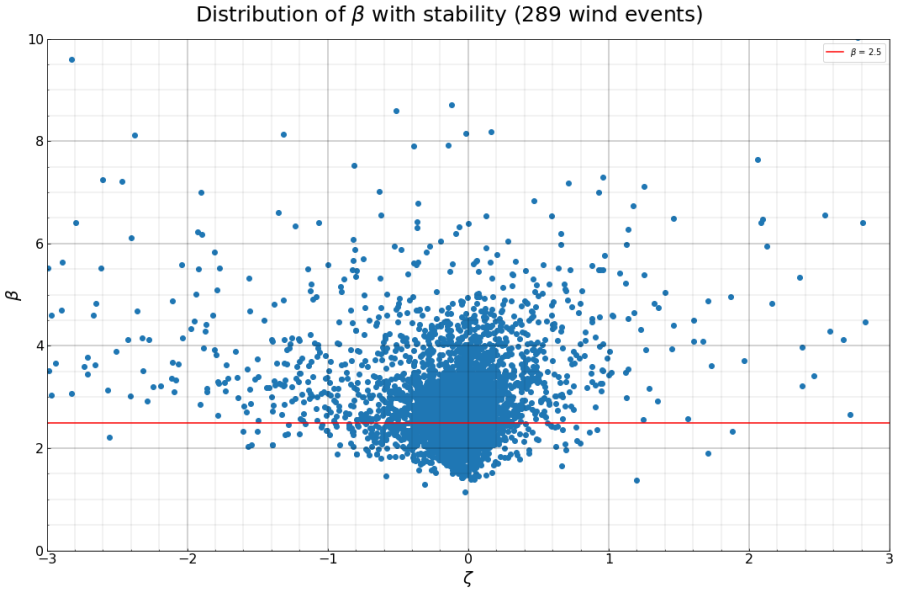

Figure 5.33:Variation of the turbulence ratio with stability at Osp1.

5.5 Length scales

Figure 5.34 and Figure 5.35 show the variation of the integral length scales L_i^x with mean wind speed, based on 10-minute wind events. When comparing the wind masts at Ospøya, Ospøya 1 shows much larger scatter compared to Ospøya 2. This scatter is also present at Svarvhelleholmen and Synnøytangen. The general trend for Ospøya 2 is that as the mean wind speed increases, the range of the scatter for L_i^x increases slightly; this increase of scatter range is highest for the longitudinal component and lowest for the vertical component. Conversely, the opposite is the case for the three remaining wind masts, which show a decrease of L_i^x as the mean wind speed increases. At the height at which sensor A is installed, N400 gives the suggested values for L_u^x, L_v^x, and L_w^x as 160.1, 40.2, and 13.4 respectively. These values underestimate the integral length scales, due to the large variability when estimating L_i^x. As Fenerci and Øiseth (2018) concluded, the use of the estimated L_i^x as design parameters should be avoided when possible in complex terrain and non-stationary wind. [29]

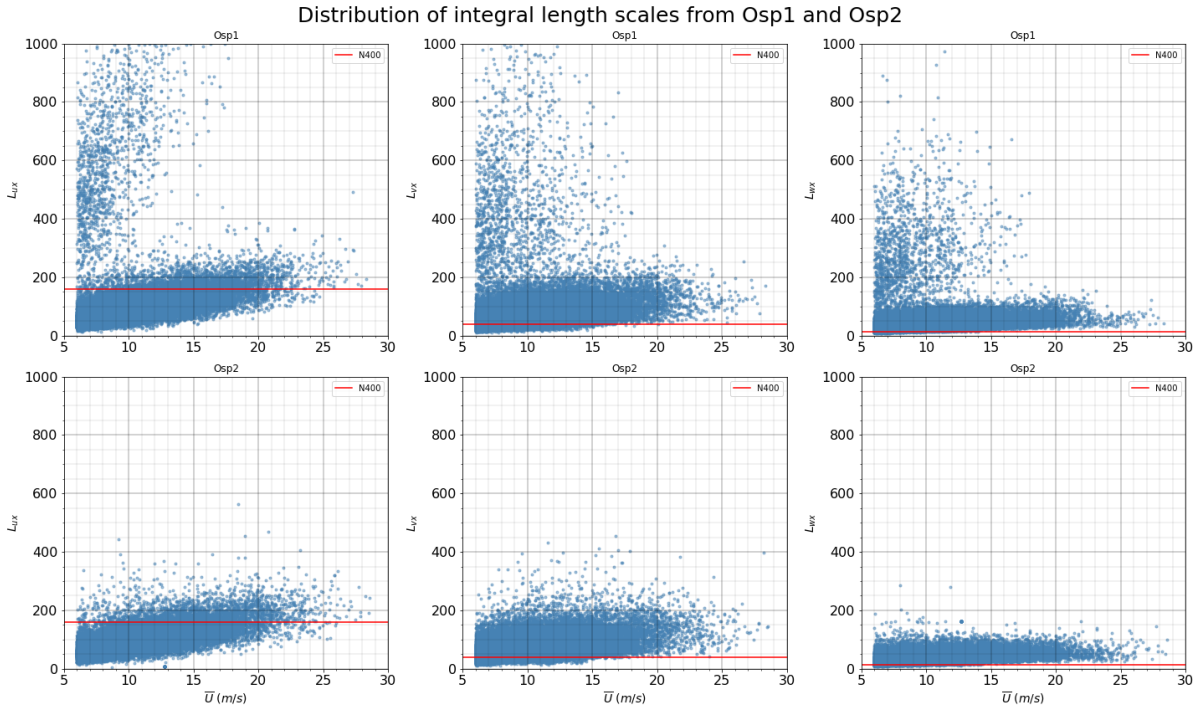

Figure 5.34: Comparison of length scales at Ospøya 1 and 2.

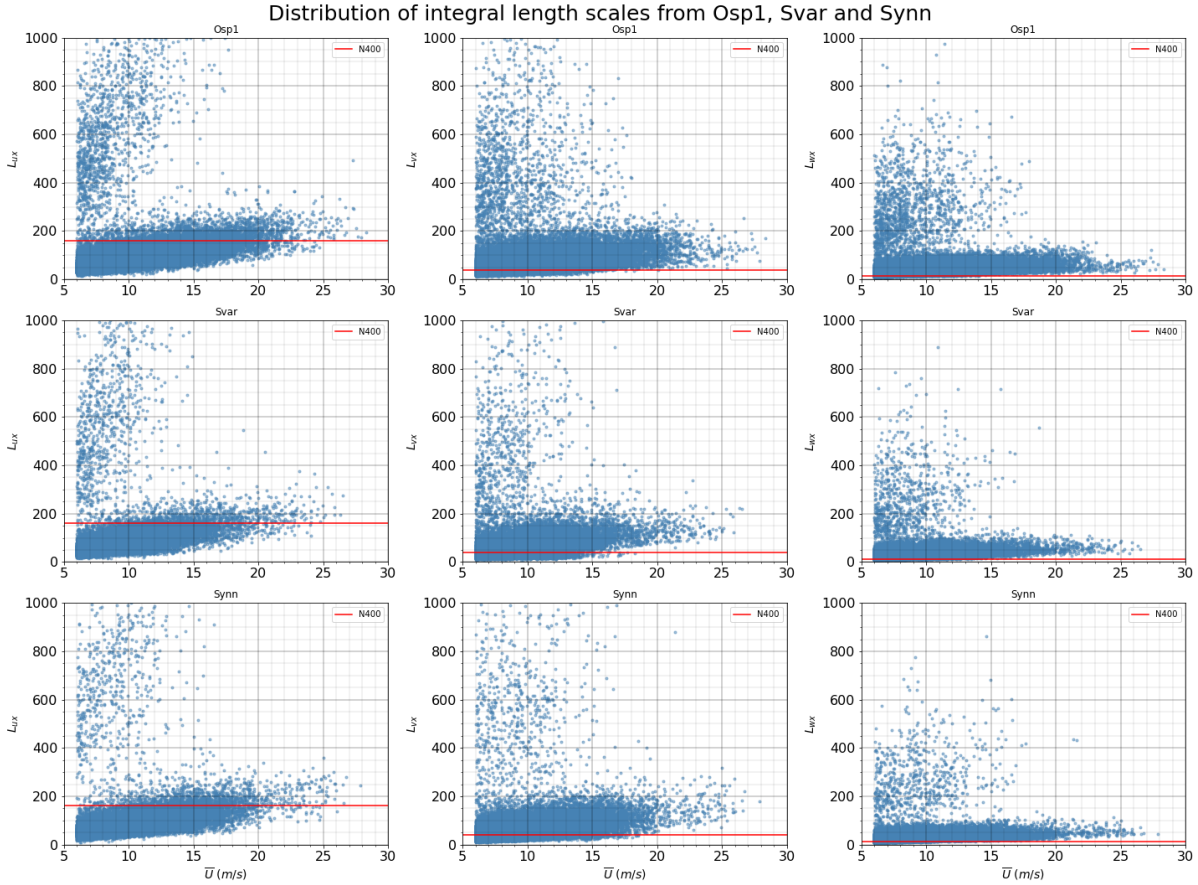

Figure 5.35: Variation of length scales with wind speed at Osp1, Svar and Synn.

5.6 Spectral models

A comparison between the non-normalized fitted spectral models are presented in Figures 5.36-5.39 for the same 1-hour event, from 09:00 to 10:00 on October 23, 2018. The black vertical lines represent the frequency range in which the fitting was performed, while the shaded grey area covers the eigen-frequencies of the floating bridge. The estimated power spectral densities based on the wind records from the fjord are represented by the black markers. It should be noted that the spectrum fitting for the N400 using 2 parameters, shown in Equation 4.2, coincides exactly with the original Kaimal spectrum for the longitudinal and lateral turbulence components, which is why it is not visible on the figures below. In addition, the selected 1-hour event is not representative for all cases; observations made below may not apply to a randomly selected 1-hour event or wind event, however, it may still give some insight into the behavior of the spectra in relation to each other.

As expected, the spectral models represent the measured spectral densities quite well in the frequency range from 0.1 and higher. In general, the Frøya spectrum seem to capture most of the energy content in the low frequency range for the longitudinal component, except for the 1-hour event at Ospøya 1. The fitted Harris spectrum seems to closely follow the shape of the longitudinal Kaimal spectrum in the fitted frequency range, deviating only around frequencies smaller than 0.01 Hz. In the frequency range of the bridge's eigen-frequencies, the Harris spectrum represents the energy fairly well. The 1-parameter N400 and Kaimal spectra are very similar in the bridge's eigen-frequency range and tend to deviate near the lower limit of fitted frequency range. In general, the Kaimal spectra fit the data well, and this may be because the atmospheric stability most of the time is near-neutral. The spectral models for the vertical turbulence component show significant variation in the energy content compared to the longitudinal and lateral turbulence component. The reference Busch-Panofsky seems to slightly overestimate the energy content at frequencies higher than 0.1 Hz.

Event No.91 Dir=315.0^o Osp1 at height 48.8m from 2018-10-23 09:00 to 2018-10-23 10:00

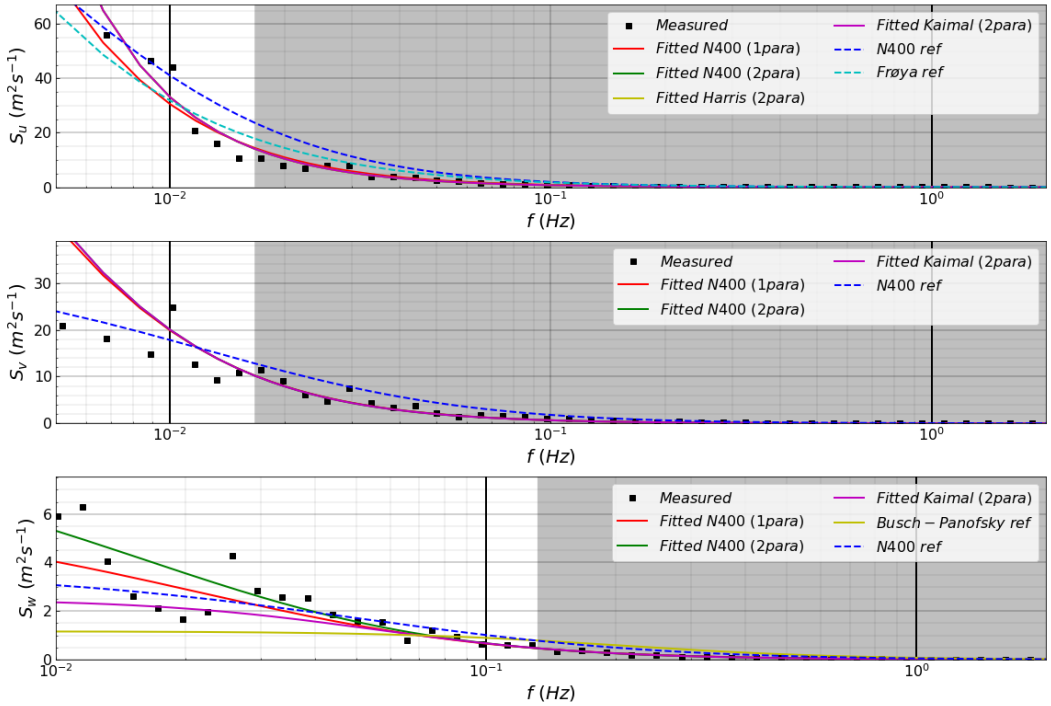

Figure 5.36: Comparison of spectral models on October 23, 2018, at Osp1.

Event No.91 Dir=309.7^o Osp2 at height 48.8m from 2018-10-23 09:00 to 2018-10-23 10:00

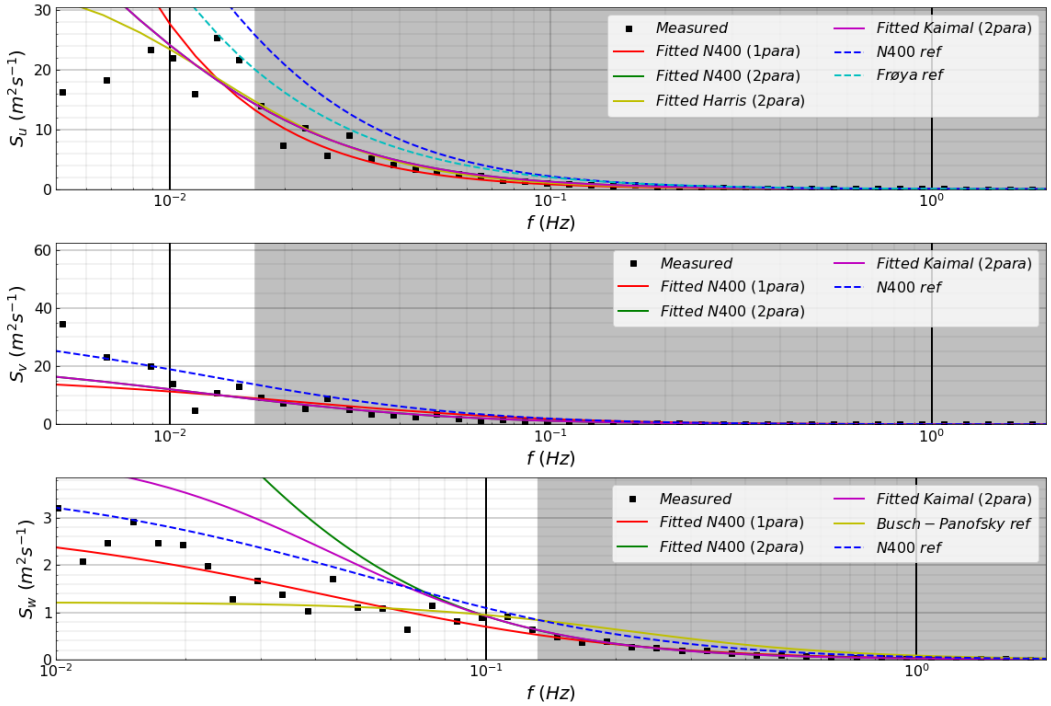

Figure 5.37: Comparison of spectral models on October 23, 2018, at Osp2.

Event No.91 Dir=302.2^o Svar at height 48.3m from 2018-10-23 09:00 to 2018-10-23 10:00

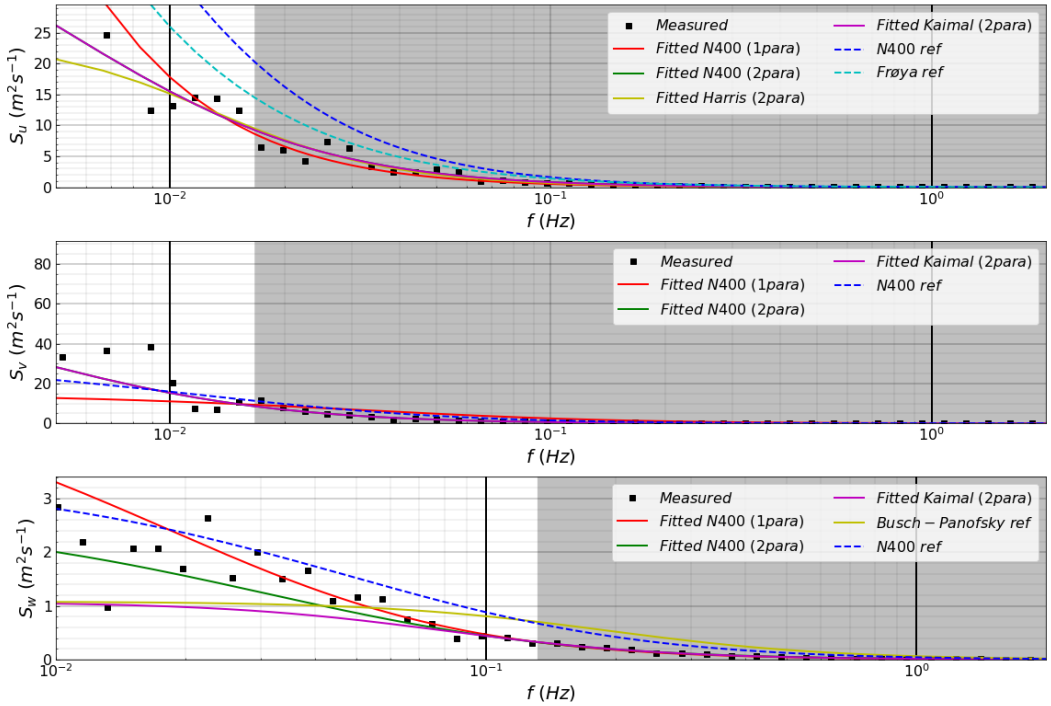

Figure 5.38: Comparison of spectral models on October 23, 2018, at Svar.

Event No.91 Dir=308.9^o Synn at height 48.3m from 2018-10-23 09:00 to 2018-10-23 10:00

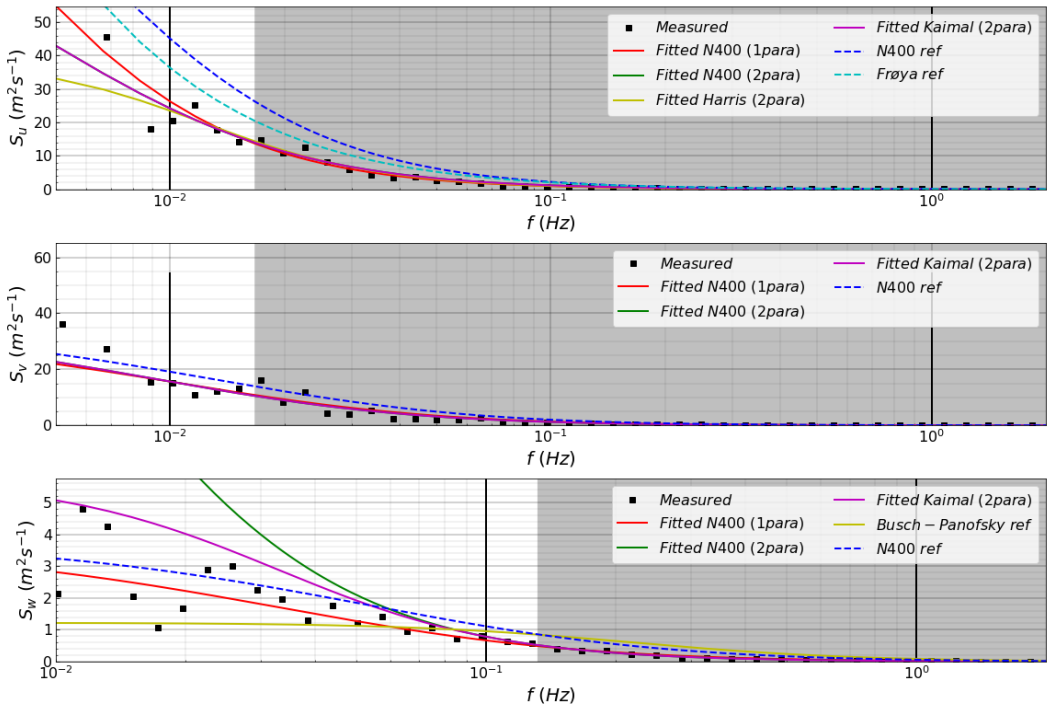

Figure 5.39: Comparison of spectral models on October 23, 2018, at Synn.

5.6.1 Fitted N400 spectrum

Figures 5.40-5.44 show the non-normalized and normalized fitted N400 spectra with the estimated spectral density. As mentioned earlier, the fitted spectral models represent the energy content well in the high frequency range. In fact, for the 1-hour event presented in Figures 5.40-5.43, there is very little difference between the 1-parameter and the 2-parameter models in the frequency range corresponding to the bridge's eigen-frequencies. For the lateral turbulence component, the 1-parameter and the 2-parameter models are almost identical, with the 1-parameter model tending to underestimate the energy content in the low frequency range compared to the 2-parameter model. Frequencies less than 0.1 Hz show varying results. The reference N400 spectrum seems to consistently capture all the energy content regardless of frequency range with a few exceptions (mainly the lateral component), but also tends to overestimate for frequencies less than 0.1 Hz. As mentioned earlier, a single 1-hour event is not representative for all cases. Figure 5.44 shows a 1-hour event at Svarvhelleholmen on January 3, 2020, from 15:00 to 16:00, where the 2-parameter N400 model gives a better fitting for the longitudinal and lateral turbulence compared to the 1-parameter model, and vice versa for the vertical turbulence.

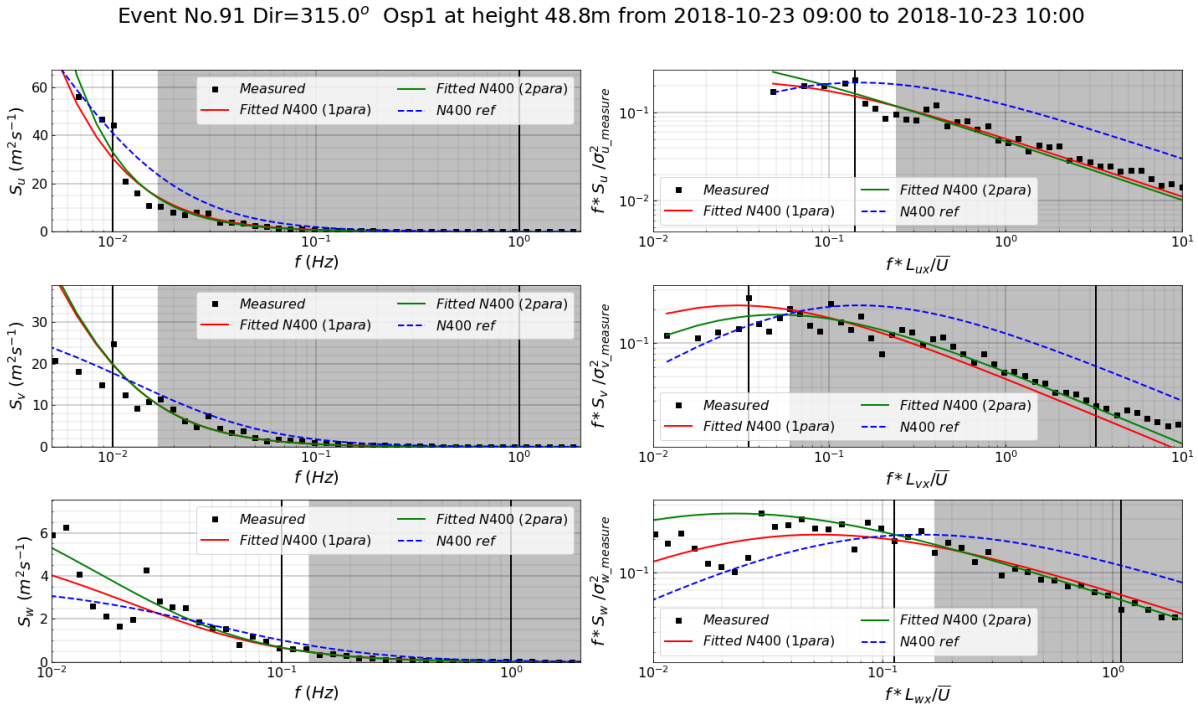

Figure 5.40: Comparison of fitted N400 spectra on October 23, 2018, at Osp1.

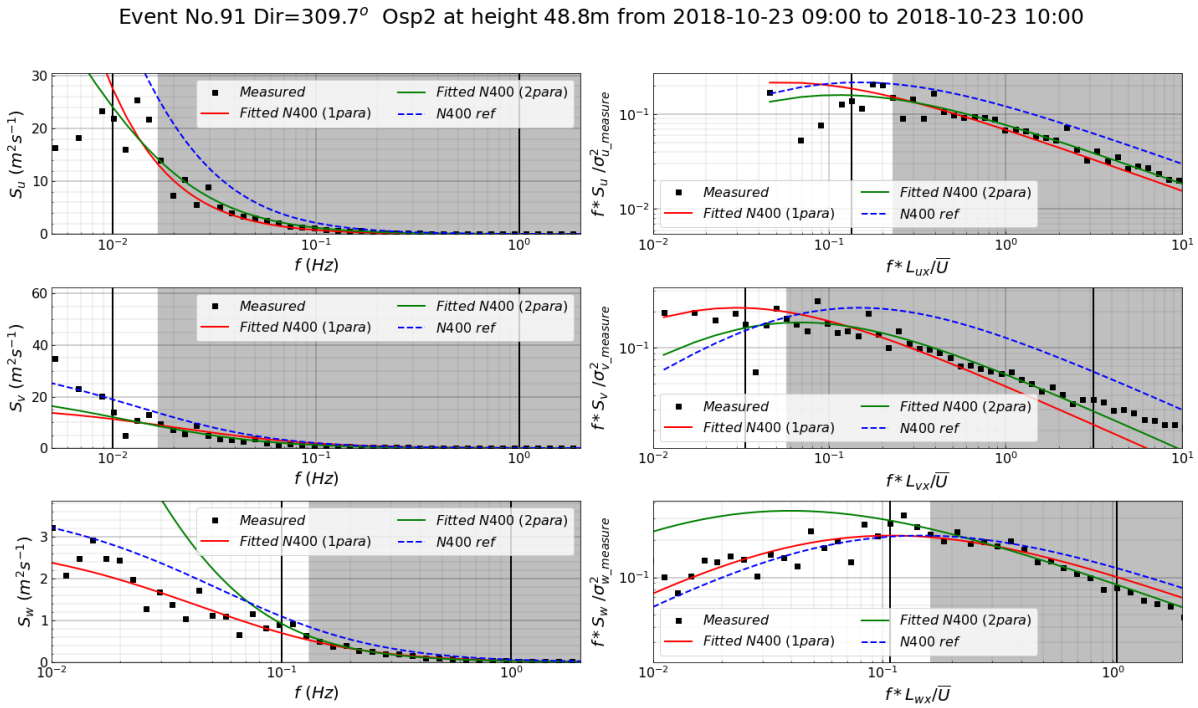

Figure 5.41: Comparison of fitted N400 spectra on October 23, 2018, at Osp2.

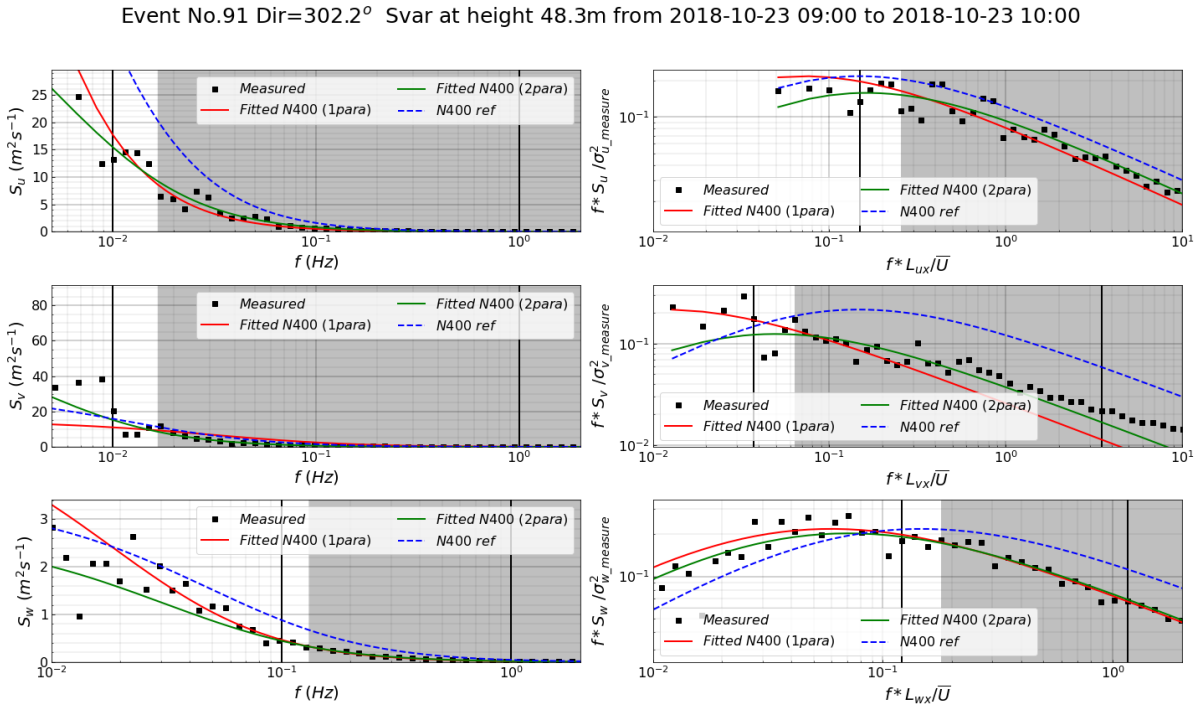

Figure 5.42: Comparison of fitted N400 spectra on October 23, 2018, at Svar.

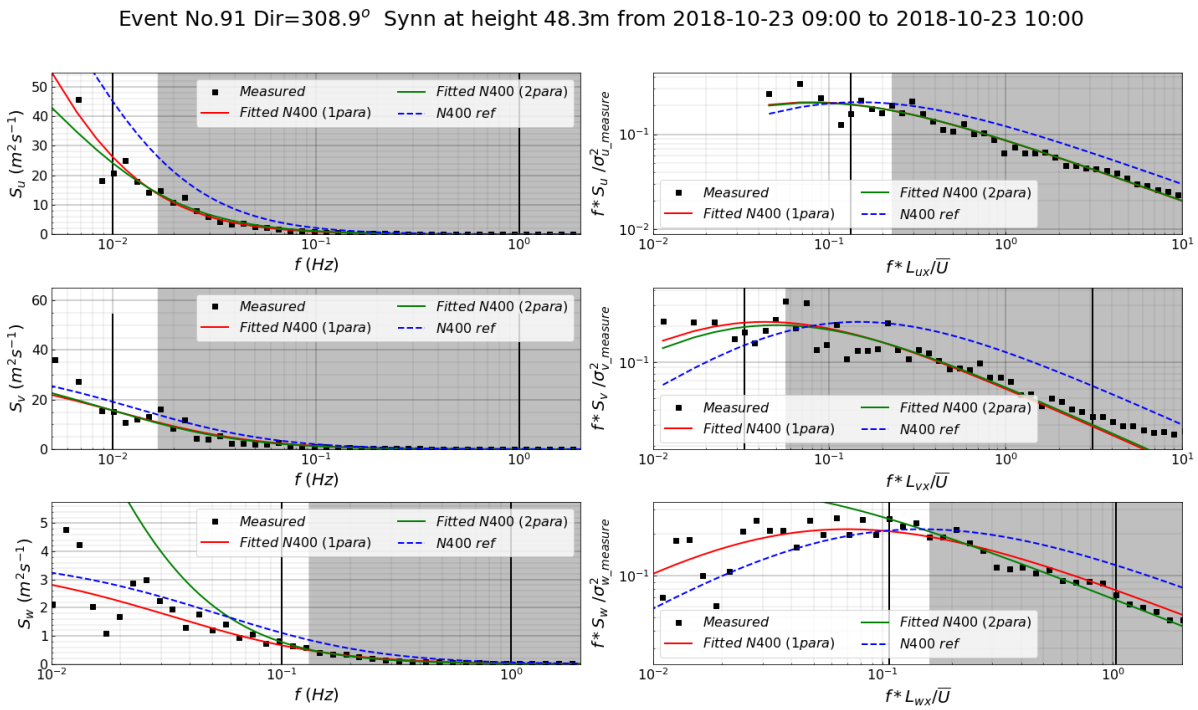

Figure 5.43: Comparison of fitted N400 spectra on October 23, 2018, at Synn.

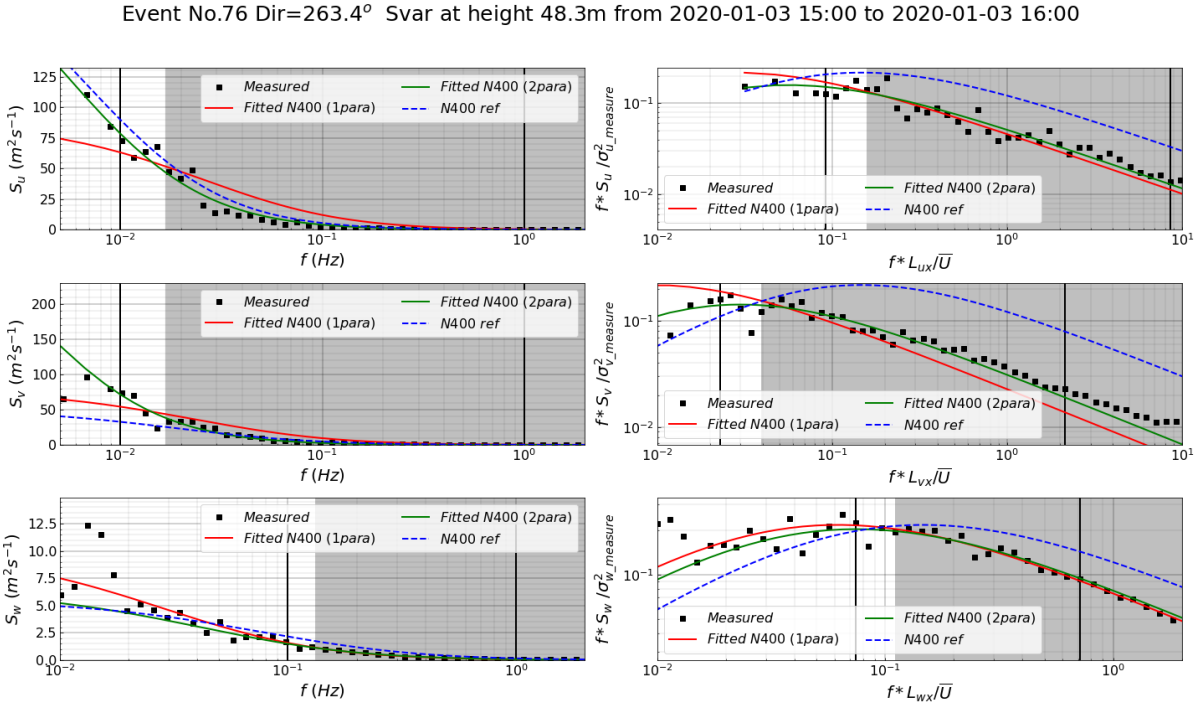

Figure 5.44: Comparison of fitted N400 spectra on January 3, 2020, at Svar.

5.6.1.1 Distribution of the fitted parameters

The distribution of the fitted parameters in the 1-parameter N400 model with mean wind speed and mean wind direction are shown in Figures 5.45-5.53. Results from Ospøya 2 are omitted from this section as they are similar to Ospøya 1.

The fitted A_u has the highest directional densities from south-east and north-west at Ospøya, from west, north-west and east at Svarvhelleholmen, and from south-west and north-west at Synnøytangen. These directions correspond to wind flow over water, and the fitted A_u shows the largest scatter here. Midjiyawa et. al. (2021) studied wind characteristics in different Norwegian fjords and found that for a long upstream fetch of water, the spectral peak moved to the low frequency range [30], as is the case here as well. Additionally, Li et. al. (2012) found that the longitudinal spectral parameter increased with increasing turbulence ratio β [9]. Comparing Figure 5.31 to Figures 5.45-5.47, this relationship between β and A_u is also observed here for all wind masts. As the mean wind speed range increases, A_u tends to be closer to the reference value in N400, however, this may also be because the number of 1-hour events in the higher velocity ranges are smaller than in the previous range. For the ranges U3 and U4, the energy content in the low frequency range seems to be smaller than for the U1 and U2 ranges.

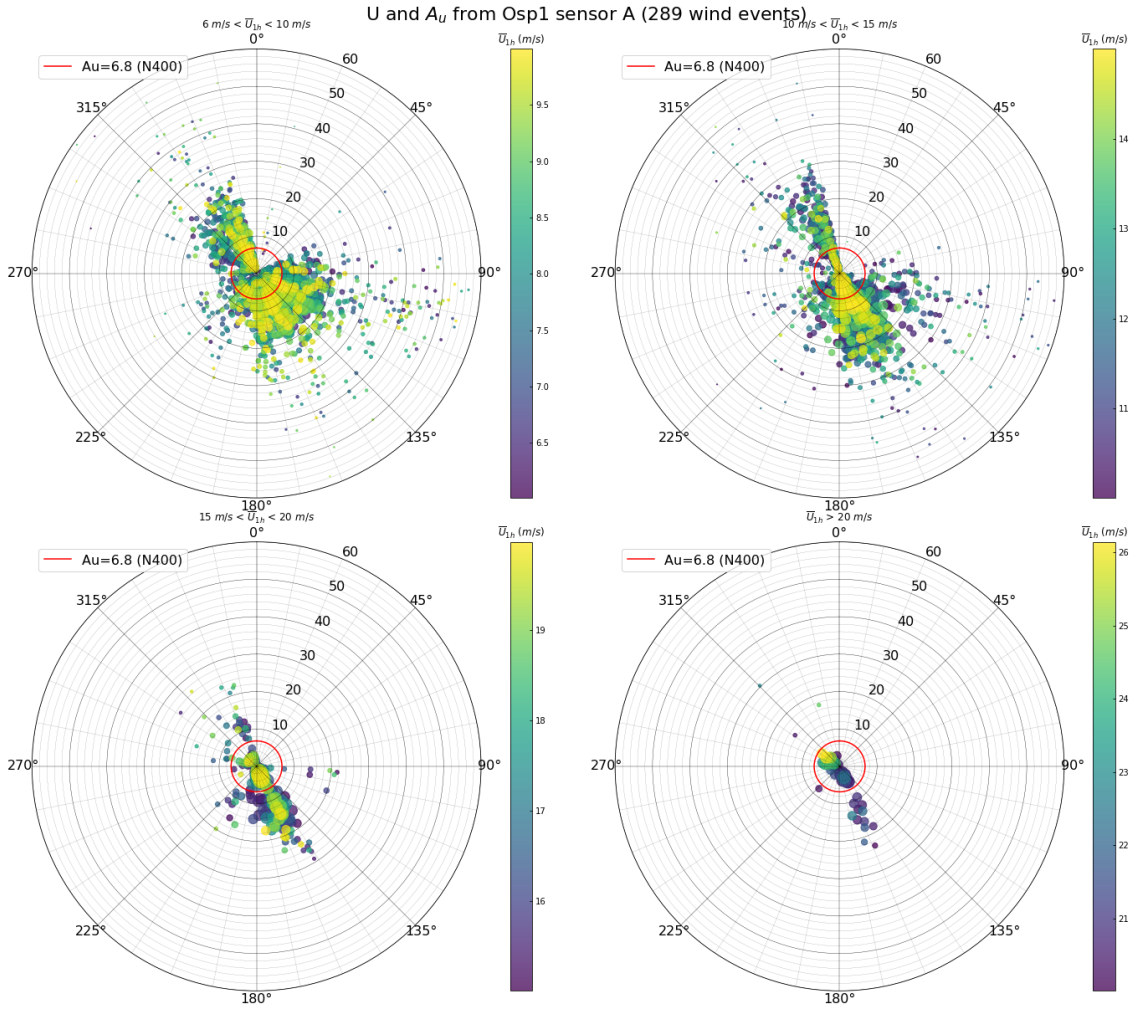

Figure 5.45: Distribution of the fitted A_u with mean wind speed at Osp1.

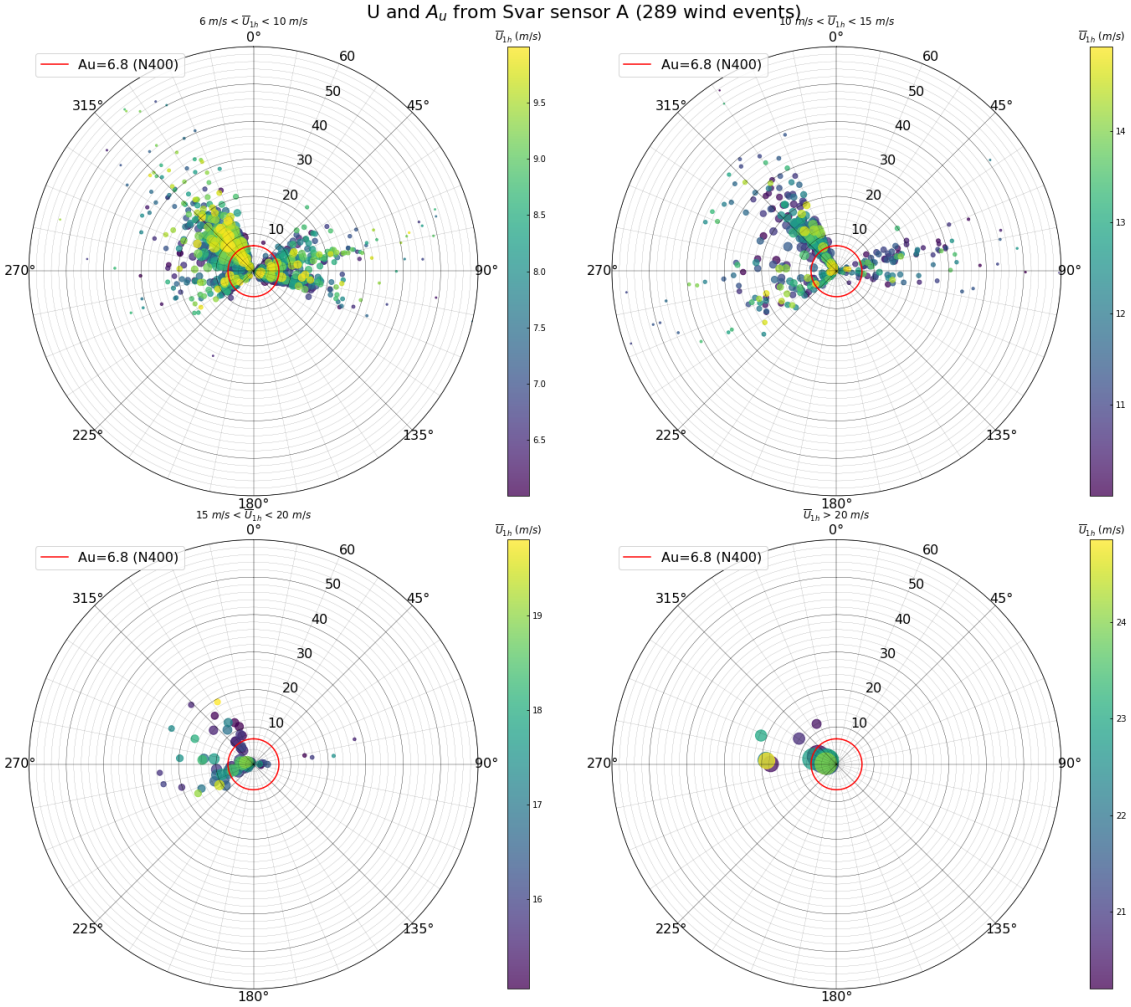

Figure 5.46: Distribution of the fitted A_u with mean wind speed at Svar.

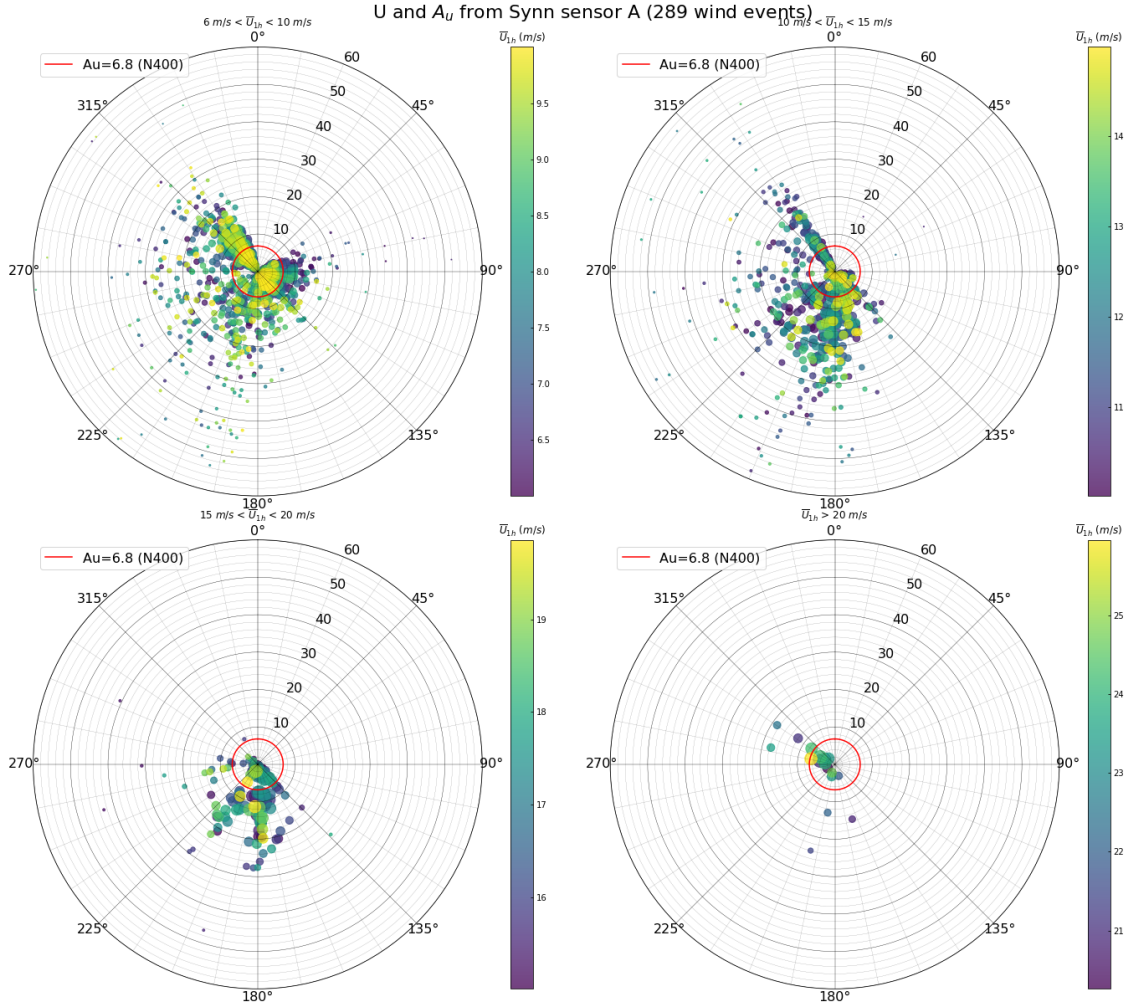

Figure 5.47: Distribution of the fitted A_u with mean wind speed at Synn.

Figures 5.48-5.50 show that the fitted parameter for the lateral turbulence component is generally higher compared to the longitudinal component, particularly when comparing the case at Ospøya for U4. The main directions are mostly the same as for A_u, but with more coming from the west at Svarvhelleholmen and from the south at Synnøytangen. Generally, the fitted A_v tends to be larger than the suggested value in N400 for all the wind speed ranges, indicating more energy in the low frequency region.

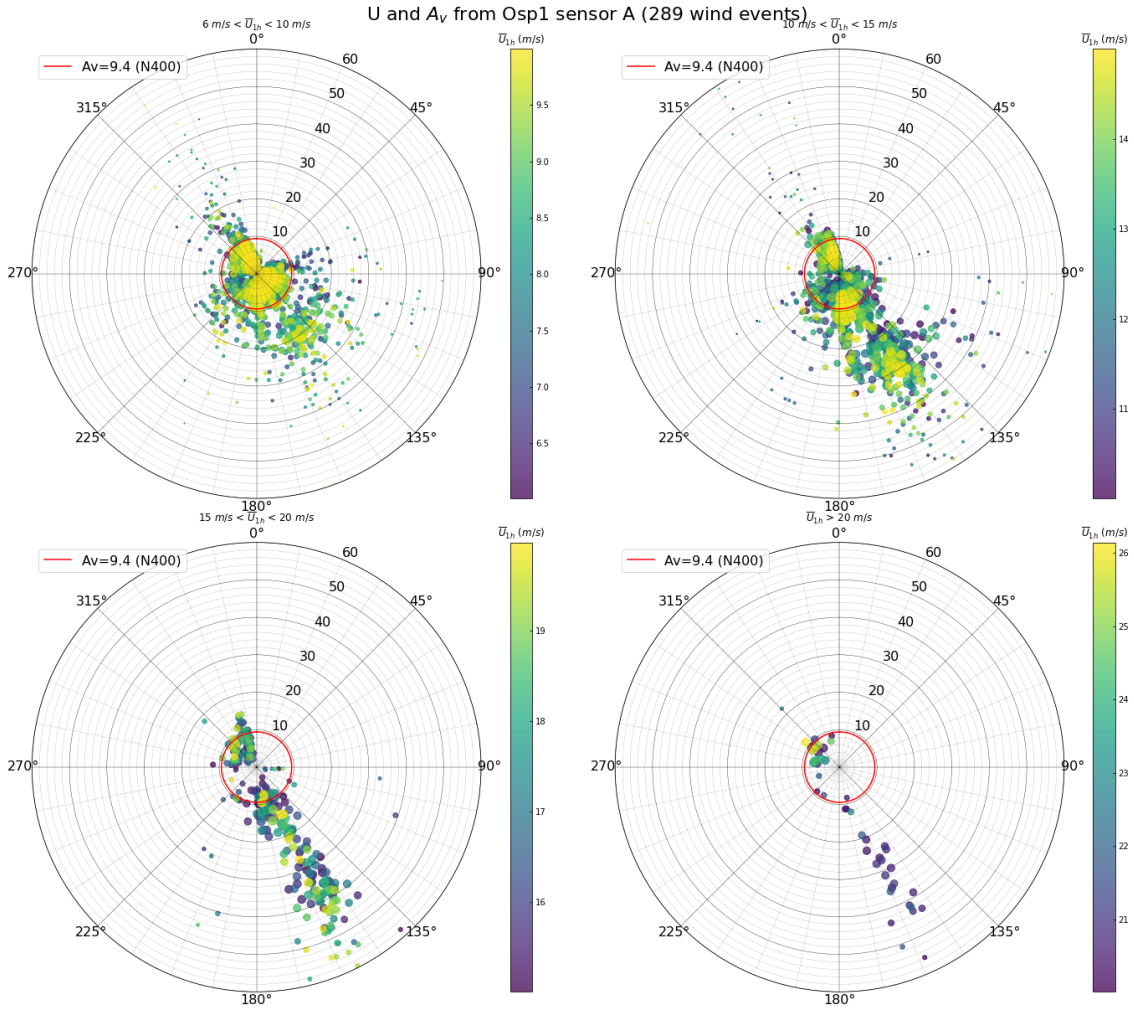

Figure 5.48: Distribution of the fitted A_v with mean wind speed at Osp1.

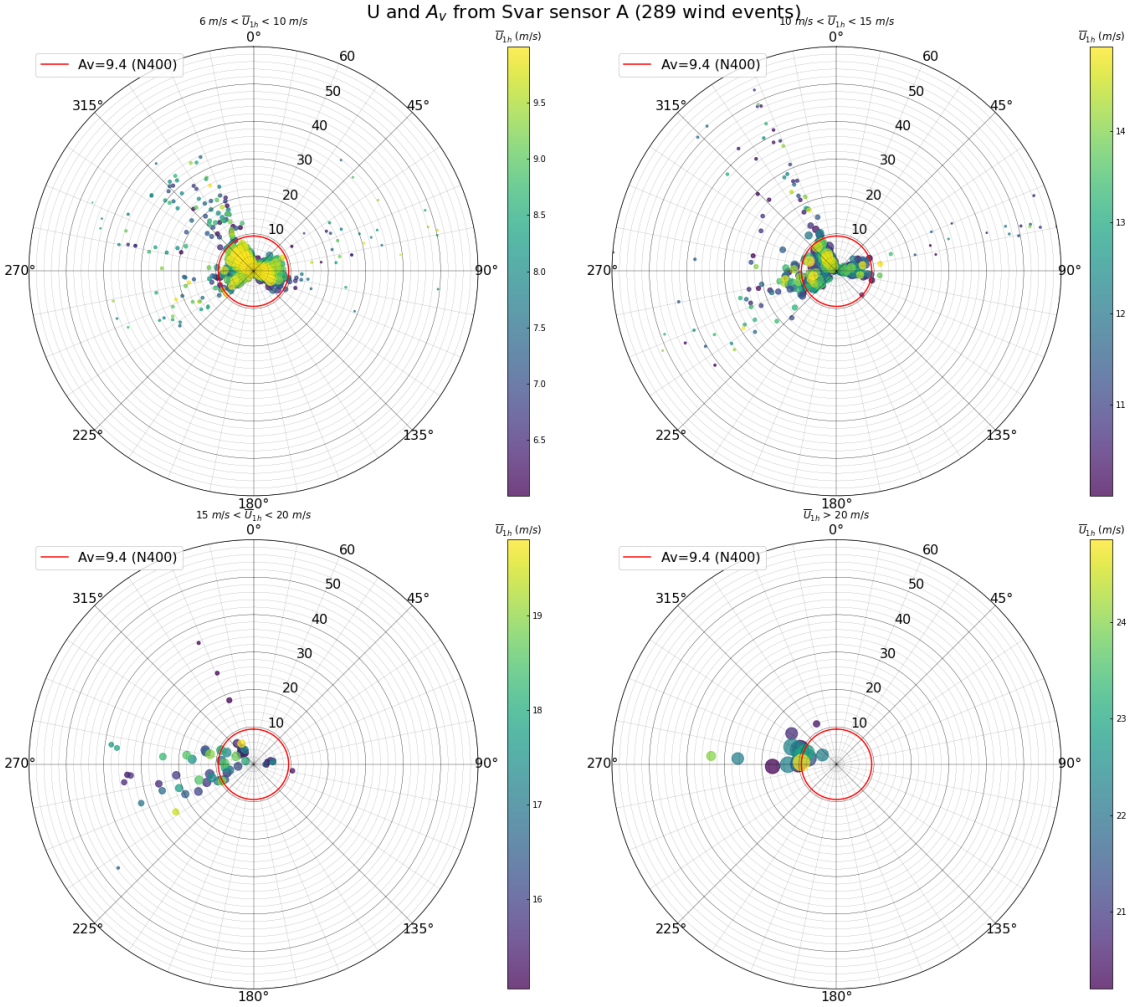

Figure 5.49: Distribution of the fitted A_v with mean wind speed at Svar.

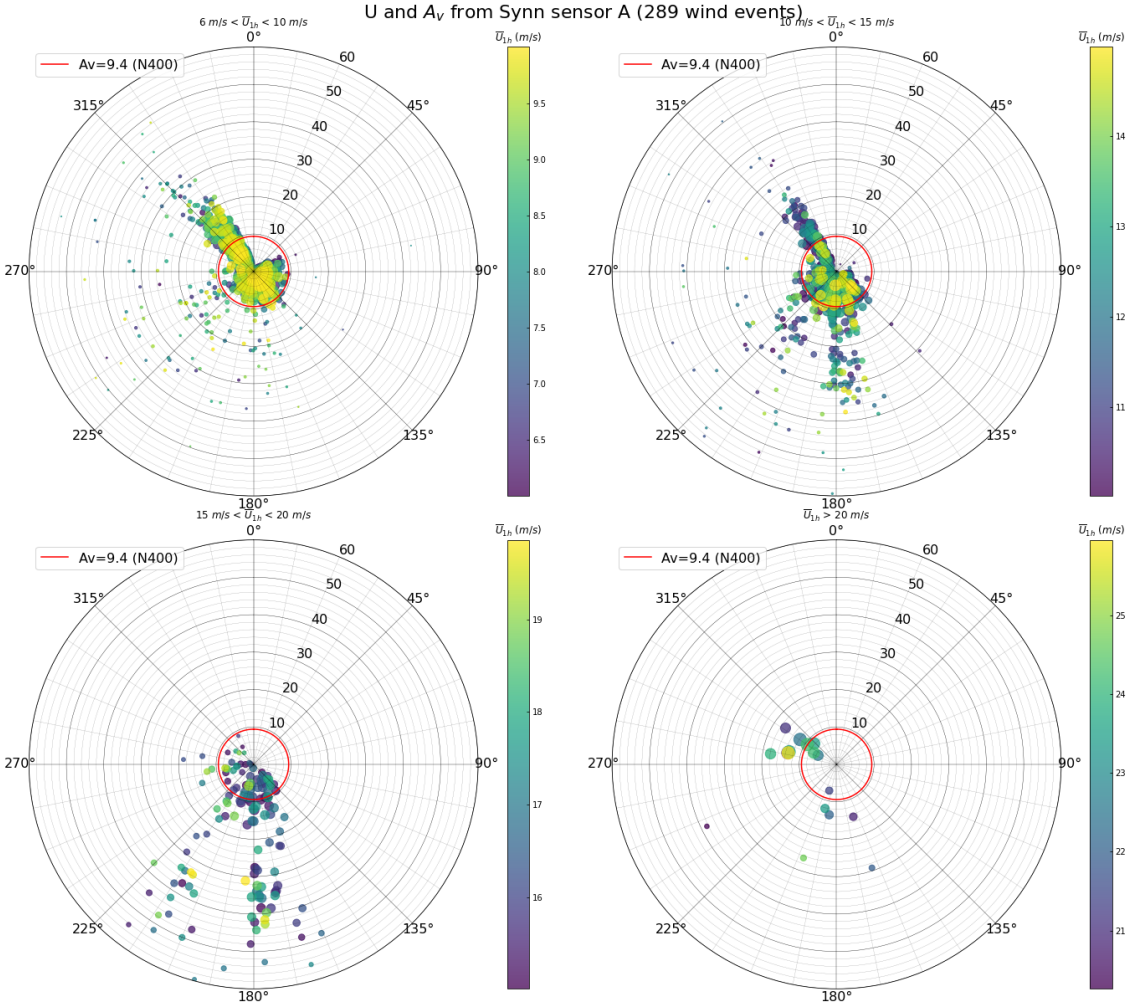

Figure 5.50: Distribution of the fitted A_v with mean wind speed at Synn.

Figures 5.51-5.53 show the distribution of the fitted A_w for the different wind speed ranges. The vertical spectral parameter is generally larger than the reference N400 value for all wind speed ranges, again indicating that N400 underestimates the energy content in the low frequency range. The dominant directions all correspond to wind flow over water. The fitted A_w shows larger values for the U3 and U4 ranges compared to A_u, similar to A_v.

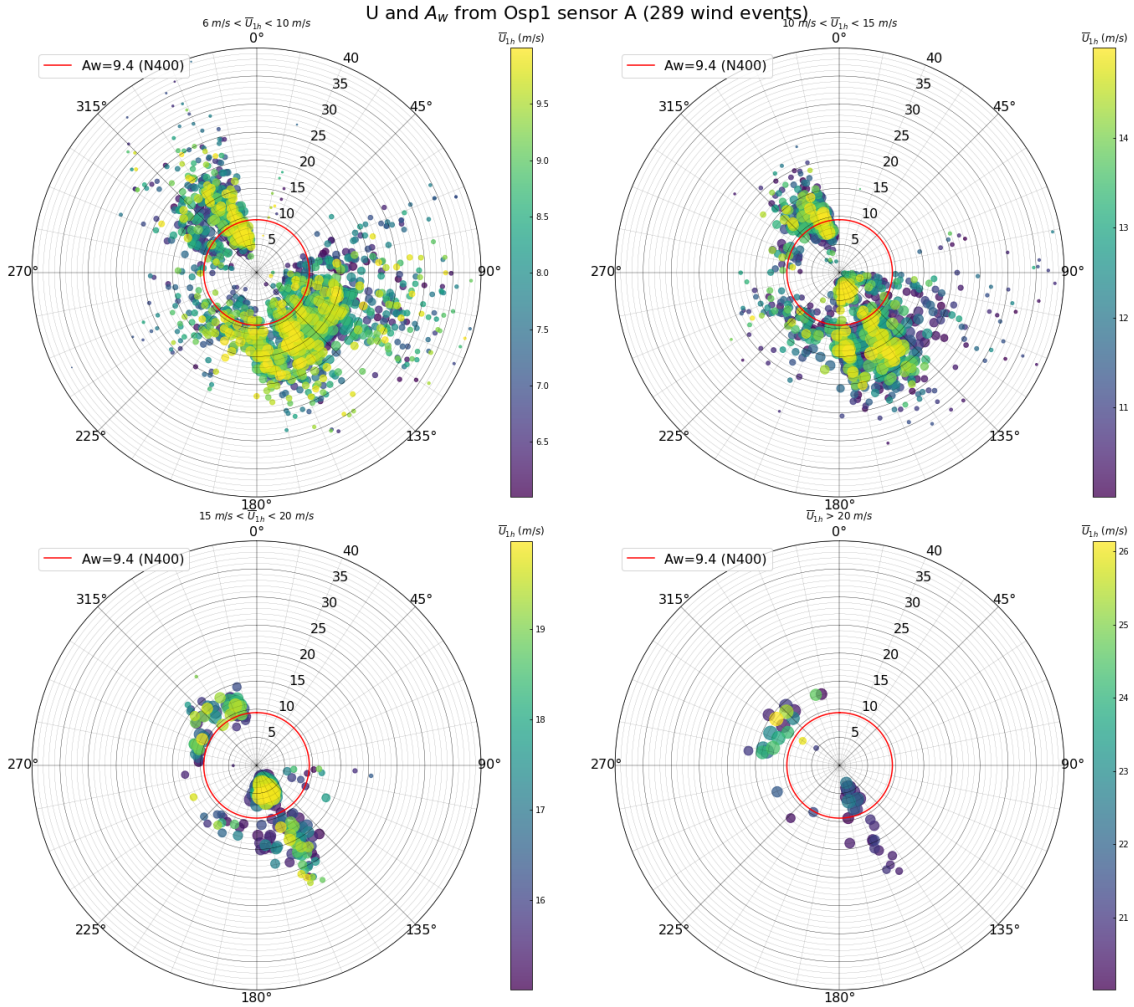

Figure 5.51: Distribution of the fitted A_w with mean wind speed at Osp1.

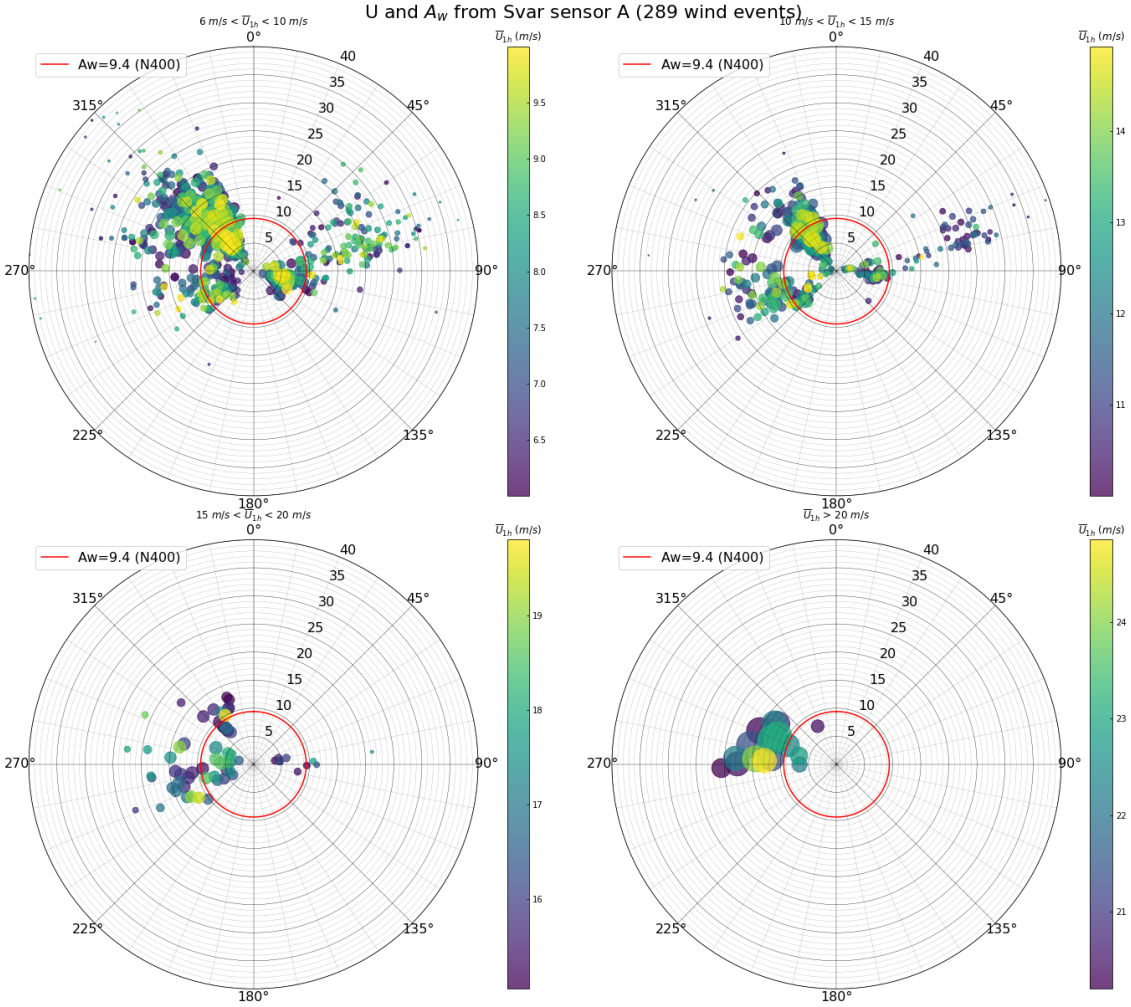

Figure 5.52: Distribution of the fitted A_w with mean wind speed at Svar.

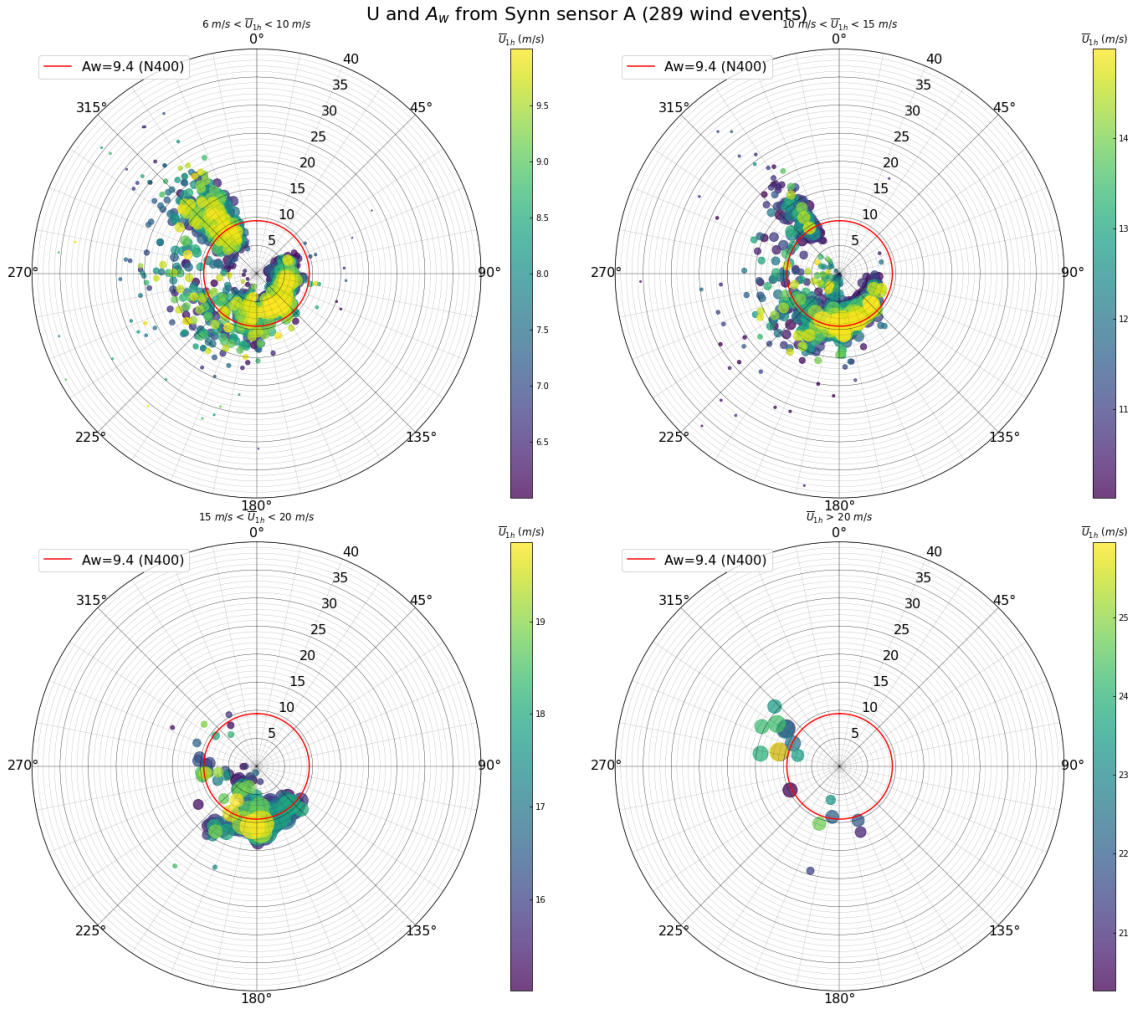

Figure 5.53: Distribution of the fitted A_w with mean wind speed at Synn.

Figures 5.54-5.62 present the results for the fitted spectral parameters in histograms based on the different directional sectors specified in the beginning of the chapter.

At Ospøya in sector 1, as the wind speed range increases, the peak shifts from the right to the left of the reference value of A_u, which agrees with the observations above in which the U3 and U4 ranges showed less scatter for the parameter. In sector 1 at Svarvhelleholmen there are no observations for mean wind speeds over 20 m/s. In sector 3 of all the wind masts, most of the values are lower than the reference value, indicating less energy in the low frequency region.

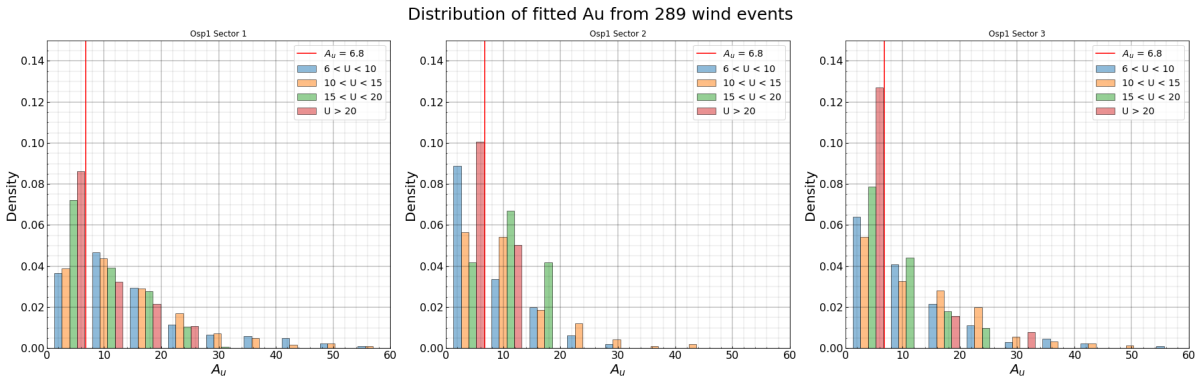

Figure 5.54: Distribution of the fitted A_u with sectors at Osp1.

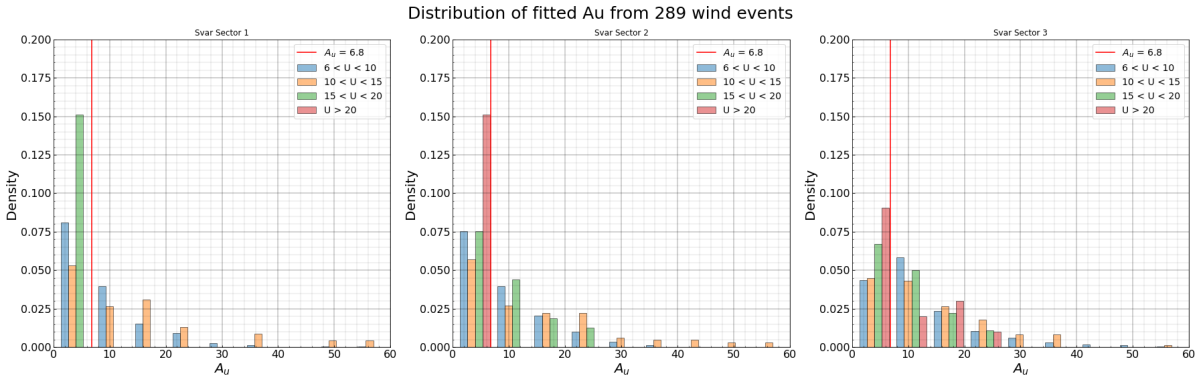

Figure 5.55: Distribution of the fitted A_u with sectors at Svar.

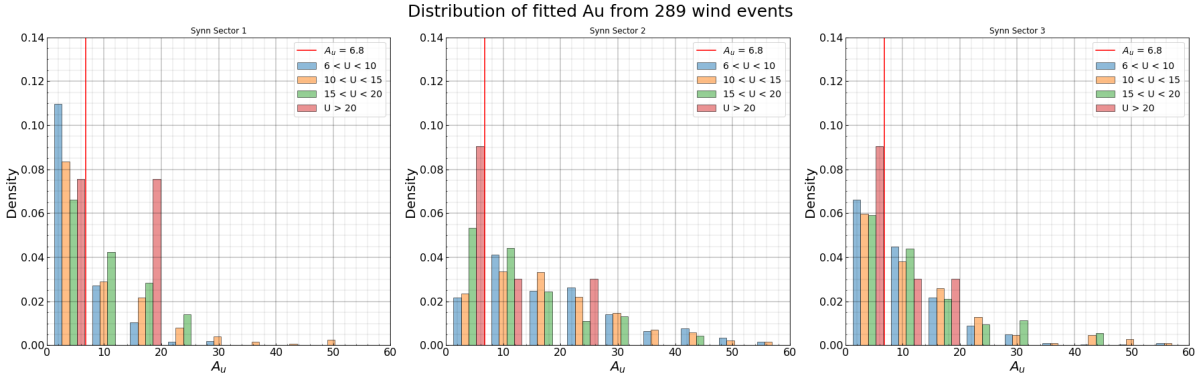

Figure 5.56: Distribution of the fitted A_u with sectors at Synn.

Fitted values for A_v in sector 1 at Ospøya 1 show a fairly uniform distribution for wind speed ranges U2, U3 and U4, while the U1 range has a high peak under the reference value. This is also true for Svarvhelleholmen and Synnøytangen, in addition to the U2 and U3, and U2 ranges, respectively. The U4 range has high peaks larger than the reference value in all the sectors except in sector 1 of Svarvhelleholmen. For all sectors, the general trend is that as the wind speed ranges decrease, the peaks are lower than the reference value.

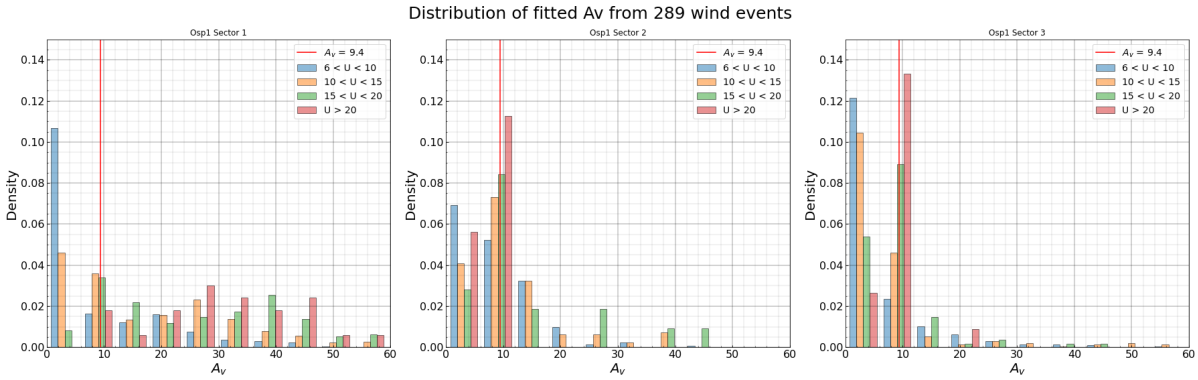

Figure 5.57: Distribution of the fitted A_v with sectors at Osp1.

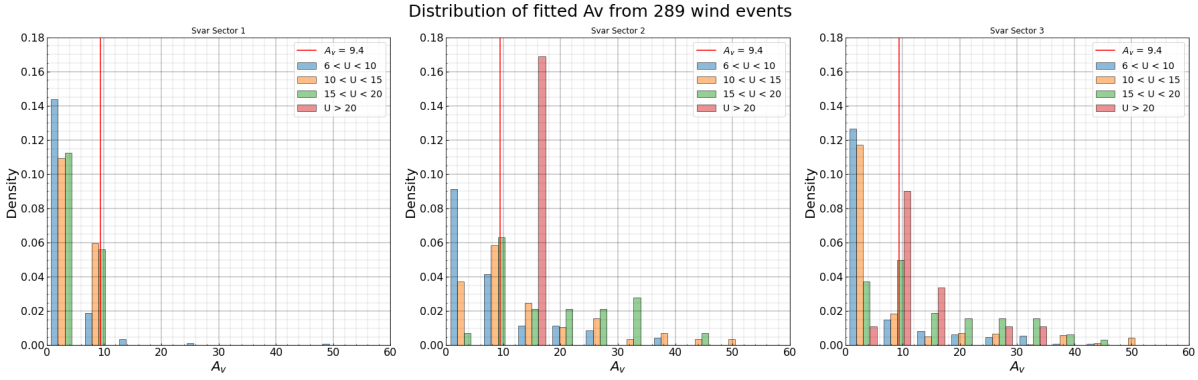

Figure 5.58: Distribution of the fitted A_v with sectors at Svar.

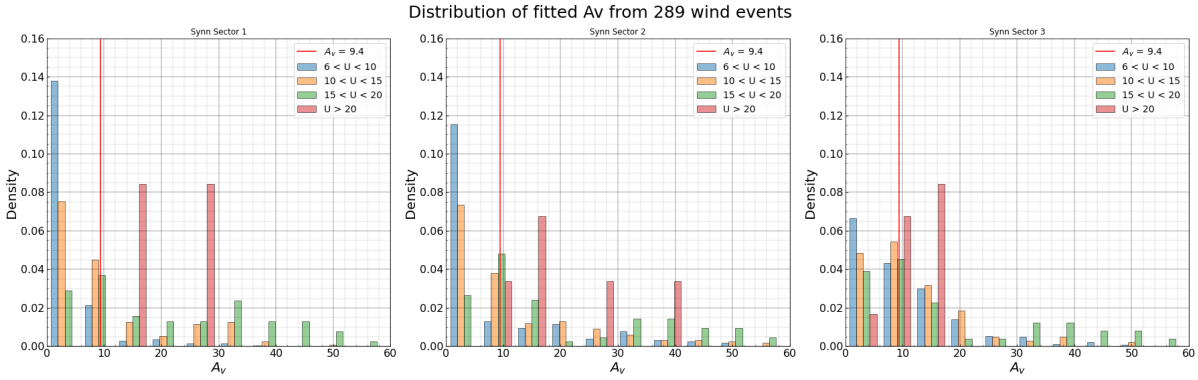

Figure 5.59: Distribution of the fitted A_v with sectors at Synn.

The fitted A_w has a smaller range compared to A_u and A_v, but the general trend for all sectors is that as the wind speed range increases, the peak shifts to the right of the reference value, which again indicates more energy in the low frequency region.

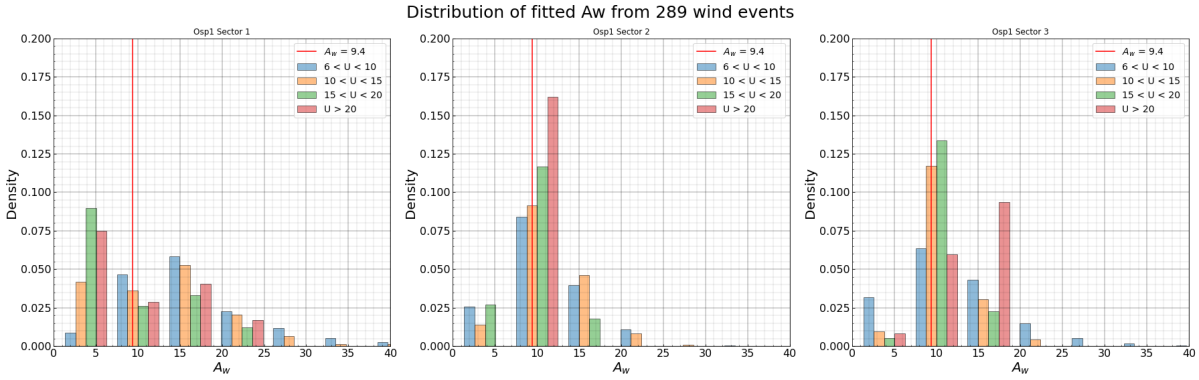

Figure 5.60: Distribution of the fitted A_w with sectors at Osp1.

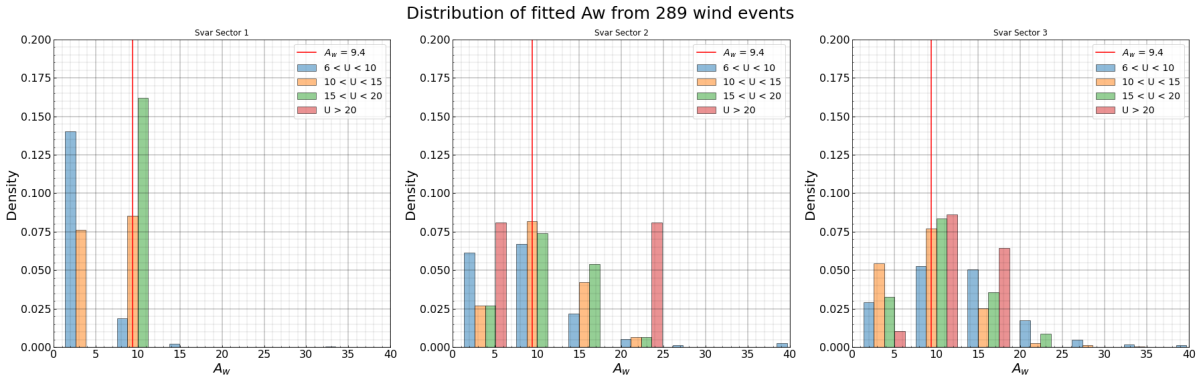

Figure 5.61: Distribution of the fitted A_w with sectors at Svar.

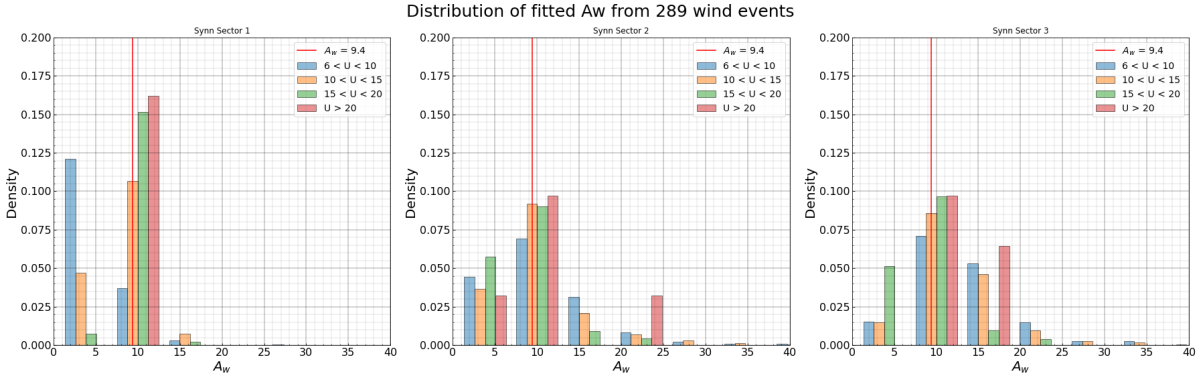

Figure 5.62: Distribution of the fitted A_w with sectors at Synn.

5.6.1.2 Distribution with atmospheric stability

Figures 5.63-5.65 show the distribution of the fitted A_u under different conditions of the atmospheric stability. Unstable, neutral, and stable conditions are represented by the first, second, and third column respectively. The first row shows the cases for mean wind velocities less than 15 m/s and the second shows for higher than 15 m/s. Li et. al. (2012) noted that the atmospheric stability significantly influenced the turbulent energy distribution in the low

frequency range [9]. For the results presented here, the stability does have somewhat of an influence, but not as much as was expected. This may be because when estimating the stability parameter using Equation 2.24, it is dependent on whether the vertical turbulence component was corrected for tilt, as mentioned in section 5.3. As it is difficult to notice any particular trends in how the spectral parameters vary with stability, only the results for A_u are shown here. The figures for A_v and A_w will be given in the appendix.

Most of the observations for all the wind masts are for the stability cases S1, S3, S5 and S4. For wind flow over water, the fitted A_u shows large scatter in all stability conditions, however it is still for the most part larger than the reference value. In S4, the same scatter is present particularly from the south-east and west with A_u larger than the reference value. Values for A_u at Ospøya and Svarvhelleholmen from the east are larger for unstable conditions in comparison to near-neutral and stable conditions, and the opposite is the case for Synnøytangen.

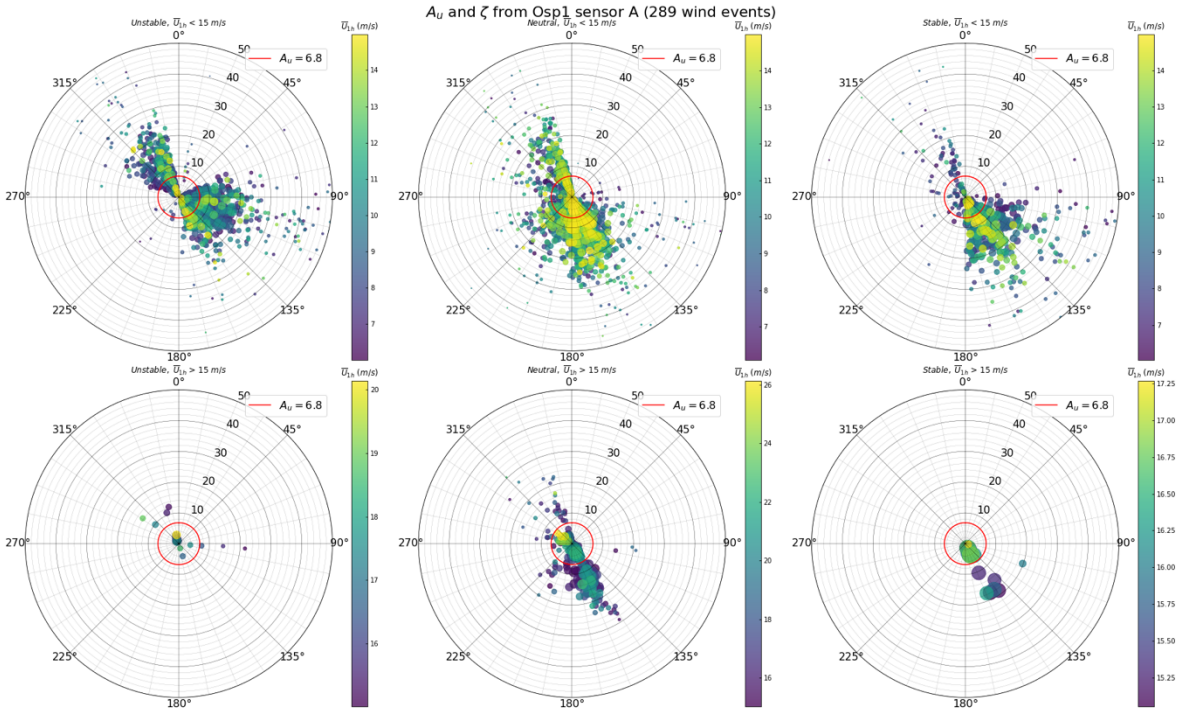

Figure 5.63: Distribution of the fitted A_u with stability at Osp1.

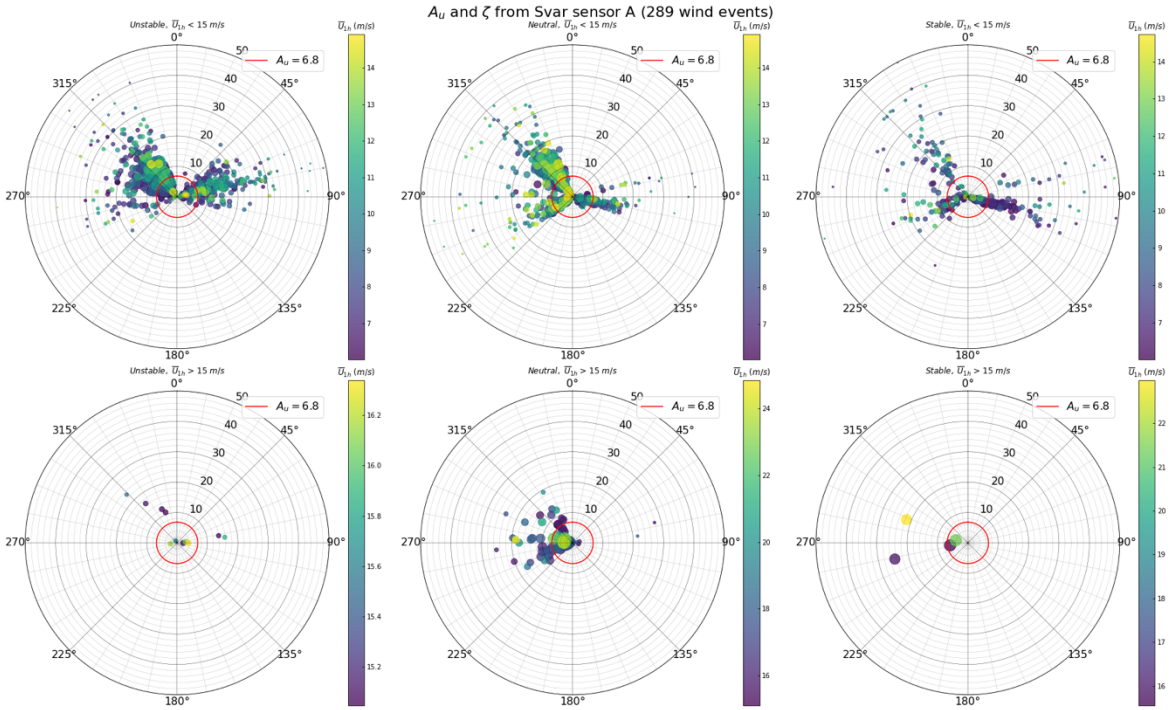

Figure 5.64: Distribution of the fitted A_u with stability at Svar.

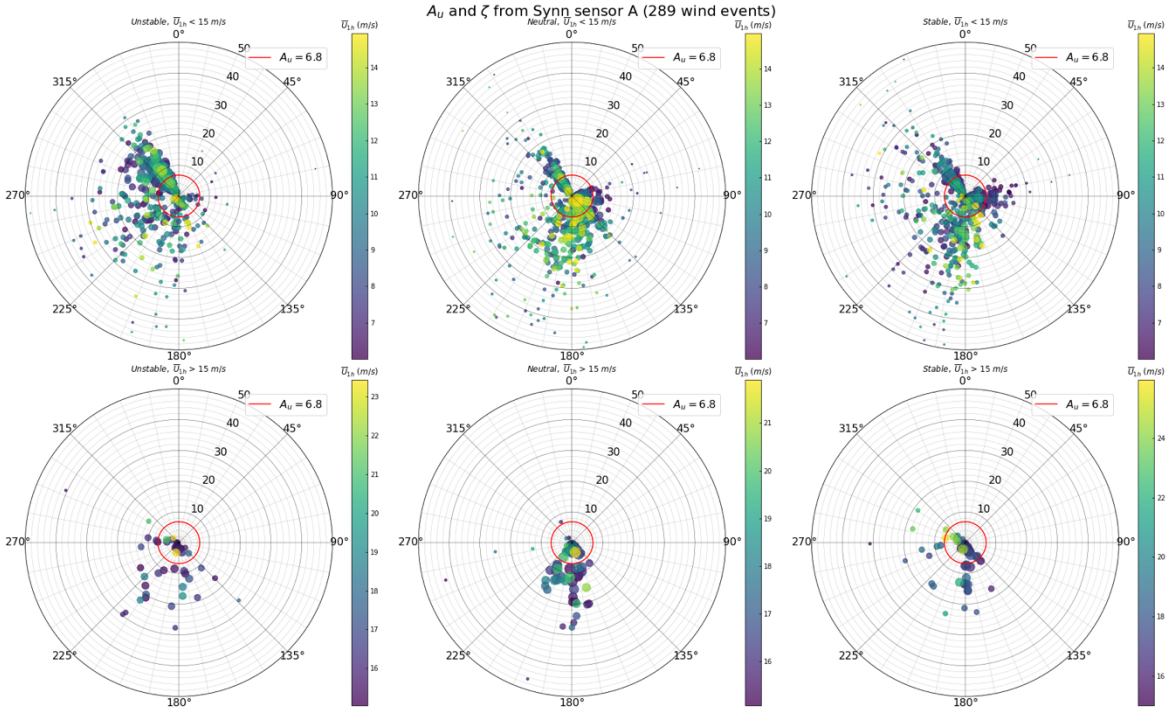

Figure 5.65: Distribution of the fitted A_u with stability at Synn.

To better see the effect of stability on the fitted parameters, Figure 5.66 shows the variation of A_u with stability only in the eastern sector (from 65˚ to 115˚) for wind speeds less than 15 m/s. This sector corresponds to wind flow over water for Ospøya and Svarvhelleholmen, and wind

flow over land for Synnøytangen. From this figure it is easier to see the dominance of unstable conditions at Ospøya, which agrees with existing literature. Svarvhelleholmen has more stable conditions in comparison, as the wind mast is located on the western side of the island, and at Synnøytangen stable conditions dominate as the wind travels mostly over land. The main observation is that A_u shows large scatter in unstable conditions, while for stable conditions, particularly at Svarvhelleholmen and Synnøytangen, the scatter is not as broad, and A_u is closer to the reference N400 value.

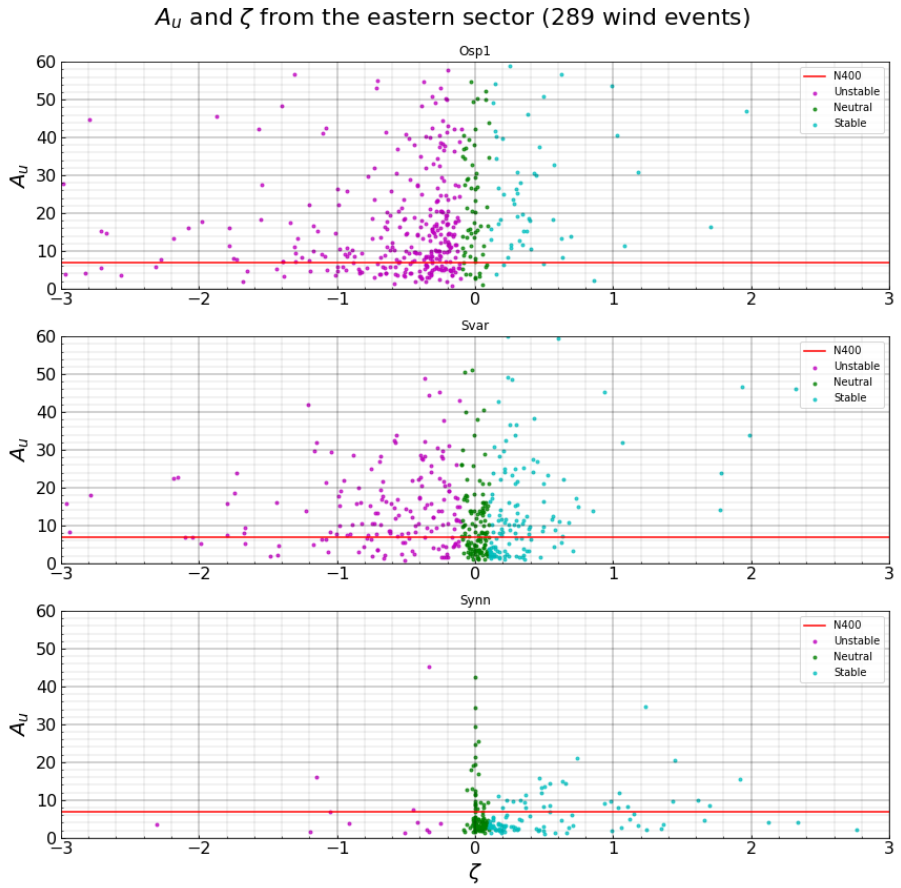

Figure 5.66: Distribution of the fitted A_u with stability in the eastern sector at Osp1, Svar and Synn.

5.7 Coherence parameters

A comparison between the fitted and the reference coherence models at the wind masts in Ospøya are presented in Figure 5.67 and Figure 5.68. Although the coherence models using the reference values generally give an adequate representation of the estimated coherence, using the fitted parameters gives the variation of the coherence over the frequency range even better. In the frequency range of the bridge's eigen-frequencies, the fitted horizontal coherence tends to underestimate the coherence, but only by a small margin. Near frequencies of 1 Hz, the fitted horizontal coherence sometimes overestimates the actual coherence, as seen at Ospøya 2. The fitted vertical coherence varies in how well it represents the coherence between Ospøya 1 and 2. This may be due to Ospøya 2 having a broader scatter of its estimated coherence in comparison to Ospøya 1, where the terrain is more complex for the former.

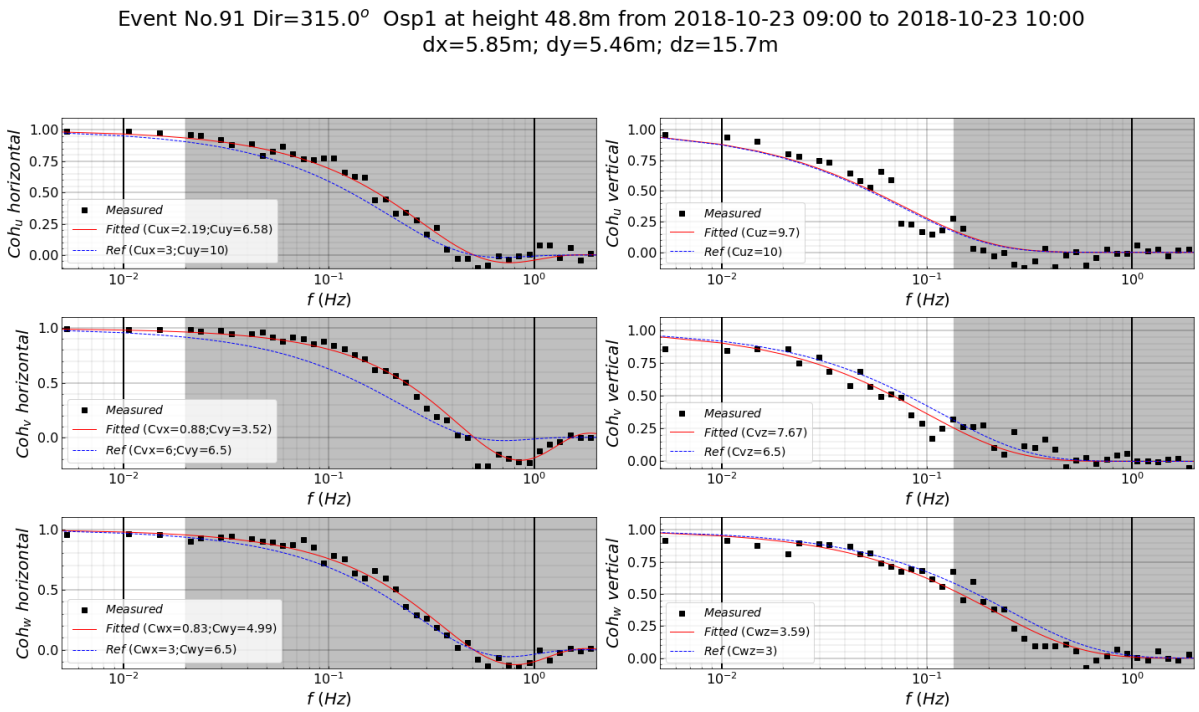

Figure 5.67: Comparison of fitted and reference coherence model on October 23, 2018, at Osp1.

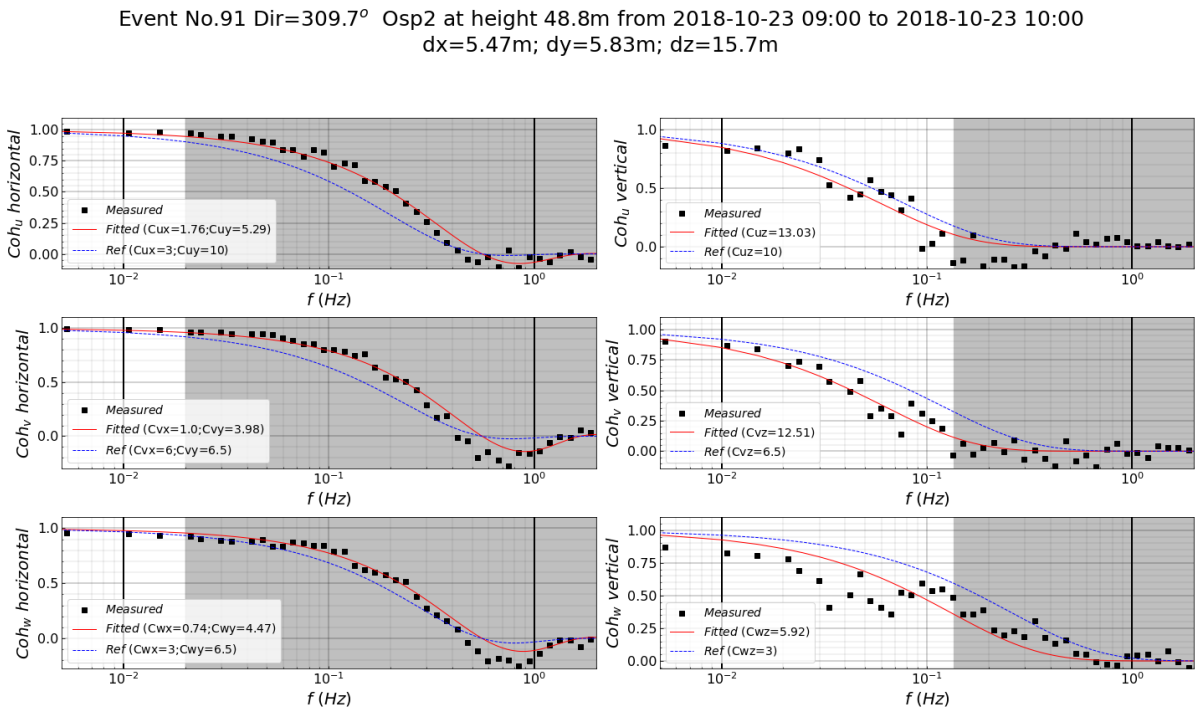

Figure 5.68: Comparison of fitted and reference coherence model on October 23, 2018, at Osp2.

In the following sections, only the C_{iy} and C_{iz} components are studied further, as the fitted C_{ix} depends on the fitting of C_{iy}.

5.7.1 Distribution of the fitted parameters

5.7.1.1 Lateral turbulence

Figure 5.69 and Figure 5.70 show the directional distribution the fitted C_{uy} at Ospøya 1 and 2. From the south in the ranges U1 to U3 there is a noticeable cluster of fitted values that are higher than the reference. This is observed at both wind masts but is more pronounced at Ospøya 1. Most of the fitted values range from 3 to 9 and fall under the reference value of 10 from N400, i.e. there is more coherence than the code suggests. Figure 5.71 shows the distribution of C_{uy} in each sector. In sector 1, when the wind flow is over water, C_{uy} is smaller compared to sector 2 and 3, where the wind flows over a combination of land and water.

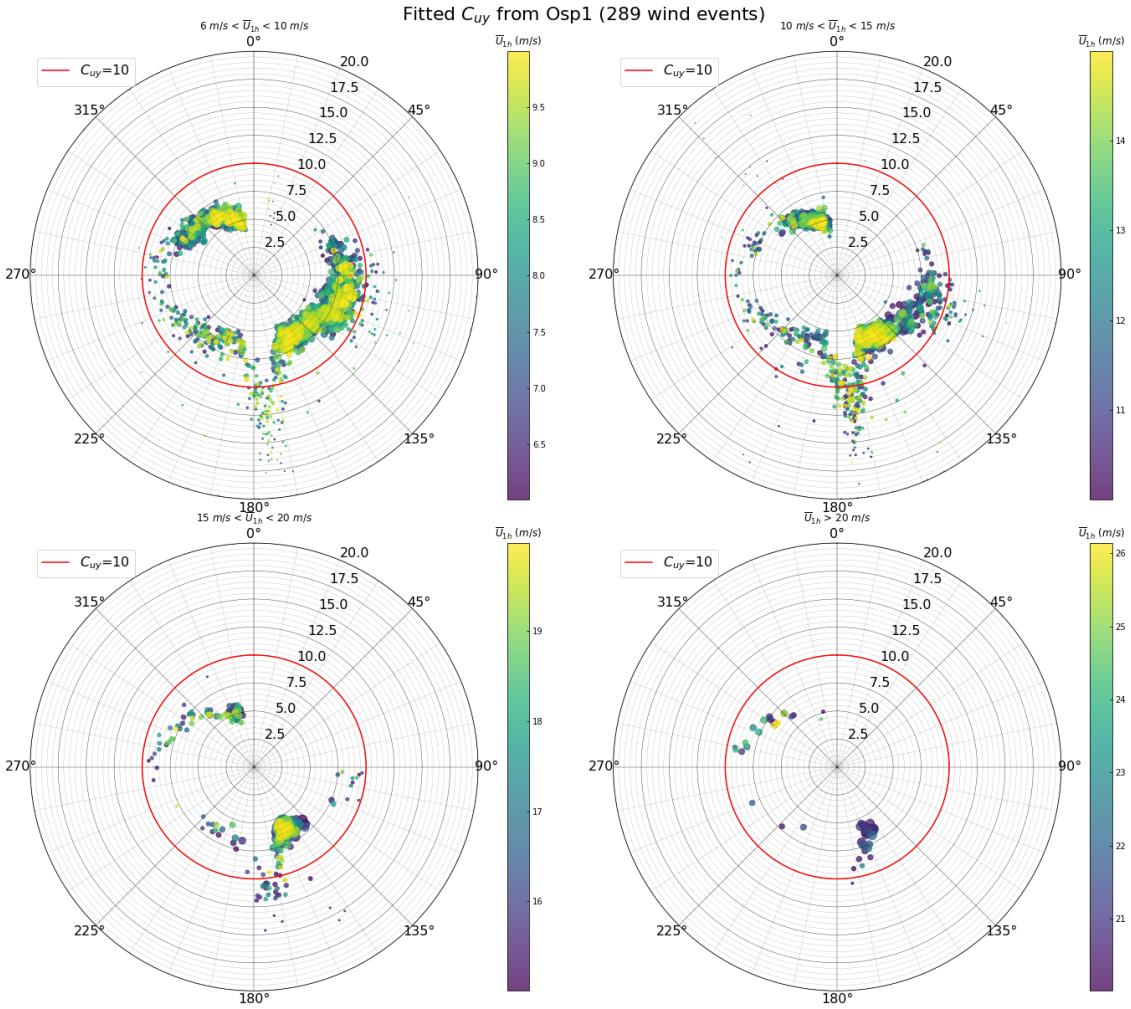

Figure 5.69: Distribution of the fitted C_{uy} with mean wind speed at Osp1.

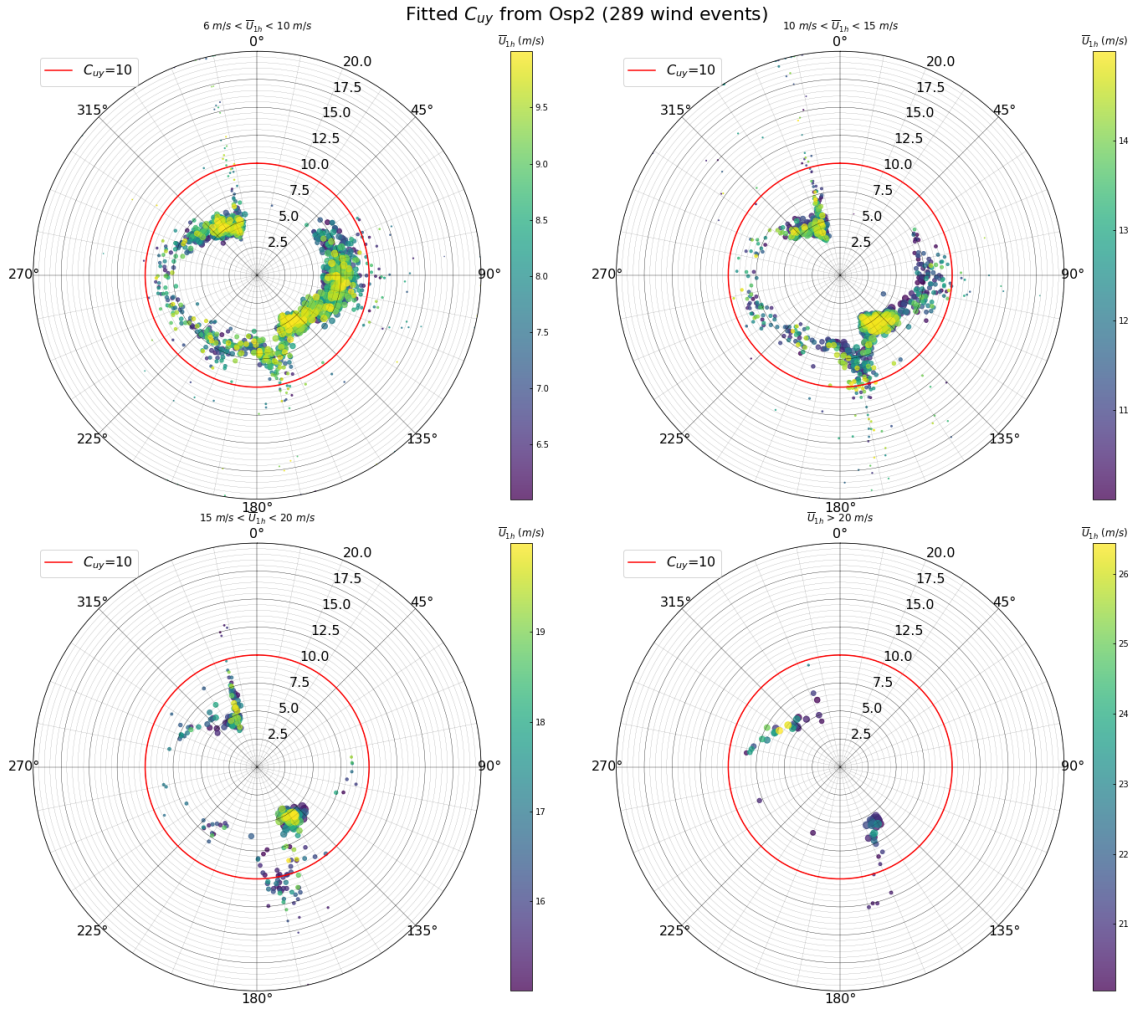

Figure 5.70: Distribution of the fitted C_{uy} with mean wind speed at Osp2.

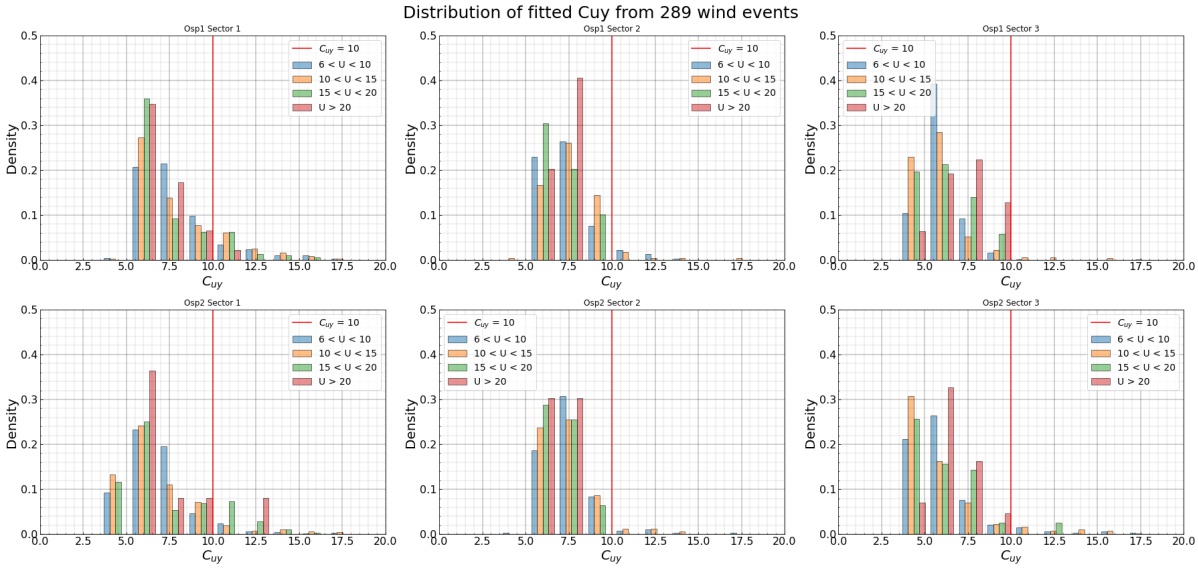

Figure 5.71: Distribution of the fitted C_{uy} with sectors at Osp1 and Osp2.

Figure 5.72 and Figure 5.73 show the distribution of the fitted C_{vy}. C_{vy} shows very little difference between the two wind masts at Ospøya. The cluster of values in the south is present again as it was for C_{uy}, however the values generally do not exceed the reference value of 6.5. As expected, the coherence from the lateral turbulence component is stronger in the lateral direction compared to the longitudinal direction. As the wind speed range increases, the values move closer to the reference. Most of the values for C_{vy} lie between 2 and 4. Figure 5.74 shows that for sector 1 the values are more spread out, and for sector 2 and 3 the coherence tends to be lower, which is the opposite of what was observed for C_{uy}. This may be due to the lower surface roughness of the water compared to land.

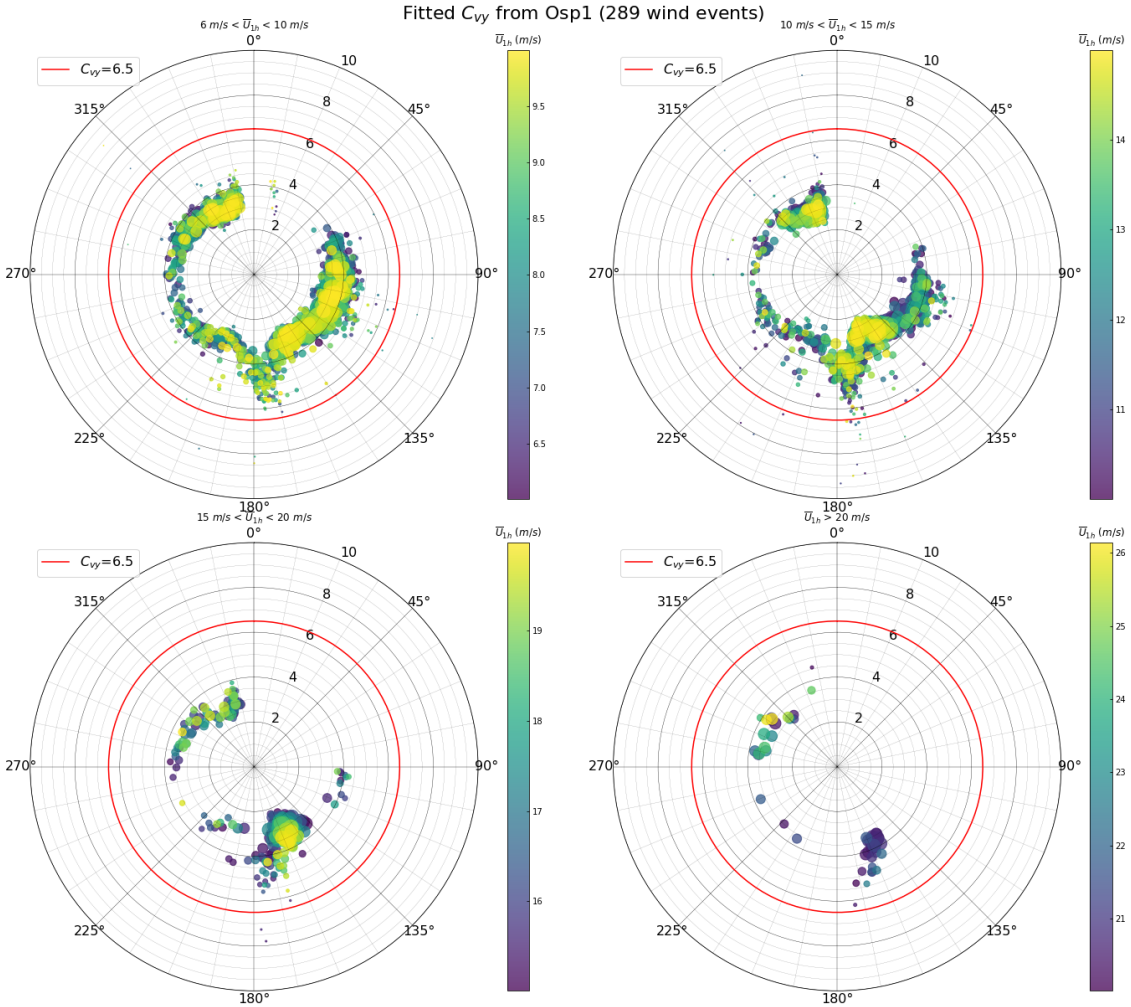

Figure 5.72: Distribution of the fitted C_{vy} with mean wind speed at Osp1.

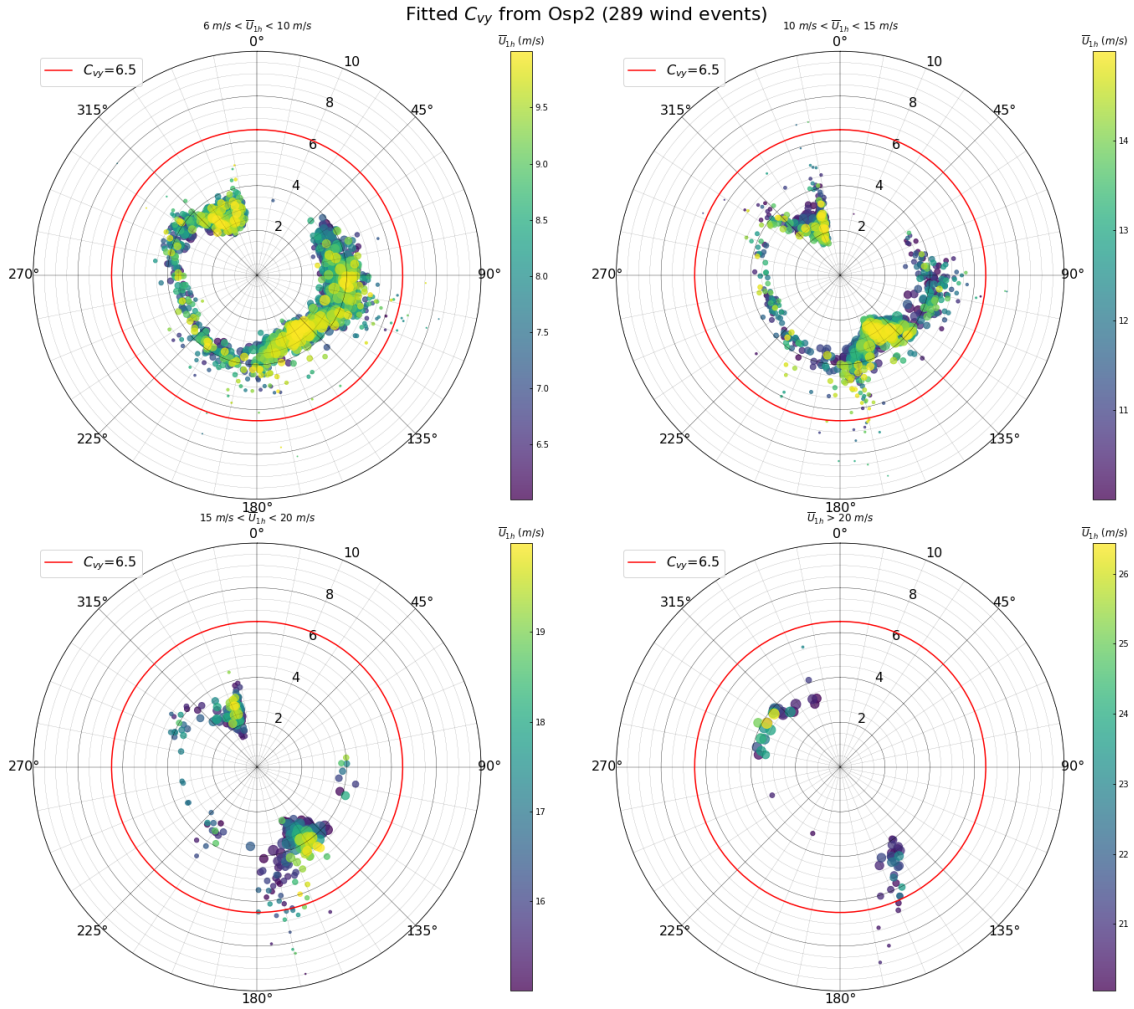

Figure 5.73: Distribution of the fitted C_{vy} with mean wind speed at Osp2.

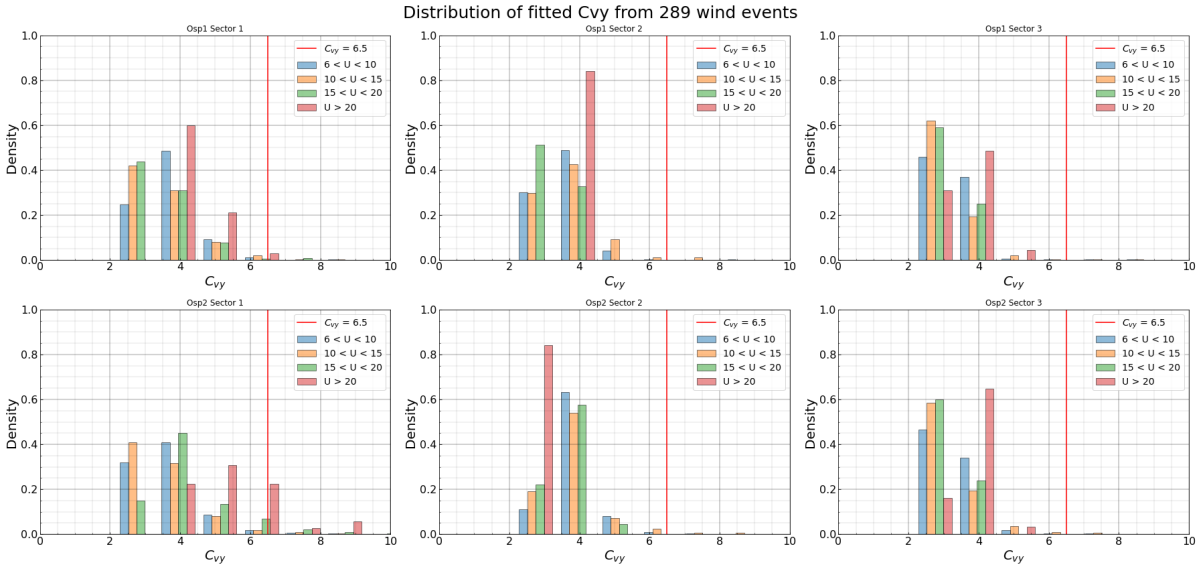

Figure 5.74: Distribution of the fitted C_{vy} with sectors at Osp1 and Osp2.

Figure 5.75 and Figure 5.76 show the distribution of the fitted C_{wy}. Compared to C_{vy}, the values are much closer to the reference value, which is the same for both parameters. This indicates less coherence from the vertical turbulence component in the lateral direction. The wind masts at Ospøya 1 and 2 show very little difference, except for a small sector from 140˚ to 160˚ at Ospøya 2, where there is a sharp decrease in the fitted values. C_{wy} gets closer to the reference value as the wind speed range increases. Figure 5.77 shows that there is some variation between sectors for the two wind masts, suggesting that the topography has an effect of the coherence in the vertical direction. When the wind is travelling down the mountain towards Ospøya 1, C_{wy} increases, meaning there is less coherence, as seen in Sector 1 and 2.

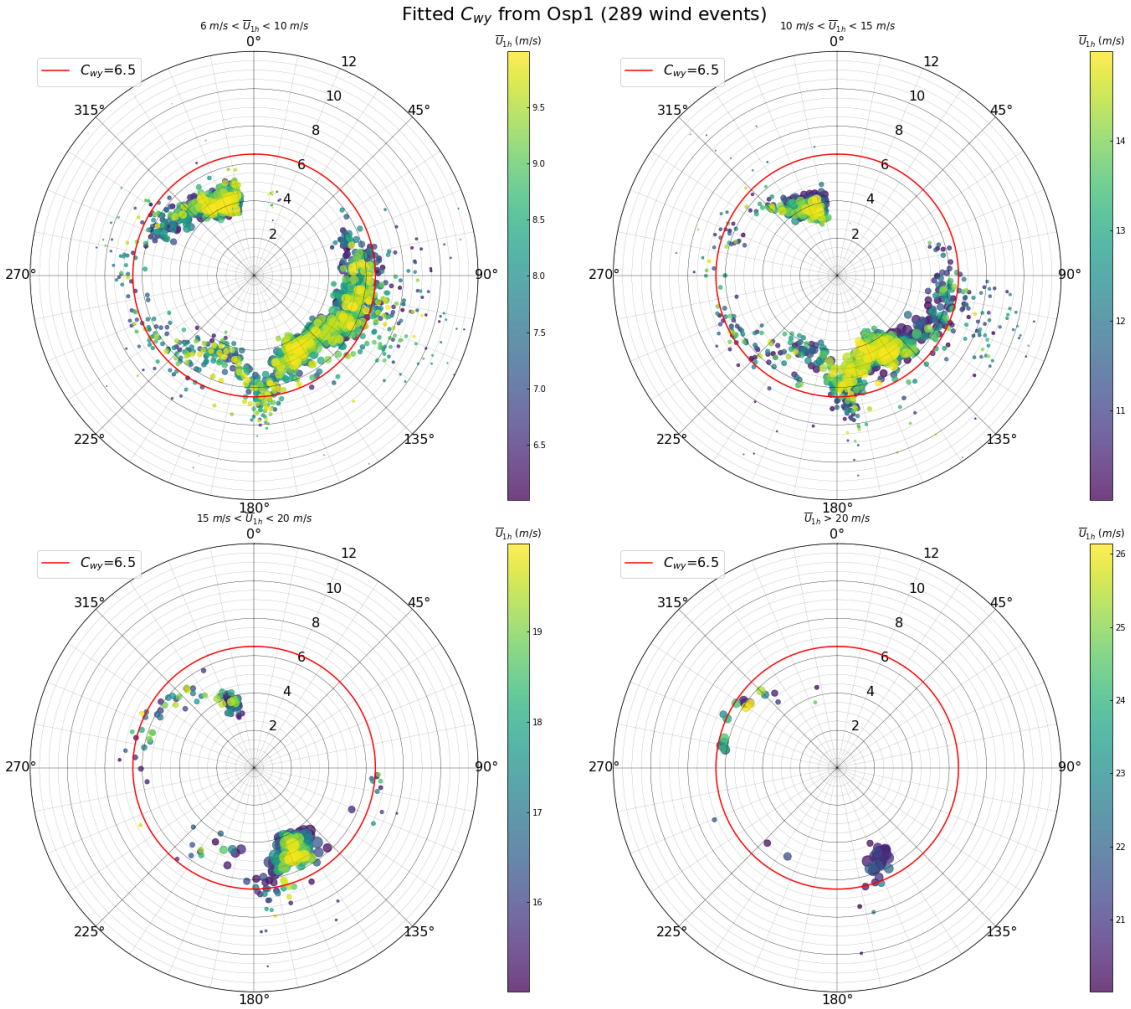

Figure 5.75: Distribution of the fitted C_{wy} with mean wind speed at Osp1.

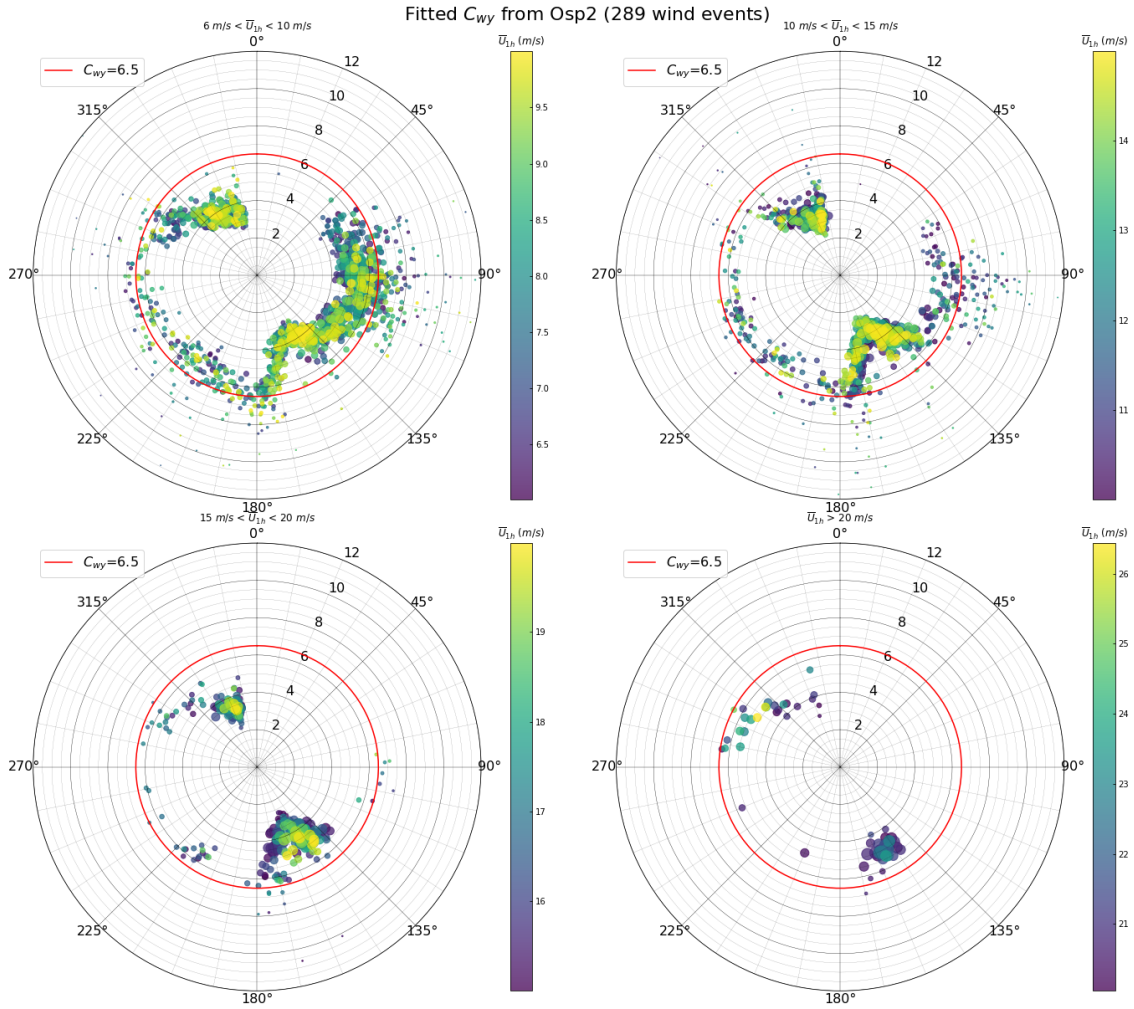

Figure 5.76: Distribution of the fitted C_{wy} with mean wind speed at Osp2.

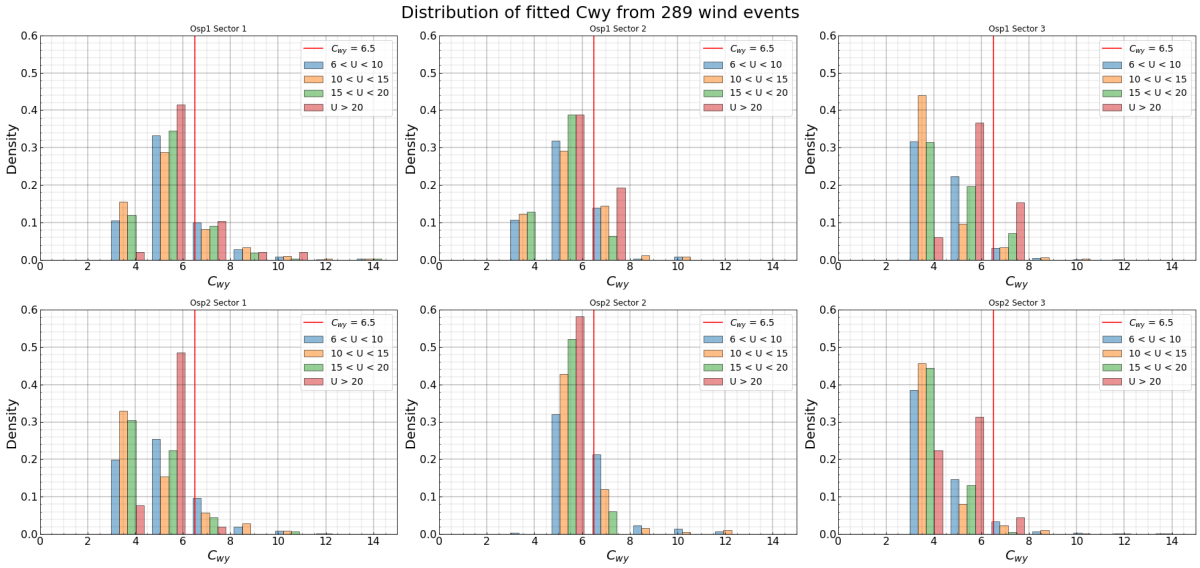

Figure 5.77: Distribution of the fitted C_{wy} with sectors at Osp1 and Osp2.

5.7.1.2 Vertical turbulence

Figure 5.78 and Figure 5.79 show the directional distribution the fitted C_{uz}. For all wind speed ranges, C_{uz} is consistently higher than the reference value of 10, which indicates less coherence than N400 suggests in the vertical direction. This is the case for both wind masts at Ospøya. A similar spike in values is present for the same direction as for the other parameters. Figure 5.80 shows that the values at Ospøya 1 and 2 are mostly similar, with C_{uz} being higher than the reference. Sector 1 at Ospøya 1 has a broader scatter and sector 2 has a higher peak for all wind speed ranges.

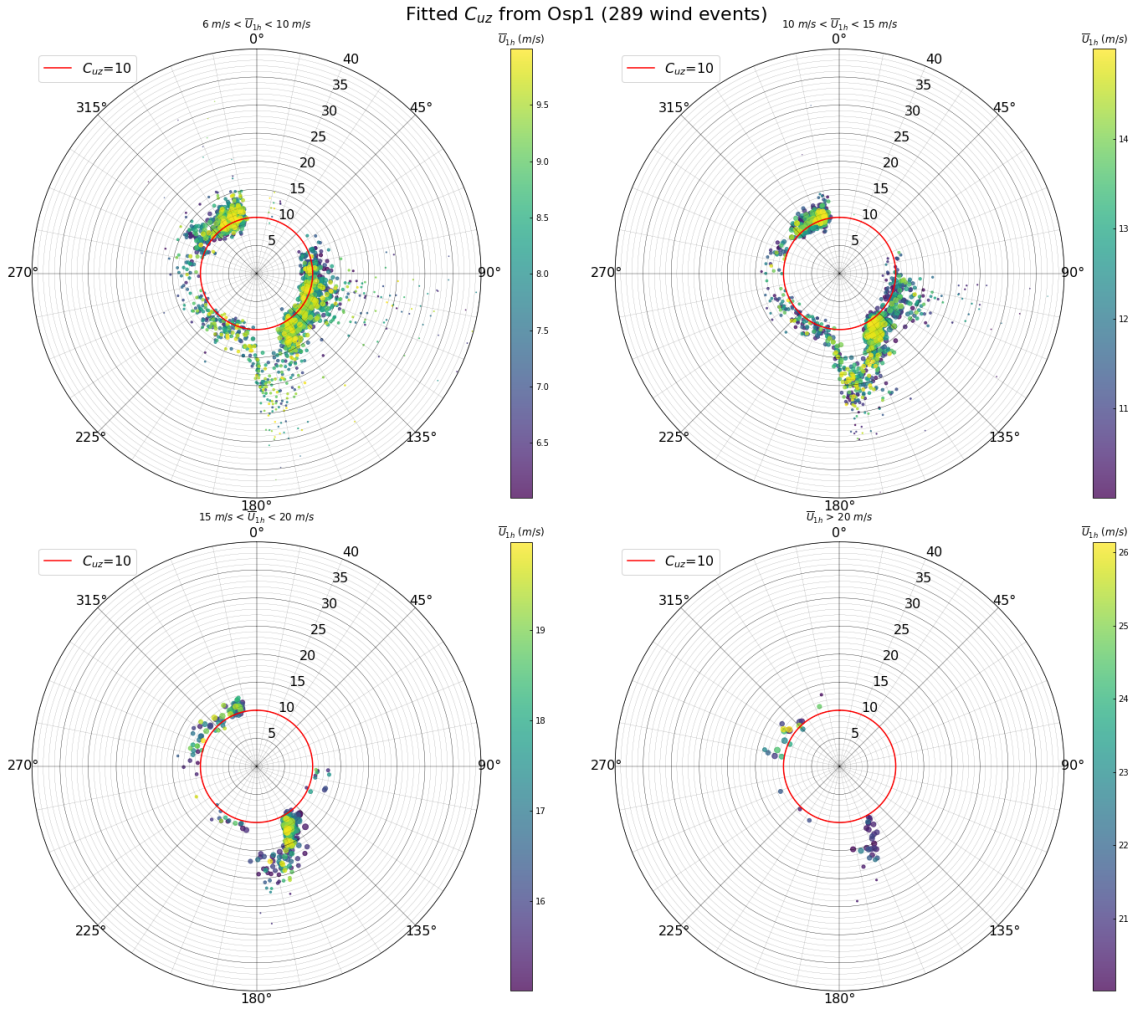

Figure 5.78: Distribution of the fitted C_{uz} with mean wind speed at Osp1.

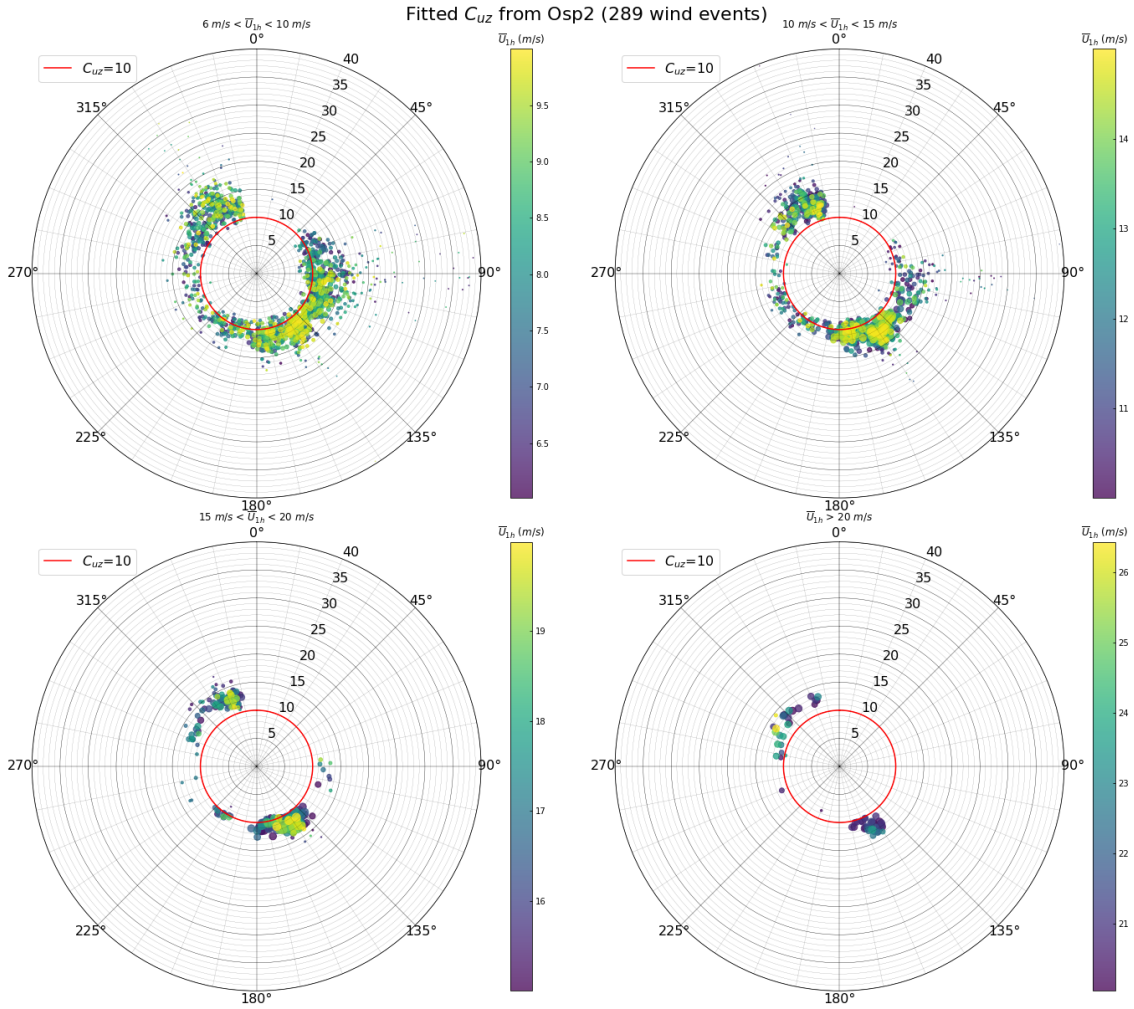

Figure 5.79: Distribution of the fitted C_{uz} with mean wind speed at Osp2.

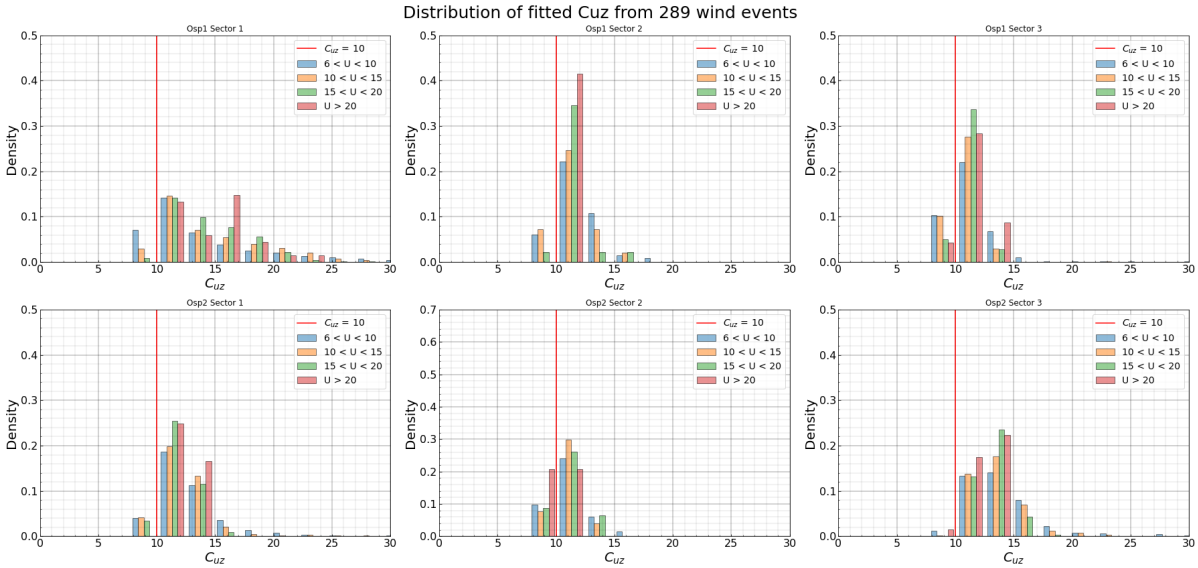

Figure 5.80: Distribution of the fitted C_{uz} with sectors at Osp1 and Osp2.

Figure 5.81 and Figure 5.82 show the distribution of the fitted C_{vz}, which similar to C_{uz} is also consistently higher than the reference value, in this case 6.5. In Figure 5.83, all sectors generally have the same scatter of values, except for sector 3 in Ospøya 2, where the peak is shifted slightly more to the right of the reference value compared to others.

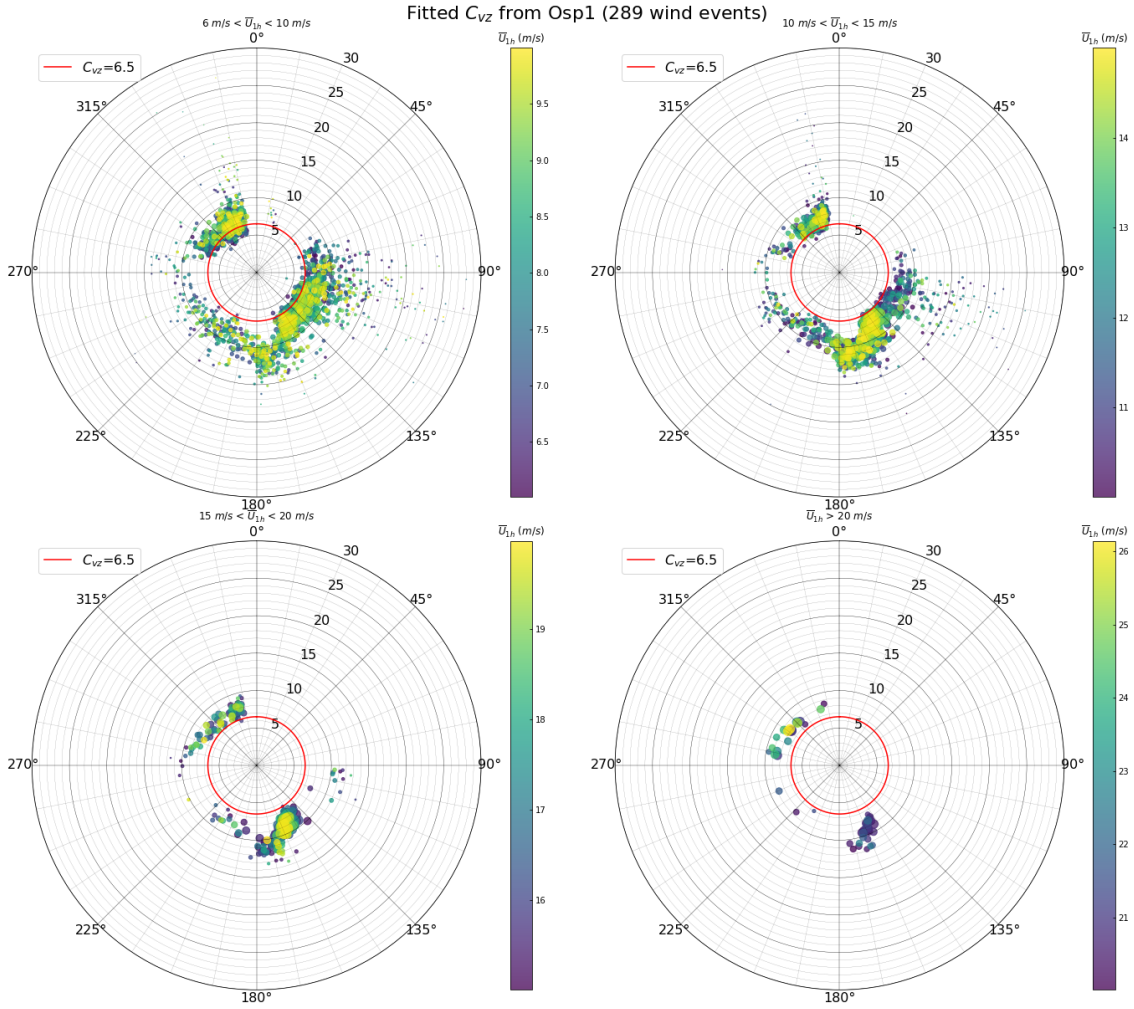

Figure 5.81: Distribution of the fitted C_{vz} with mean wind speed at Osp1.

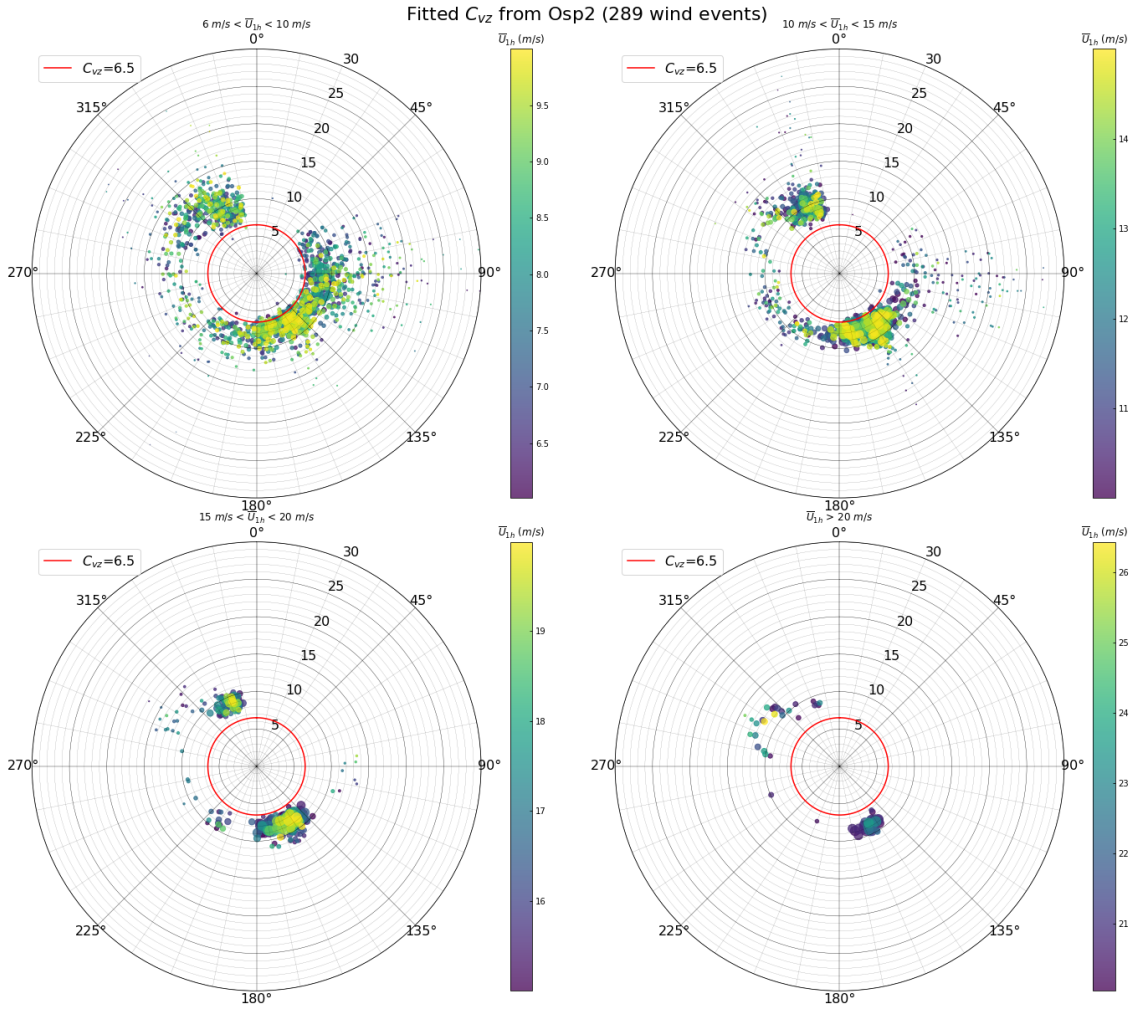

Figure 5.82: Distribution of the fitted C_{vz} with mean wind speed at Osp2.

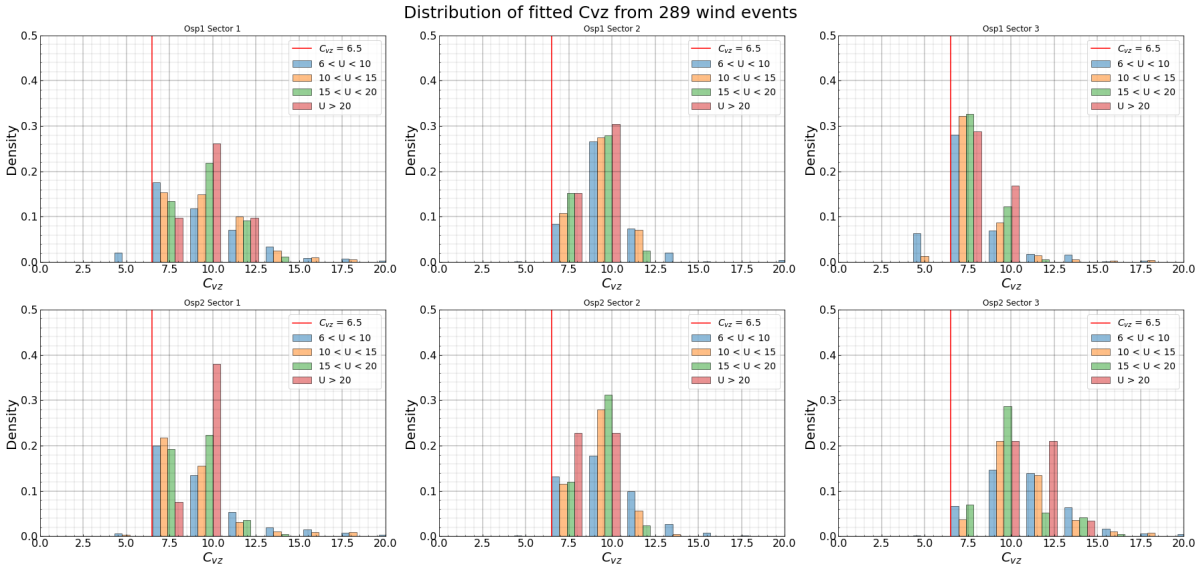

Figure 5.83: Distribution of the fitted C_{vz} with sectors at Osp1 and Osp2.

Figure 5.84 and Figure 5.85 show the distribution of the fitted C_{wz}. Similar to C_{uz} and C_{vz}, most values of C_{wz} are higher than the reference value, indicating less coherence for the vertical direction in general. Both wind masts show similar results except for winds coming from the south, where Ospøya 1 shows much higher values than Ospøya 2. Same as for C_{wy}, this may be due to the wind flowing down the mountain towards Ospøya 1, indicating lower coherence, and this same effect is observed from the north-west in Ospøya 2. These observations are reflected in Figure 5.86 where Ospøya 1 has broader scatter compared to Ospøya 2 in sector 1, and vice versa in sector 3.

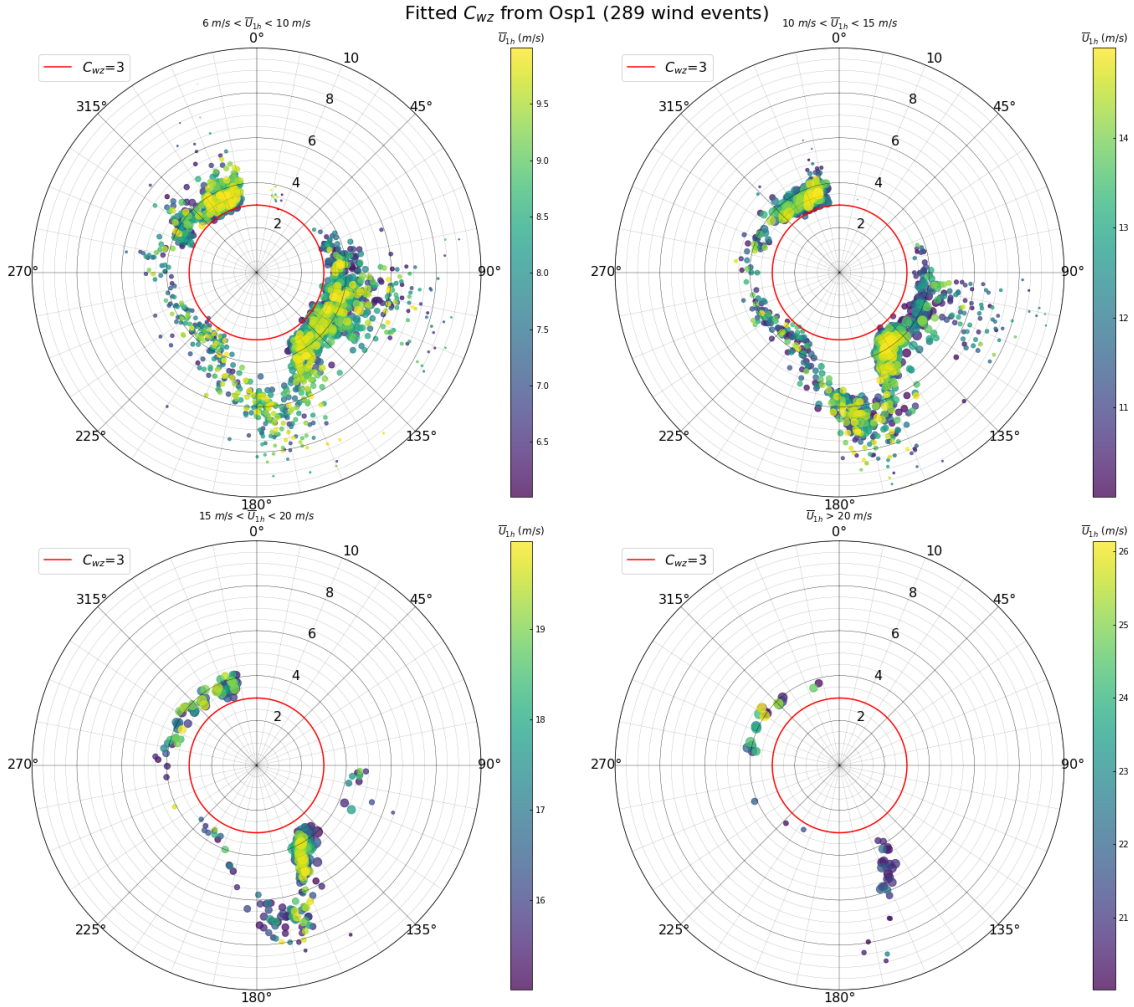

Figure 5.84: Distribution of the fitted C_{wz} with mean wind speed at Osp1.

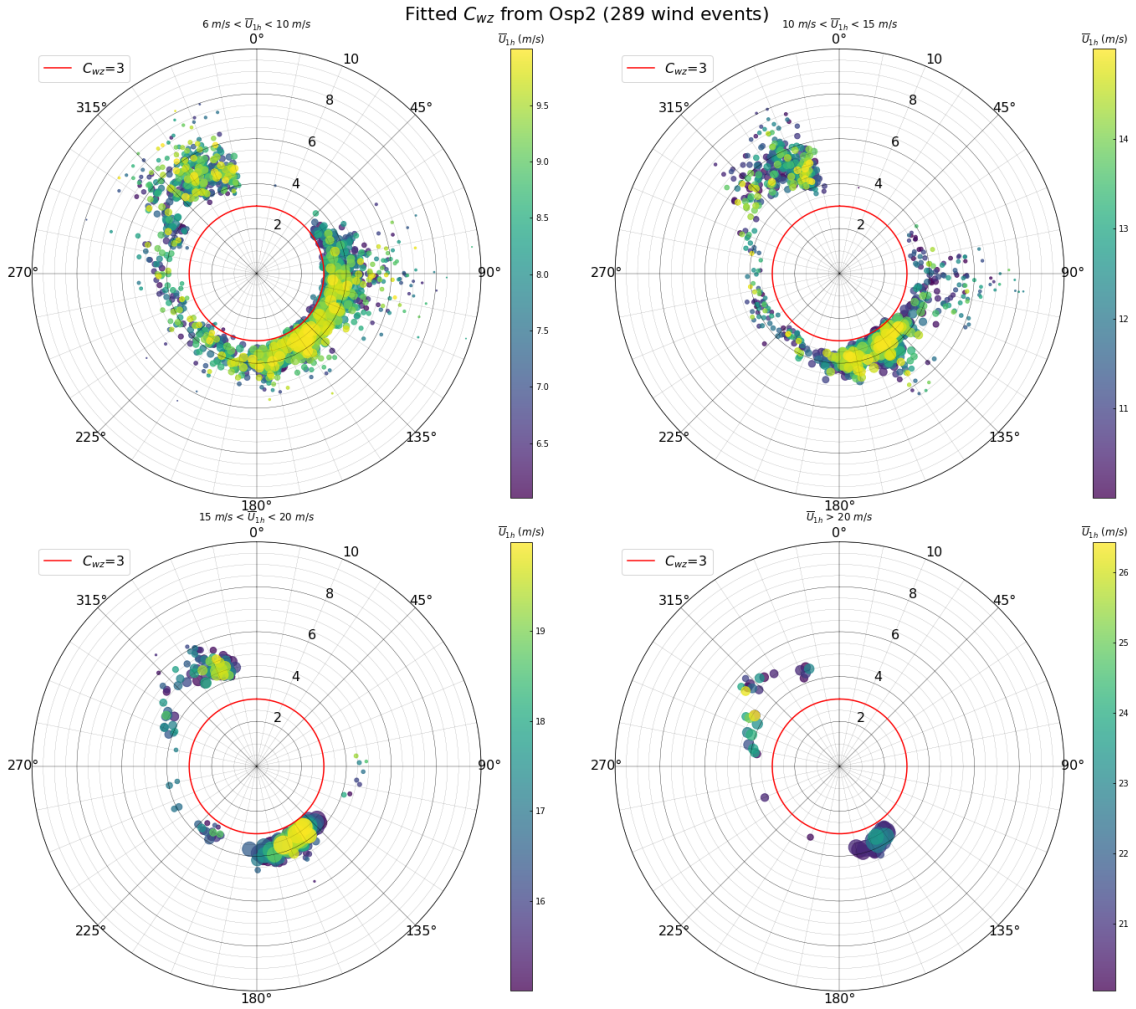

Figure 5.85: Distribution of the fitted C_{wz} with mean wind speed at Osp2.

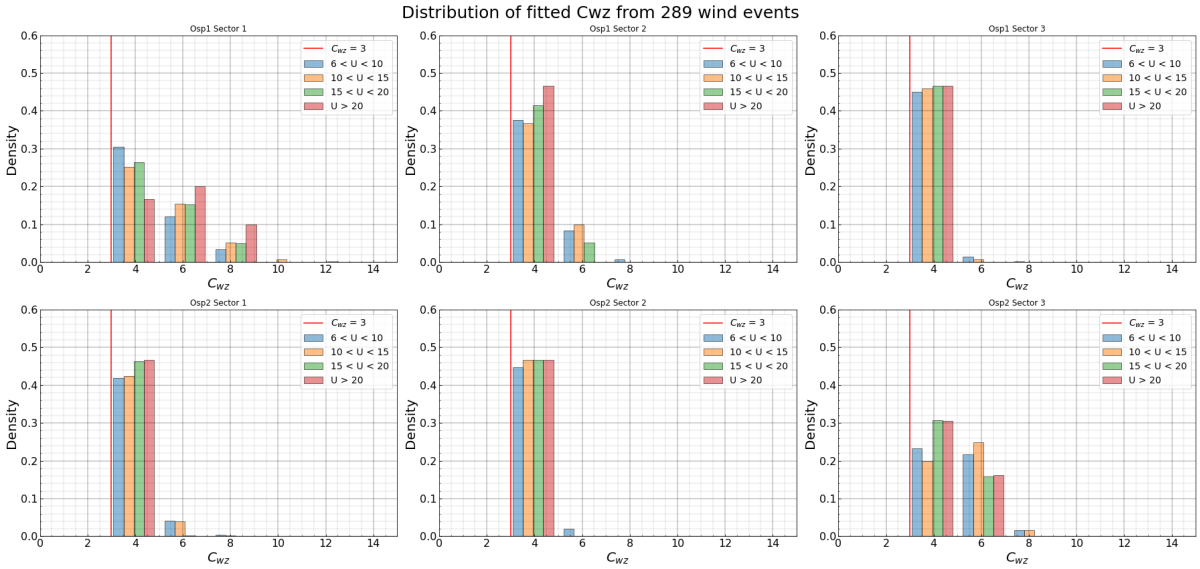

Figure 5.86: Distribution of the fitted C_{wz} with sectors at Osp1 and Osp2.

5.7.2 Distribution with atmospheric stability

Figure 5.87 and Figure 5.88 show the distribution of the fitted C_{uy} under different atmospheric stabilities. Again, the stability does not appear to have much of an effect the distribution of the parameters, therefore only C_{uy} is shown here and the figures for the remaining parameters are given in the appendix. Most of the values for C_{uy} lie under the reference value for all cases, except for southern winds which are affected by changes in terrain elevation.

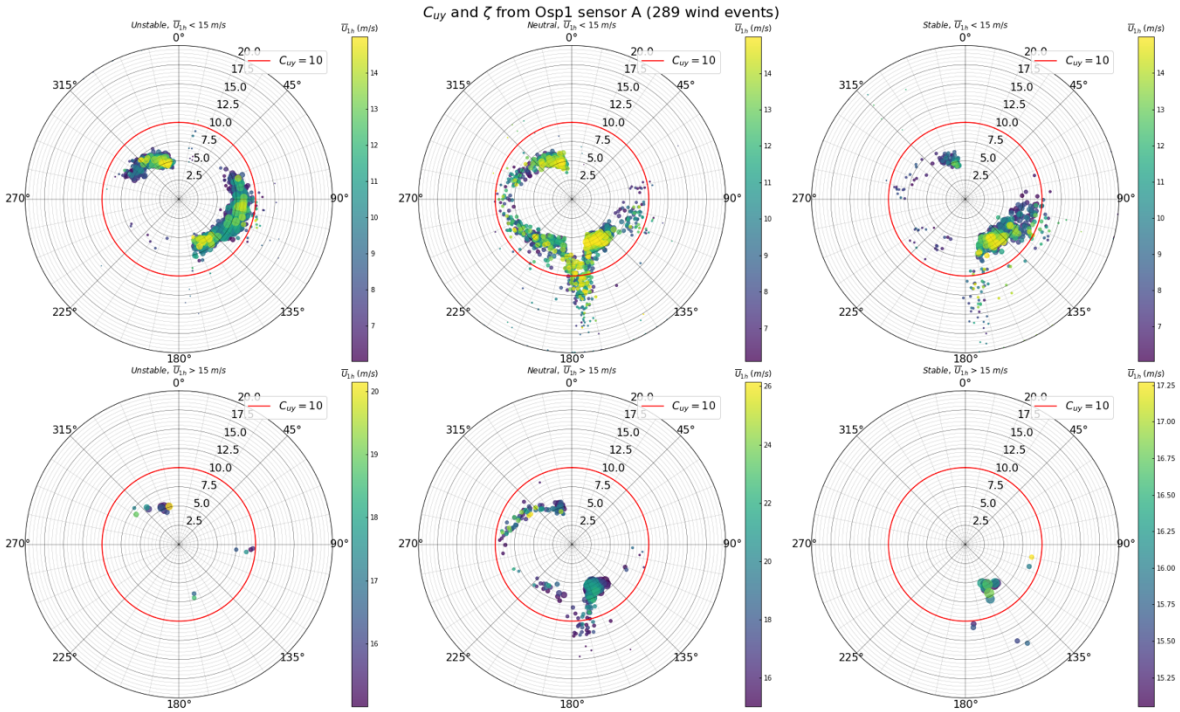

Figure 5.87: Distribution of the fitted C_{uy} with stability at Osp1.

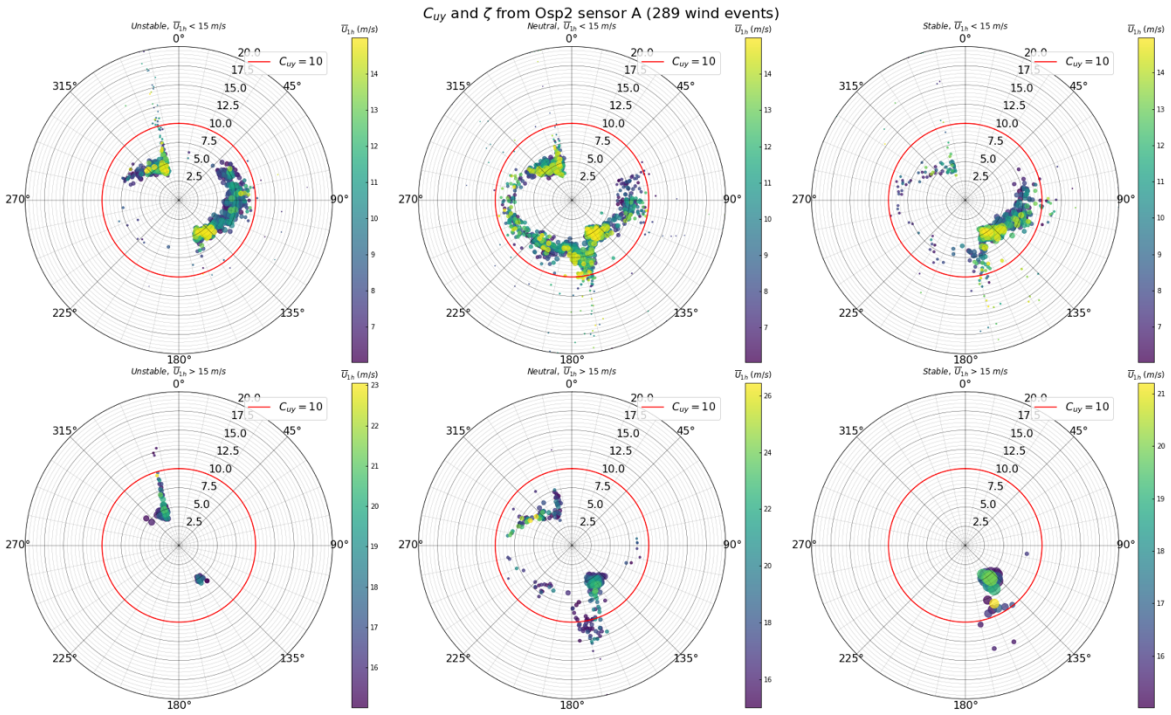

Figure 5.88: Distribution of the fitted C_{uy} with stability at Osp2.

As was done for the fitted A_u in Figure 5.66, Figure 5.89 shows the variation of C_{uy} with stability only in the eastern sector (from 65˚ to 115˚), corresponding to wind flow over water with a long fetch. Unstable conditions are marginally more represented, as was expected, however C_{uy} shows broad scatter regardless of stability condition. A reason for this may be that the sensors used in the fitting of the coherence parameters experience the same stability conditions at the same time, therefore the stability will not have much of an effect on the distribution of the coherence parameter. Looking at this in context with the observations made from Figure 5.66 further verifies this.

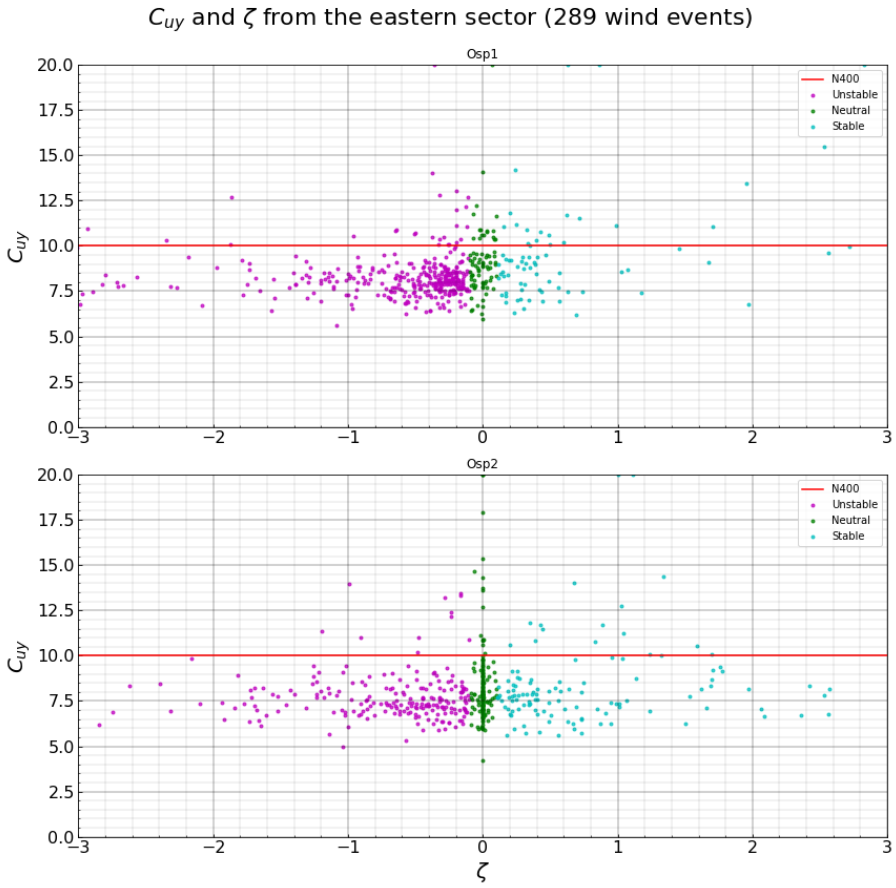

Figure 5.89: Distribution of the fitted C_{uy} with stability in the eastern sector at Osp1 and Osp2.

6. Conclusion

This thesis evaluated the environmental conditions at a Norwegian fjord where a 5 km long floating bridge is planned to be built. This was done based on wind measurements from 2015 to 2020 from four wind masts in the fjord. A threshold of 6 m/s for the mean wind speed was selected to identify wind events, resulting in a total of 289 wind events. The wind characteristics were studied with the variation of mean wind speed, mean wind direction and atmospheric stability, with focus given to the spectral and coherence parameters and comparing them to the N400 code. Non-stationary events were disregarded for the applicability of the theory used in the analysis.

The angle of attack at the four wind masts showed expected results, highlighting the effect of the fjord's complex topography on the wind characteristics. The main wind directions were found to be from north-west, south-west and south-east, corresponding to wind flow over water with long fetch. Results for the turbulence intensity showed that for wind flow over land, it was higher than what was suggested in N400, while the opposite was the case for wind flow over water. In general, N400 underestimated the lateral turbulence intensity regardless of direction, but gave a good representation of the vertical turbulence intensity. As the mean wind speed increases, the turbulence intensity for the longitudinal and vertical directions move closer to what is suggested in N400. The friction velocity was found to be low for wind flow over water and higher for wind flow over land, as expected. Results of the turbulence ratio agreed with observations from earlier research done in Norwegian fjords, with the values ranging between 2 and 4. The turbulence ratio also increases with the mean wind speed. The integral length scales in the longitudinal direction were found to have a very broad scatter, and the suggested values in N400 significantly underestimated the integral length scales. Unlike the turbulence intensity and turbulence ratio, the integral length scales decrease with increasing mean wind speed.

For the spectral analysis, the 2-parameter N400 spectrum showed marginally better results overall in the frequency range relevant for the floating bridge, compared to the 1-parameter N400 spectrum. However, the improvement is very small, and it is found that the 1-parameter model is still very accurate in the relevant frequency range. The fitted spectral parameters for all turbulence components were largest for wind flow over water, indicating more energy content in the low frequency region for these directions. The fitted A_u moves closer to the

reference value in N400 as the mean wind speed increases, however A_v and A_w tend to be higher than the reference regardless of wind speed. Results from the coherence fittings showed that, generally, the fitted C_{iy} parameters were always lower than the reference value in N400 and C_{iz} was always higher, i.e. N400 underestimated the coherence in the horizontal plane and overestimated the coherence in the vertical plane. The results for C_{wy} and C_{wz} imply that the vertical turbulence component is affected by the local topography, with a negative vertical turbulence component associated with a lower coherence.

The atmospheric stability was found to be dominated by near-neutral conditions for the most part, followed closely by unstable and stable conditions. Synnøytangen in particular showed large scatter in the stability with both mean wind speed and direction compared to the other wind masts. As the mean wind speed increases, the neutral conditions occur more frequently. No tilt correction algorithm was used for the vertical turbulence component, so this may have influenced the stability results obtained. Additionally, the amount of data disregarded due to non-stationarity of the wind was substantial, affecting results even further. The distribution of the spectral and coherence parameters with atmospheric stability was difficult to interpret. However, for wind flow from the east, it was found that unstable conditions increased the scatter of A_u and vice versa for stable conditions, while C_{uy} showed no clear dependence on the stability.

For future work, investigating the effect of using a tilt correction algorithm on the vertical turbulence component to better estimate the atmospheric stability is recommended. This may give clearer results on what effect the stability has on the spectral and coherence parameters. Another suggestion is to be stricter with the stationarity criteria for the lower wind speeds, to see if the results obtained here are affected by the non-stationarity. Alternatively, because the current stationarity criteria led to significant amounts of disregarded data, non-stationary analysis may also be considered.

References

[1] Statens Vegvesen. "Ferjefri E39." https://www.vegvesen.no/vegprosjekter/europaveg/ferjefrie39/ (accessed 29.01, 2022).

[2] Statens Vegvesen. "Norway takes on its largest infrastructure project in modern history." https://www.vegvesen.no/vegprosjekter/europaveg/ferjefrie39/nyhetsarkiv/presenterte-ny-bruteknologi-pa-tedx-bologna/norway-takes-on-its-largest-infrastructure-project-in-modern-history/ (accessed 29.01, 2022).

[3] Statens Vegvesen, "MetOcean Specification," 2021.

[4] C. Dyrbye and S. O. Hansen, *Wind Loads on Structures*. Chichester: Wiley, 1997.

[5] J. D. Holmes, *Wind Loading of Structures*, 3. ed. London: Taylor & Francis, 2015.

[6] *Eurocode 1: Actions on structures – Part 1-4: General actions – Wind actions*, European Standard.

[7] R. B. Stull, *An Introduction to Boundary Layer Meteorology*. Dordrecht: Kluwer Academic Publishers, 1988.

[8] Z. Midjiyawa, E. Cheynet, J. Reuder, H. Ágústsson, and T. Kvamsdal, "Potential and challenges of wind measurements using met-masts in complex topography for bridge design: Part I – Integral flow characteristics," *Journal of Wind Engineering & Industrial Aerodynamics,* vol. 211, 2021, doi: https://doi.org/10.1016/j.jweia.2021.104584.

[9] L. Li, Y. Xiao, A. Kareem, L. Song, and P. Qin, "Modeling typhoon wind power spectra near sea surface based on measurements in the South China sea," *Journal of Wind Engineering and Industrial Aerodynamics,* vol. 104-106, pp. 565-576, 2012, doi: https://doi.org/10.1016/j.jweia.2012.04.005.

[10] E. Cheynet, J. B. Jakobsen, and C. Obhrai, "Spectral characteristics of surface-layer turbulence in the North Sea," *Energy Procedia,* vol. 137, pp. 414-427, 2017, doi: https://doi.org/10.1016/j.egypro.2017.10.366.

[11] E. Cheynet, J. B. Jakobsen, B. Svardal, J. Reuder, and V. Kumer, "Wind coherence measurement by a single pulsed Doppler wind lidar," *Energy Procedia,* vol. 94, pp. 462-477, 2016, doi: https://doi.org/10.1016/j.egypro.2016.09.217.

[12] G. Solari, "Turbulence Modeling for Gust Loading," *Journal of Structural Engineering,* vol. 113, no. 7, pp. 1550-1569, 1987, doi: https://doi.org/10.1061/(ASCE)0733-9445(1987)113:7(1550).

[13] J. F. Manwell, J. G. McGowan, and A. L. Rogers, *Wind energy explained : theory, design and application*, 2. ed. Chichester: Wiley, 2009.

[14] E. Cheynet, J. B. Jakobsen, and J. Reuder, "Velocity Spectra and Coherence Estimates in the Marine Atmospheric Boundary Layer," *Boundary-Layer Meteorology,* vol. 169, no. 3, pp. 429–460, 2018, doi: https://doi.org/10.1007/s10546-018-0382-2.

[15] J. C. Kaimal, J. C. Wyngaard, Y. Izumi, O. R. Coté, "Spectral characteristics of surface-layer turbulence," *Quarterly Journal of the Royal Meteorological Society,* vol. 98, no. 417, pp. 563-589, 1972.

[16] *Håndbok N400 Bruprosjektering*, Statens Vegvesen, 2022.

[17] O. J. Andersen and J. Løvseth, "The Frøya database and maritime boundary layer wind description," *Marine Structures,* vol. 19, no. 2-3, pp. 173-192, 2006, doi: https://doi.org/10.1016/j.marstruc.2006.07.003.

[18] Standard Norge, *Actions and action effects*, 2017 + AC:2018. ed. (NORSOK standard). Lysaker: Standard Norge, 2018.

[19] D. E. Newland, *An introduction to random vibrations, spectral & wavelet analysis*, 3. ed. Mineola, N.Y: Dover, 2005.

[20] P. Welch, "The use of fast Fourier transform for the estimation of power spectra: A method based on time averaging over short, modified periodograms," *IEEE Transactions on Audio and Electroacoustics,* vol. 15, no. 2, pp. 70-73, 1967, doi: 10.1109/TAU.1967.1161901.

[21] *WindMaster and WindMaster Pro User Manual*, 1561-PS-0001 Issue 16, Gill Instruments Limited, 2021. [Online]. Available: http://gillinstruments.com/data/WindMaster/Manuals/1561-PS-0001%20WindMaster%20Windmaster%20Pro%20Manual%20Issue%2016.pdf

[22] Kjeller Vindteknikk, "E39, brukrysninger Hordaland," 2019.

[23] "Power spectral density using Welch's method." https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.welch.html (accessed 17.02, 2022).

[24] "Curve fit of a function to data using non-linear least squares method." https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.curve_fit.html?highlight=curve_fit (accessed 17.02, 2022).

[25] "Cross power spectral density using Welch's method." https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.csd.html (accessed 17.02, 2022).

[26] G. Solari and G. Piccardo, "Probabilistic 3-D turbulence modeling for gust buffeting of structures," *Probabilistic Engineering Mechanics,* vol. 16, no. 1, pp. 73-86, 2001, doi: https://doi.org/10.1016/S0266-8920(00)00010-2.

[27] Kartverket. "Norgeskart." https://www.norgeskart.no (accessed 2022).

[28] A. Sathe, J. Mann, T. Barlas, W. A. A. M. Bierbooms, and G. J. W. van Bussel, "Influence of atmospheric stability on wind turbine loads," *Wind Energy,* vol. 16, no. 7, pp. 1013-1032, 2013, doi: https://doi.org/10.1002/we.1528.

[29] A. Fenerci and O. Øiseth, "Strong wind characteristics and dynamic response of a long-span suspension bridge during a storm," *Journal of Wind Engineering & Industrial Aerodynamics,* vol. 172, pp. 116-138, 2018, doi: https://doi.org/10.1016/j.jweia.2017.10.030.

[30] Z. Midjiyawa, E. Cheynet, J. Reuder, H. Ágústsson, and T. Kvamsdal, "Potential and challenges of wind measurements using met-masts in complex topography for bridge design: Part II – Spectral flow characteristics," *Journal of Wind Engineering and Industrial Aerodynamics,* vol. 211, 2021, doi: https://doi.org/10.1016/j.jweia.2021.104585.

Appendix

A: Python scripts

All the Python scripts shown in this section (except A2) are for Osp1 only; in those cases, identical scripts were used for Osp2, Svar and Synn.

A1 Identification of wind events

```
# -*- coding: utf-8 -*-
"""
Created on Thu Sep 10 08:44:42 2020
select the wind stroms based on four masts and reference station
Criteria:
    mast: top sensor (50m, 1 h average) velocity above 19m/s using 'signal.find_peaks'
    ref light house: 10mins velocity above 19m/s

# new matplotlib 3.3.3 count num 1970-01-01T00:00 while the old count from 0001-01-01 00:00:00

@author: junwan
"""

import numpy as np
import scipy.io as spio
import matplotlib
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
import scipy.signal as signal
from   scipy.optimize import curve_fit
from   scipy.optimize import least_squares
import re
import pandas as pd
from   datetime import datetime
from   datetime import timedelta
from scipy.stats import binned_statistic

import matplotlib.dates as mdates
import json
import transformations      as TF

from sklearn import neighbors, datasets
from scipy.signal import argrelextrema

def running_mean(x, N):
    cumsum = np.cumsum(np.insert(x, 0, 0))
    return (cumsum[N:] - cumsum[:-N]) / float(N)
def datenum(d):
    return 366 + d.toordinal() + (d - datetime.fromordinal(d.toordinal())).total_seconds()/(24*60*60)
class NumpyEncoder(json.JSONEncoder):
    def default(self, obj):
        if isinstance(obj, np.ndarray):
            return obj.tolist()
        return json.JSONEncoder.default(self, obj)

# loop over month
def month_year_iter( start_month, start_year, end_month, end_year ):
    ym_start= 12*start_year + start_month - 1
    ym_end= 12*end_year + end_month
    loop=[]
    for ym in range( ym_start, ym_end ):
        loop = np.append(loop,divmod( ym, 12 ))
        #y[ym], m[ym] = divmod( ym, 12 )
    loop=loop.reshape((int(loop.size/2),2))
    return loop

# function to calculate the wind direciton and tranformation matrix from global XYZ to local wind uvw
```

```
# local v axis is conti-clockwise 90 deg with respect to u axis
# gloabl X is pointing east and global Y is pointing north
def windXY2dir(X,Y):
    dir_W  =  np.arctan2(X,Y)
    if dir_W<0:
        dir_W=dir_W+2*np.pi
    G_x = [1,0,0]
    G_y = [0,1,0]
    G_z = [0,0,1]
    S_u     = [np.sin(dir_W),        np.cos(dir_W),        0]
    S_v     = [np.sin(dir_W-np.pi/2),np.cos(dir_W-np.pi/2),0]
    S_w     = [0,0,1]
    # transformation matrix from XYZ to UVW
    TT      = TF.T_xyzXYZ(G_x, G_y, G_z, S_u, S_v, S_w, dim='3x3')
    return (dir_W,TT)

# function to calculate the wind direction, mean wind speed, local wind uvw components and the time
tags
def windXYZ2uvw(data):
    # calculate the non-mean for each component of the wind components in global XYZ within starting
and ending time
    data_temp = np.nanmean(data,0)
    if ~np.isnan(data.any()):
        [dir_w,tt_w] = windXY2dir(data_temp[0],data_temp[1])
        U            = np.matmul(tt_w.T , data_temp)[0]
        uvw          = np.matmul(tt_w.T , data.T).T
        return(dir_w,U,uvw)


# round time to 1 minute
def round_time(dt=None, round_to=60):
   if dt == None:
       dt = datetime.now()
   seconds = (dt - dt.min).seconds
   rounding = (seconds+round_to/2) // round_to * round_to
   return dt + timedelta(0,rounding-seconds,-dt.microsecond)


regex_num  = r"[-+]?[.]?[\d]+(?:\d\d\d)*[\.]?\d*(?:[eE][-+]?\d+)?"

z0=0.01   # roughness lengthscale

#%%load OSP1
cases = pd.read_excel('files_osp1.xls', sheet_name = 'files_osp1')
height= pd.read_excel('files_osp1.xls', sheet_name = 'Height')
# get the ending time tag
time_e = cases['Last timestep'][cases.index[-1]]
temp = np.asarray(re.findall(regex_num,time_e), dtype = 'float64')
year_e =     int(temp[0])
mon_e  =     int(abs(temp[1]))
day_e  =     int(abs(temp[2]))
hour_e =     int(temp[3])
minu_e =     int(temp[4])
sec_e  =     int(np.floor(temp[5]))
msec_e =     int(np.round((temp[5] - np.floor(temp[5]))*1000))
time_e = mdates.date2num( datetime(year_e,mon_e,day_e,hour_e,minu_e,sec_e,msec_e))


# delta_mon is one month time
delta_ms         = timedelta(milliseconds = 100)
delta_10min      = mdates.date2num(datetime(2,1,1,1,10)) - mdates.date2num(datetime(2,1,1,1,0))
delta_1h         = mdates.date2num(datetime(2,1,1,2,0)) - mdates.date2num(datetime(2,1,1,1,0))
delta_mon        = mdates.date2num(datetime(2,2,1,1,0)) - mdates.date2num(datetime(2,1,1,1,0))

date_s_ref = datetime(2015,12,4,0,0,0)
date_e_ref = mdates.num2date(time_e + delta_mon).replace(tzinfo=None)

monthloop  = month_year_iter(date_s_ref.month,date_s_ref.year,date_e_ref.month,date_e_ref.year)

# load all the 1 hour mean values
A_U    = []
A_Dir  = []
Time   = []
B_U    = []
```

```
B_Dir  = []
C_U    = []
C_Dir  = []
for i in range(0,monthloop.shape[0]-1):
    date_start = mdates.date2num(datetime(int(monthloop[i,0]),int(monthloop[i,1])+1,1,0,0))

    if i == 0:
        date_start = mdates.date2num(date_s_ref)

    #create the string of 'date_search '
    t_s = mdates.num2date(date_start)
    year = str(t_s.year)
    mon = str(t_s.month)
    U    =    np.load('D:/DATA/osp1/npz files/' + 'Osp1_' + str(year) + '_' + str(mon) +
'_A_U_1h.npz')['arr_0']
    Dir  =    np.load('D:/DATA/osp1/npz files/' + 'Osp1_' + str(year) + '_' + str(mon) +
'_A_dir_1h.npz')['arr_0']/np.pi*180  # change radians to degrees
    time =    np.load('D:/DATA/osp1/npz files/' + 'Osp1_' + str(year) + '_' + str(mon) +
'_A_time_1h.npz')['arr_0']
    A_U   = np.append(A_U,U)
    A_Dir = np.append(A_Dir,Dir)
    Time  = np.append(Time,time)

    U    =    np.load('D:/DATA/osp1/npz files/' + 'Osp1_' + str(year) + '_' + str(mon) +
'_B_U_1h.npz')['arr_0']
    Dir  =    np.load('D:/DATA/osp1/npz files/' + 'Osp1_' + str(year) + '_' + str(mon) +
'_B_dir_1h.npz')['arr_0']/np.pi*180  # change radians to degrees
    B_U   = np.append(B_U,U)
    B_Dir = np.append(B_Dir,Dir)

    U    =    np.load('D:/DATA/osp1/npz files/' + 'Osp1_' + str(year) + '_' + str(mon) +
'_C_U_1h.npz')['arr_0']
    Dir  =    np.load('D:/DATA/osp1/npz files/' + 'Osp1_' + str(year) + '_' + str(mon) +
'_C_dir_1h.npz')['arr_0']/np.pi*180  # change radians to degrees
    C_U   = np.append(C_U,U)
    C_Dir = np.append(C_Dir,Dir)
A_Dir  = (A_Dir + 180)%360
B_Dir  = (B_Dir + 180)%360
C_Dir  = (C_Dir + 180)%360

# new matplotlib count num 1970-01-01T00:00 while the old count from 0001-01-01 00:00:00
dateo      = datetime(1968, 12, 31, 0, 0, 0)
delta_date = datenum(dateo)

time_date = mdates.num2date(Time - delta_date)

for i in range(len(time_date)):
    time_date[i] = round_time(time_date[i].replace(tzinfo=None), round_to = 60*10)
Time      = mdates.date2num(time_date)
# ================================================================================
# A_U  = A_U*np.log(10/z0)/np.log(height['A'][0]/z0)
# B_U  = B_U*np.log(10/z0)/np.log(height['B'][0]/z0)
# C_U  = C_U*np.log(10/z0)/np.log(height['C'][0]/z0)
# ================================================================================


Osp1 = {'Time':Time, 'A_U':A_U, 'B_U':B_U, 'C_U':C_U, 'A_Dir':A_Dir, 'B_Dir':B_Dir, 'C_Dir':C_Dir}

#%%Load osp2
cases = pd.read_excel('files_osp2.xls', sheet_name = 'files_osp2')
height = pd.read_excel('files_osp2.xls', sheet_name = 'Height')

# get the ending time tag
time_e = cases['Last timestep'][cases.index[-1]]
temp = np.asarray(re.findall(regex_num,time_e), dtype = 'float64')
year_e =     int(temp[0])
mon_e  =     int(abs(temp[1]))
day_e  =     int(abs(temp[2]))
hour_e =     int(temp[3])
minu_e =     int(temp[4])
sec_e  =     int(np.floor(temp[5]))
msec_e =     int(np.round((temp[5] - np.floor(temp[5]))*1000))
time_e = mdates.date2num( datetime(year_e,mon_e,day_e,hour_e,minu_e,sec_e,msec_e))
```

```
# delta_mon is one month time
delta_ms         = timedelta(milliseconds = 100)
delta_10min      = mdates.date2num(datetime(2,1,1,1,10)) - mdates.date2num(datetime(2,1,1,1,0))
delta_1h         = mdates.date2num(datetime(2,1,1,2,0)) - mdates.date2num(datetime(2,1,1,1,0))
delta_mon        = mdates.date2num(datetime(2,2,1,1,0)) - mdates.date2num(datetime(2,1,1,1,0))

date_s_ref = datetime(2015,12,19,0,0,0)
date_e_ref = mdates.num2date(time_e + delta_mon).replace(tzinfo = None)

monthloop  = month_year_iter(date_s_ref.month,date_s_ref.year,date_e_ref.month,date_e_ref.year)

# load all the 1 hour mean values
A_U    = []
A_Dir  = []
Time   = []
B_U    = []
B_Dir  = []
C_U    = []
C_Dir  = []
for i in range(0,monthloop.shape[0]-1):
    date_start = mdates.date2num(datetime(int(monthloop[i,0]),int(monthloop[i,1])+1,1,0,0))

    if i == 0:
        date_start = mdates.date2num(date_s_ref)

    #create the string of 'date_search '
    t_s = mdates.num2date(date_start)
    year = str(t_s.year)
    mon = str(t_s.month)
    U    =    np.load('D:/DATA/osp2/npz files/' + 'Osp2_' + str(year) + '_' + str(mon) +
'_A_U_1h.npz')['arr_0']
    Dir  =    np.load('D:/DATA/osp2/npz files/' + 'Osp2_' + str(year) + '_' + str(mon) +
'_A_dir_1h.npz')['arr_0']/np.pi*180  # change radians to degrees
    time =    np.load('D:/DATA/osp2/npz files/' + 'Osp2_' + str(year) + '_' + str(mon) +
'_A_time_1h.npz')['arr_0']
    A_U   = np.append(A_U,U)
    A_Dir = np.append(A_Dir,Dir)
    Time  = np.append(Time,time)

    U    =    np.load('D:/DATA/osp2/npz files/' + 'Osp2_' + str(year) + '_' + str(mon) +
'_B_U_1h.npz')['arr_0']
    Dir  =    np.load('D:/DATA/osp2/npz files/' + 'Osp2_' + str(year) + '_' + str(mon) +
'_B_dir_1h.npz')['arr_0']/np.pi*180  # change radians to degrees
    B_U   = np.append(B_U,U)
    B_Dir = np.append(B_Dir,Dir)

    U    =    np.load('D:/DATA/osp2/npz files/' + 'Osp2_' + str(year) + '_' + str(mon) +
'_C_U_1h.npz')['arr_0']
    Dir  =    np.load('D:/DATA/osp2/npz files/' + 'Osp2_' + str(year) + '_' + str(mon) +
'_C_dir_1h.npz')['arr_0']/np.pi*180  # change radians to degrees
    C_U   = np.append(C_U,U)
    C_Dir = np.append(C_Dir,Dir)
A_Dir  = (A_Dir + 180)%360
B_Dir  = (B_Dir + 180)%360
C_Dir  = (C_Dir + 180)%360

# new matplotlib count num 1970-01-01T00:00 while the old count from 0001-01-01 00:00:00
dateo      = datetime(1968, 12, 31, 0, 0, 0)
delta_date = datenum(dateo)

time_date = mdates.num2date(Time - delta_date)
for i in range(len(time_date)):
    time_date[i] = round_time(time_date[i].replace(tzinfo=None), round_to = 60*10)
Time      = mdates.date2num(time_date)
# ================================================================================
# A_U  = A_U*np.log(10/z0)/np.log(height['A'][0]/z0)
# B_U  = B_U*np.log(10/z0)/np.log(height['B'][0]/z0)
# C_U  = C_U*np.log(10/z0)/np.log(height['C'][0]/z0)
# ================================================================================

Osp2 = {'Time':Time, 'A_U':A_U, 'B_U':B_U, 'C_U':C_U, 'A_Dir':A_Dir, 'B_Dir':B_Dir, 'C_Dir':C_Dir}

#%%Load Svar
```

```
cases = pd.read_excel('files_svar.xls', sheet_name = 'files_svar')
height = pd.read_excel('files_svar.xls', sheet_name = 'Height')

# get the ending time tag
time_e = cases['Last timestep'][cases.index[-1]]
temp = np.asarray(re.findall(regex_num,time_e), dtype = 'float64')
year_e =     int(temp[0])
mon_e  =     int(abs(temp[1]))
day_e  =     int(abs(temp[2]))
hour_e =     int(temp[3])
minu_e =     int(temp[4])
sec_e  =     int(np.floor(temp[5]))
msec_e =     int(np.round((temp[5] - np.floor(temp[5]))*1000))
time_e = mdates.date2num(datetime(year_e,mon_e,day_e,hour_e,minu_e,sec_e,msec_e))


# delta_mon is one month time
delta_ms         = timedelta(milliseconds = 100)
delta_10min      = mdates.date2num(datetime(2,1,1,1,10)) - mdates.date2num(datetime(2,1,1,1,0))
delta_1h         = mdates.date2num(datetime(2,1,1,2,0)) - mdates.date2num(datetime(2,1,1,1,0))
delta_mon        = mdates.date2num(datetime(2,2,1,1,0)) - mdates.date2num(datetime(2,1,1,1,0))

date_s_ref = datetime(2015,3,18,0,0,0)
date_e_ref = mdates.num2date(time_e + delta_mon).replace(tzinfo = None)

monthloop  = month_year_iter(date_s_ref.month,date_s_ref.year,date_e_ref.month,date_e_ref.year)

# load all the 1 hour mean values
A_U    = []
A_Dir  = []
Time   = []
B_U    = []
B_Dir  = []
C_U    = []
C_Dir  = []
for i in range(0,monthloop.shape[0]-1):
    date_start = mdates.date2num(datetime(int(monthloop[i,0]),int(monthloop[i,1])+1,1,0,0))

    if i == 0:
        date_start = mdates.date2num(date_s_ref)

    #create the string of 'date_search '
    t_s = mdates.num2date(date_start)
    year = str(t_s.year)
    mon = str(t_s.month)
    U    =    np.load('D:/DATA/svar/npz files/' + 'Svar_' + str(year) + '_' + str(mon) +
'_A_U_1h.npz')['arr_0']
    Dir  =    np.load('D:/DATA/svar/npz files/' + 'Svar_' + str(year) + '_' + str(mon) +
'_A_dir_1h.npz')['arr_0']/np.pi*180  # change radians to degrees
    time =    np.load('D:/DATA/svar/npz files/' + 'Svar_' + str(year) + '_' + str(mon) +
'_A_time_1h.npz')['arr_0']
    A_U   = np.append(A_U,U)
    A_Dir = np.append(A_Dir,Dir)
    Time  = np.append(Time,time)

    U    =    np.load('D:/DATA/svar/npz files/' + 'Svar_' + str(year) + '_' + str(mon) +
'_B_U_1h.npz')['arr_0']
    Dir  =    np.load('D:/DATA/svar/npz files/' + 'Svar_' + str(year) + '_' + str(mon) +
'_B_dir_1h.npz')['arr_0']/np.pi*180  # change radians to degrees
    B_U   = np.append(B_U,U)
    B_Dir = np.append(B_Dir,Dir)

    U    =    np.load('D:/DATA/svar/npz files/' + 'Svar_' + str(year) + '_' + str(mon) +
'_C_U_1h.npz')['arr_0']
    Dir  =    np.load('D:/DATA/svar/npz files/' + 'Svar_' + str(year) + '_' + str(mon) +
'_C_dir_1h.npz')['arr_0']/np.pi*180  # change radians to degrees
    C_U   = np.append(C_U,U)
    C_Dir = np.append(C_Dir,Dir)
A_Dir  = (A_Dir + 180)%360
B_Dir  = (B_Dir + 180)%360
C_Dir  = (C_Dir + 180)%360

# new matplotlib count num 1970-01-01T00:00 while the old count from 0001-01-01 00:00:00
dateo      = datetime(1968, 12, 31, 0, 0, 0)
```

```
delta_date = datenum(dateo)

time_date = mdates.num2date(Time - delta_date)
for i in range(len(time_date)):
    time_date[i] = round_time(time_date[i].replace(tzinfo = None), round_to = 60*10)
Time      = mdates.date2num(time_date)
# ========================================================================
# A_U  = A_U*np.log(10/z0)/np.log(height['A'][0]/z0)
# B_U  = B_U*np.log(10/z0)/np.log(height['B'][0]/z0)
# C_U  = C_U*np.log(10/z0)/np.log(height['C'][0]/z0)
# ========================================================================

Svar = {'Time':Time, 'A_U':A_U, 'B_U':B_U, 'C_U':C_U, 'A_Dir':A_Dir, 'B_Dir':B_Dir, 'C_Dir':C_Dir}
#%%Load Synn
cases = pd.read_excel('files_synn.xls', sheet_name = 'files_synn')
height= pd.read_excel('files_synn.xls', sheet_name = 'Height')

# get the ending time tag
time_e = cases['Last timestep'][cases.index[-1]]
temp = np.asarray(re.findall(regex_num,time_e), dtype = 'float64')
year_e =     int(temp[0])
mon_e  =     int(abs(temp[1]))
day_e  =     int(abs(temp[2]))
hour_e =     int(temp[3])
minu_e =     int(temp[4])
sec_e  =     int(np.floor(temp[5]))
msec_e =     int(np.round((temp[5] - np.floor(temp[5]))*1000))
time_e = mdates.date2num(datetime(year_e,mon_e,day_e,hour_e,minu_e,sec_e,msec_e))


# delta_mon is one month time
delta_ms         = timedelta(milliseconds = 100)
delta_10min      = mdates.date2num(datetime(2,1,1,1,10)) - mdates.date2num(datetime(2,1,1,1,0))
delta_1h         = mdates.date2num(datetime(2,1,1,2,0)) - mdates.date2num(datetime(2,1,1,1,0))
delta_mon        = mdates.date2num(datetime(2,2,1,1,0)) - mdates.date2num(datetime(2,1,1,1,0))

date_s_ref = datetime(2015,2,23,0,0,0)
date_e_ref = mdates.num2date(time_e + delta_mon).replace(tzinfo = None)

monthloop  = month_year_iter(date_s_ref.month,date_s_ref.year,date_e_ref.month,date_e_ref.year)

# load all the 1 hour mean values
A_U    = []
A_Dir  = []
Time   = []
B_U    = []
B_Dir  = []
C_U    = []
C_Dir  = []
for i in range(0,monthloop.shape[0]-1):
    date_start = mdates.date2num(datetime(int(monthloop[i,0]),int(monthloop[i,1])+1,1,0,0))

    if i == 0:
        date_start = mdates.date2num(date_s_ref)

    #create the string of 'date_search '
    t_s = mdates.num2date(date_start)
    year = str(t_s.year)
    mon = str(t_s.month)
    U    =    np.load('D:/DATA/synn/npz files/' + 'Synn_' + str(year) + '_' + str(mon) +
'_A_U_1h.npz')['arr_0']
    Dir  =    np.load('D:/DATA/synn/npz files/' + 'Synn_' + str(year) + '_' + str(mon) +
'_A_dir_1h.npz')['arr_0']/np.pi*180  # change radians to degrees
    time =    np.load('D:/DATA/synn/npz files/' + 'Synn_' + str(year) + '_' + str(mon) +
'_A_time_1h.npz')['arr_0']
    A_U   = np.append(A_U,U)
    A_Dir = np.append(A_Dir,Dir)
    Time  = np.append(Time,time)

    U    =    np.load('D:/DATA/synn/npz files/' + 'Synn_' + str(year) + '_' + str(mon) +
'_B_U_1h.npz')['arr_0']
    Dir  =    np.load('D:/DATA/synn/npz files/' + 'Synn_' + str(year) + '_' + str(mon) +
'_B_dir_1h.npz')['arr_0']/np.pi*180  # change radians to degrees
    B_U   = np.append(B_U,U)
```

```
    B_Dir = np.append(B_Dir,Dir)

    U    =    np.load('D:/DATA/synn/npz files/' + 'Synn_' + str(year) + '_' + str(mon) +
'_C_U_1h.npz')['arr_0']
    Dir  =    np.load('D:/DATA/synn/npz files/' + 'Synn_' + str(year) + '_' + str(mon) +
'_C_dir_1h.npz')['arr_0']/np.pi*180  # change radians to degrees
    C_U   = np.append(C_U,U)
    C_Dir = np.append(C_Dir,Dir)
A_Dir  = (A_Dir + 180)%360
B_Dir  = (B_Dir + 180)%360
C_Dir  = (C_Dir + 180)%360

# new matplotlib count num 1970-01-01T00:00 while the old count from 0001-01-01 00:00:00
dateo      = datetime(1968, 12, 31, 0, 0, 0)
delta_date = datenum(dateo)

time_date = mdates.num2date(Time - delta_date)
for i in range(len(time_date)):
    time_date[i] = round_time(time_date[i].replace(tzinfo = None), round_to = 60*10)
Time      = mdates.date2num(time_date)
# ============================================================================
# A_U  = A_U*np.log(10/z0)/np.log(height['A'][0]/z0)
# B_U  = B_U*np.log(10/z0)/np.log(height['B'][0]/z0)
# C_U  = C_U*np.log(10/z0)/np.log(height['C'][0]/z0)
# ============================================================================

Synn = {'Time':Time, 'A_U':A_U, 'B_U':B_U, 'C_U':C_U, 'A_Dir':A_Dir, 'B_Dir':B_Dir, 'C_Dir':C_Dir}
#%%Load ref station data

data = pd.read_excel('SLÅTTERØY FYR.xlsx', sheet_name = '48330')

#%%
# find the peak above height and with distance of at least 48 hours
Osp1_A_p,_  = signal.find_peaks(Osp1['A_U'],height = 6,distance = 24)
Osp2_A_p,_  = signal.find_peaks(Osp2['A_U'],height = 6,distance = 24)
#Osp2_A_p,_  = signal.find_peaks(Osp2['B_U'],height=19,distance=2*24)

Svar_A_p,_  = signal.find_peaks(Svar['A_U'],height = 6,distance=24)
Synn_A_p,_  = signal.find_peaks(Synn['A_U'],height = 6,distance=24)
Ref_p,_     = signal.find_peaks(data['Ws'],height = 6,distance=24)


# new matplotlib count num 1970-01-01T00:00 while the old count from 0001-01-01 00:00:00
dateo      = datetime(1968, 12, 31, 0, 0, 0)
delta_date = datenum(dateo)

#get the time tag of each selected storm
Time_Osp1_A_storm  = mdates.num2date(Osp1['Time'][Osp1_A_p])
Time_Osp2_A_storm  = mdates.num2date(Osp2['Time'][Osp2_A_p])
Time_Svar_A_storm  = mdates.num2date(Svar['Time'][Svar_A_p])
Time_Synn_A_storm  = mdates.num2date(Synn['Time'][Synn_A_p])


Time_Ref_storm     = mdates.num2date(mdates.date2num(pd.to_datetime(data['Date'][Ref_p])))
Time_Ref           = mdates.num2date(mdates.date2num(pd.to_datetime(data['Date'])))

# summarzie the all the time tags of storms
Time_storm = mdates.date2num(Time_Osp1_A_storm + Time_Osp2_A_storm + Time_Svar_A_storm +
Time_Synn_A_storm + Time_Ref_storm)
Time_storm_sorted = mdates.date2num(mdates.num2date((np.sort(pd.Series(Time_storm).unique())),tz =
None))


data_s  = pd.DataFrame(columns=['Time_storm', 'Osp1A_U', 'Osp1A_Dir','Osp1B_U', 'Osp1B_Dir','Osp1C_U',
'Osp1C_Dir',\
                                            'Osp2A_U', 'Osp2A_Dir','Osp2B_U', 'Osp2B_Dir','Osp2C_U',
'Osp2C_Dir',\
                                            'SvarA_U', 'SvarA_Dir','SvarB_U', 'SvarB_Dir','SvarC_U',
'SvarC_Dir',\
                                            'SynnA_U', 'SynnA_Dir','SynnB_U', 'SynnB_Dir','SynnC_U',
'SynnC_Dir',\
                                            'Ref_U', 'Ref_Dir'])

data_s['Time_storm'] = mdates.num2date(Time_storm_sorted)
```

```
# collect all the data in the selected storm events
# find index in Osp1 correspnds to common storm
idx_osp1 = np.where(np.in1d(Osp1['Time'], Time_storm_sorted))[0]
idx      = np.where(np.in1d(Time_storm_sorted, Osp1['Time'][idx_osp1]))[0]
data_s.loc[idx,'Osp1A_U']   = Osp1['A_U'][idx_osp1]
data_s.loc[idx,'Osp1A_Dir'] = Osp1['A_Dir'][idx_osp1]
data_s.loc[idx,'Osp1B_U']   = Osp1['B_U'][idx_osp1]
data_s.loc[idx,'Osp1B_Dir'] = Osp1['B_Dir'][idx_osp1]
data_s.loc[idx,'Osp1C_U']   = Osp1['C_U'][idx_osp1]
data_s.loc[idx,'Osp1C_Dir'] = Osp1['C_Dir'][idx_osp1]

idx_osp2 = np.where(np.in1d(Osp2['Time'], Time_storm_sorted))[0]
idx      = np.where(np.in1d(Time_storm_sorted, Osp2['Time'][idx_osp2]))[0]
data_s.loc[idx,'Osp2A_U']   = Osp2['A_U'][idx_osp2]
data_s.loc[idx,'Osp2A_Dir'] = Osp2['A_Dir'][idx_osp2]
data_s.loc[idx,'Osp2B_U']   = Osp2['B_U'][idx_osp2]
data_s.loc[idx,'Osp2B_Dir'] = Osp2['B_Dir'][idx_osp2]
data_s.loc[idx,'Osp2C_U']   = Osp2['C_U'][idx_osp2]
data_s.loc[idx,'Osp2C_Dir'] = Osp2['C_Dir'][idx_osp2]

idx_svar = np.where(np.in1d(Svar['Time'], Time_storm_sorted))[0]
idx      = np.where(np.in1d(Time_storm_sorted, Svar['Time'][idx_svar]))[0]
data_s.loc[idx,'SvarA_U']   = Svar['A_U'][idx_svar]
data_s.loc[idx,'SvarA_Dir'] = Svar['A_Dir'][idx_svar]
data_s.loc[idx,'SvarB_U']   = Svar['B_U'][idx_svar]
data_s.loc[idx,'SvarB_Dir'] = Svar['B_Dir'][idx_svar]
data_s.loc[idx,'SvarC_U']   = Svar['C_U'][idx_svar]
data_s.loc[idx,'SvarC_Dir'] = Svar['C_Dir'][idx_svar]

idx_synn = np.where(np.in1d(Synn['Time'], Time_storm_sorted))[0]
idx      = np.where(np.in1d(Time_storm_sorted, Synn['Time'][idx_synn]))[0]
data_s.loc[idx,'SynnA_U']   = Synn['A_U'][idx_synn]
data_s.loc[idx,'SynnA_Dir'] = Synn['A_Dir'][idx_synn]
data_s.loc[idx,'SynnB_U']   = Synn['B_U'][idx_synn]
data_s.loc[idx,'SynnB_Dir'] = Synn['B_Dir'][idx_synn]
data_s.loc[idx,'SynnC_U']   = Synn['C_U'][idx_synn]
data_s.loc[idx,'SynnC_Dir'] = Synn['C_Dir'][idx_synn]

idx_ref = np.where(np.in1d(mdates.date2num(pd.to_datetime(data['Date'])), Time_storm_sorted))[0]
idx     = np.where(np.in1d(Time_storm_sorted,
mdates.date2num(pd.to_datetime(data['Date'][idx_ref]))))[0]
data_s.loc[idx,'Ref_U']      = data.loc[idx_ref,'Ws'].values
data_s.loc[idx,'Ref_Dir']    = data.loc[idx_ref,'Wd'].values


#%%
# find the date clusters using KernelDensity function defining bandwidth
temp = np.asarray(Time_storm).reshape(-1, 1)
kde = neighbors.KernelDensity(kernel = 'gaussian', bandwidth = 1).fit(temp)
s = np.linspace(temp[0]-10,temp[-1] + 10,100000)
e = kde.score_samples(s.reshape(-1,1))
plt.plot(s, e)

mi, ma = argrelextrema(e, np.less)[0], argrelextrema(e, np.greater)[0]

mdates.num2date(temp[temp < s[mi[0]]])

print(mdates.num2date(temp[(temp < s[mi[0]])] ))
print(' ')
for i in range(mi.size - 1) :
    print(mdates.num2date(temp[(temp >= s[mi[i]]) * (temp <= s[mi[i + 1]])]))
    print(' ')

# get the time tag for each storm event where Osp1 has largest U (rounded to integer hour)
time_ss = []
time_ss_ref = []
for i in range(mi.size - 1) :
# ================================================================================
#     time_ss.append(mdates.num2date(np.mean(temp[(temp >= s[mi[i]]) * (temp <=
s[mi[i+1]])])).replace(microsecond=0, second=0, minute=30) )
#     time_ss_ref.append(mdates.num2date(np.mean(temp[(temp >= s[mi[i]]) * (temp <=
s[mi[i+1]])])).replace(microsecond=0, second=0, minute=00) )
# ================================================================================
```

```
    # get all time tags in one cluster, its index in Osp1, and U, get the time tag with largest U
    ttt       =  temp[(temp >= s[mi[i]]) * (temp <= s[mi[i + 1]])]
    ttt_d     =  np.zeros(ttt.shape)
    idx_ttt   =  np.zeros(ttt.shape)
    U_ttt     =  np.zeros(ttt.shape)

    for ii in range(0,ttt.size):
        ttt_d[ii]   = mdates.date2num(mdates.num2date(ttt[ii]).replace(microsecond = 0, second = 0,
minute = 30))
        if np.size(np.where(np.in1d(Osp1['Time'], ttt_d[ii]))) == 0:
            U_ttt[ii]   = 50
        else:
            idx_ttt[ii] = np.where(np.in1d(Osp1['Time'], ttt_d[ii]))[0]
            U_ttt[ii]   = Osp1['A_U'][int(idx_ttt[ii])]

    idx_Uttt     =  np.where(U_ttt == np.max(U_ttt))[0][0]

    time_ss.append(mdates.num2date(ttt_d[idx_Uttt]).replace(microsecond = 0, second = 0, minute = 30))
    time_ss_ref.append(mdates.num2date(ttt_d[idx_Uttt]).replace(microsecond = 0, second = 0, minute =
00))


# get the averaged time tag for each storm event (rounded to integer hour)
data_ss  = pd.DataFrame(columns=['Time_storm', 'Osp1A_U', 'Osp1A_Dir','Osp1B_U',
'Osp1B_Dir','Osp1C_U', 'Osp1C_Dir',\
                                          'Osp2A_U', 'Osp2A_Dir','Osp2B_U', 'Osp2B_Dir','Osp2C_U',
'Osp2C_Dir',\
                                          'SvarA_U', 'SvarA_Dir','SvarB_U', 'SvarB_Dir','SvarC_U',
'SvarC_Dir',\
                                          'SynnA_U', 'SynnA_Dir','SynnB_U', 'SynnB_Dir','SynnC_U',
'SynnC_Dir',\
                                          'Ref_U', 'Ref_Dir'])
data_ss['Time_storm'] = time_ss

# create a pandas series to find the index for each mast
Time_ss    = mdates.date2num(time_ss)
idx_osp1   = np.where(np.in1d(Osp1['Time'], Time_ss))[0]
idx        = np.where(np.in1d(Time_ss, Osp1['Time'][idx_osp1]))[0]
data_ss.loc[idx,'Osp1A_U']   = Osp1['A_U'][idx_osp1]
data_ss.loc[idx,'Osp1A_Dir'] = Osp1['A_Dir'][idx_osp1]
data_ss.loc[idx,'Osp1B_U']   = Osp1['B_U'][idx_osp1]
data_ss.loc[idx,'Osp1B_Dir'] = Osp1['B_Dir'][idx_osp1]
data_ss.loc[idx,'Osp1C_U']   = Osp1['C_U'][idx_osp1]
data_ss.loc[idx,'Osp1C_Dir'] = Osp1['C_Dir'][idx_osp1]

idx_osp2   = np.where(np.in1d(Osp2['Time'], Time_ss))[0]
idx        = np.where(np.in1d(Time_ss, Osp2['Time'][idx_osp2]))[0]
data_ss.loc[idx,'Osp2A_U']   = Osp2['A_U'][idx_osp2]
data_ss.loc[idx,'Osp2A_Dir'] = Osp2['A_Dir'][idx_osp2]
data_ss.loc[idx,'Osp2B_U']   = Osp2['B_U'][idx_osp2]
data_ss.loc[idx,'Osp2B_Dir'] = Osp2['B_Dir'][idx_osp2]
data_ss.loc[idx,'Osp2C_U']   = Osp2['C_U'][idx_osp2]
data_ss.loc[idx,'Osp2C_Dir'] = Osp2['C_Dir'][idx_osp2]

idx_svar   = np.where(np.in1d(Svar['Time'], Time_ss))[0]
idx        = np.where(np.in1d(Time_ss, Svar['Time'][idx_svar]))[0]
data_ss.loc[idx,'SvarA_U']   = Svar['A_U'][idx_svar]
data_ss.loc[idx,'SvarA_Dir'] = Svar['A_Dir'][idx_svar]
data_ss.loc[idx,'SvarB_U']   = Svar['B_U'][idx_svar]
data_ss.loc[idx,'SvarB_Dir'] = Svar['B_Dir'][idx_svar]
data_ss.loc[idx,'SvarC_U']   = Svar['C_U'][idx_svar]
data_ss.loc[idx,'SvarC_Dir'] = Svar['C_Dir'][idx_svar]

idx_synn   = np.where(np.in1d(Synn['Time'], Time_ss))[0]
idx        = np.where(np.in1d(Time_ss, Synn['Time'][idx_synn]))[0]
data_ss.loc[idx,'SynnA_U']   = Synn['A_U'][idx_synn]
data_ss.loc[idx,'SynnA_Dir'] = Synn['A_Dir'][idx_synn]
data_ss.loc[idx,'SynnB_U']   = Synn['B_U'][idx_synn]
data_ss.loc[idx,'SynnB_Dir'] = Synn['B_Dir'][idx_synn]
data_ss.loc[idx,'SynnC_U']   = Synn['C_U'][idx_synn]
data_ss.loc[idx,'SynnC_Dir'] = Synn['C_Dir'][idx_synn]

Time_ss    = mdates.date2num(time_ss_ref)
```

```
idx_ref    = np.where(np.in1d(mdates.date2num(pd.to_datetime(data['Date'])), Time_ss))[0]
idx        = np.where(np.in1d(Time_ss, mdates.date2num(pd.to_datetime(data['Date'][idx_ref]))))[0]
data_ss.loc[idx,'Ref_U']      = data.loc[idx_ref,'Ws'].values
data_ss.loc[idx,'Ref_Dir']    = data.loc[idx_ref,'Wd'].values

# Export storm events to Excel
data_ss['Time_storm'] = data_ss['Time_storm'].dt.tz_localize(None) # remove timezone
data_ss.to_excel('D:/DATA/Storm events_new.xlsx', sheet_name = 'Clustered wind storms sort')

#%%

plt.close("all")
fig     = plt.figure(figsize=(20, 12))
ax1     = plt.subplot(221)
ax1.scatter(data_ss['Ref_Dir'], data_ss['Osp1A_Dir'], marker='o',alpha=0.5)
ax1.plot([0,360],[0,360],'k-')
ax1.set_xlim(0, 360)
ax1.set_ylim(0, 360)
plt.xlabel(r'$Dir\_{Ref}$ $( ^o)$', fontsize=20)
plt.ylabel(r'$Dir\_{Osp1\_A}$ $( ^o)$', fontsize=20)
plt.rc('xtick', direction='in', color='k')
plt.rc('ytick', direction='in', color='k')
ax1.set_title('', fontsize=25)
g1 = plt.grid(b=True, which='major', color='k', linestyle='-', linewidth=0.35)
g2 = plt.grid(b=True, which='minor', color='k', linestyle='-', linewidth=0.1)
ax1.tick_params(axis='both', labelsize=16)
fig.suptitle('Wind direction correlation between reference station and top sensors at Bjørnafjorden',
fontsize=25)
plt.minorticks_on()
# plt.show()

ax1     = plt.subplot(222)
ax1.scatter(data_ss['Ref_Dir'], data_ss['Osp2A_Dir'], marker='o',alpha=0.5)
ax1.plot([0,360],[0,360],'k-')
ax1.set_xlim(0, 360)
ax1.set_ylim(0, 360)
plt.xlabel(r'$Dir\_{Ref}$ $( ^o)$', fontsize=20)
plt.ylabel(r'$Dir\_{Osp2\_A}$ $( ^o)$', fontsize=20)
plt.rc('xtick', direction='in', color='k')
plt.rc('ytick', direction='in', color='k')
g1 = plt.grid(b=True, which='major', color='k', linestyle='-', linewidth=0.35)
g2 = plt.grid(b=True, which='minor', color='k', linestyle='-', linewidth=0.1)
ax1.tick_params(axis='both', labelsize=16)
plt.minorticks_on()
# plt.show()

ax1     = plt.subplot(223)
ax1.scatter(data_ss['Ref_Dir'], data_ss['SvarA_Dir'], marker='o',alpha=0.5)
ax1.plot([0,360],[0,360],'k-')
ax1.set_xlim(0, 360)
ax1.set_ylim(0, 360)
plt.xlabel(r'$Dir\_{Ref}$ $( ^o)$', fontsize=20)
plt.ylabel(r'$Dir\_{Svar\_A}$ $( ^o)$', fontsize=20)
plt.rc('xtick', direction='in', color='k')
plt.rc('ytick', direction='in', color='k')
g1 = plt.grid(b=True, which='major', color='k', linestyle='-', linewidth=0.35)
g2 = plt.grid(b=True, which='minor', color='k', linestyle='-', linewidth=0.1)
ax1.tick_params(axis='both', labelsize=16)
plt.minorticks_on()
# plt.show()

ax1     = plt.subplot(224)
ax1.scatter(data_ss['Ref_Dir'], data_ss['SynnA_Dir'], marker='o',alpha=0.5)
ax1.plot([0,360],[0,360],'k-')
ax1.set_xlim(0, 360)
ax1.set_ylim(0, 360)
plt.xlabel(r'$Dir\_{Ref}$ $( ^o)$', fontsize=20)
plt.ylabel(r'$Dir\_{Synn\_A}$ $( ^o)$', fontsize=20)
plt.rc('xtick', direction='in', color='k')
plt.rc('ytick', direction='in', color='k')
g1 = plt.grid(b=True, which='major', color='k', linestyle='-', linewidth=0.35)
g2 = plt.grid(b=True, which='minor', color='k', linestyle='-', linewidth=0.1)
ax1.tick_params(axis='both', labelsize=16)
plt.minorticks_on()
```

```
plt.show()

fig.tight_layout(rect=[0, 0, 1, 1])
save_tite = 'Wind direction correlation between reference station and top sensors at
Bjørnafjorden.png'
fig.savefig(save_tite)

#%% plot the comparison at OSP1


time_s = datetime(2019,1,1,0,0,0)
time_e = datetime(2019,1,1,14,0,0)
time_s = mdates.num2date(Time[0])
time_e = mdates.num2date(Time[-1])

plt.close("all")
fig     = plt.figure(figsize=(20, 12))
ax1     = plt.subplot(211)
# format the ticks
locator = mdates.AutoDateLocator(minticks=7, maxticks=9)
formatter = mdates.ConciseDateFormatter(locator)
ax1.xaxis.set_major_locator(locator)
ax1.xaxis.set_major_formatter(formatter)
ax1.plot(mdates.num2date(Osp1['Time']), Osp1['A_Dir'], 'k-',label='$Dir_{wi}$
$(Osp1\_A)$',markeredgecolor='k',markersize=8)
ax1.plot(Time_Osp1_A_storm, Osp1['A_Dir'][Osp1_A_p],
'r*',label='Storm',markeredgecolor='r',markersize=12)
datemin = np.datetime64(time_s, 'm')
datemax = np.datetime64(time_e, 'm') + np.timedelta64(1, 'm')
ax1.set_xlim(datemin, datemax)
plt.ylabel(r'$Dir_{wi}$ $( ^o)$', fontsize=20)
plt.rc('xtick', direction='in', color='k')
plt.rc('ytick', direction='in', color='k')
plt.legend(loc='best',ncol=1,fontsize=16)
g1 = plt.grid(b=True, which='major', color='k', linestyle='-', linewidth=0.35)
g2 = plt.grid(b=True, which='minor', color='k', linestyle='-', linewidth=0.1)
ax1.tick_params(axis='both', labelsize=16)
plt.minorticks_on()
# plt.show()


ax2     = plt.subplot(212)
# format the ticks
locator = mdates.AutoDateLocator(minticks=7, maxticks=9)
formatter = mdates.ConciseDateFormatter(locator)
ax2.xaxis.set_major_locator(locator)
ax2.xaxis.set_major_formatter(formatter)
ax2.plot(mdates.num2date(Osp1['Time']), Osp1['A_U'], 'k-',label='$\overline{U} $
$(Osp1\_A)$',markeredgecolor='k',markersize=8)
ax2.plot(Time_Osp1_A_storm, Osp1['A_U'][Osp1_A_p],
'r*',label='Storm',markeredgecolor='r',markersize=12)

datemin = np.datetime64(time_s, 'm')
datemax = np.datetime64(time_e, 'm') + np.timedelta64(1, 'm')
ax2.set_xlim(datemin, datemax)
plt.ylabel(r'$\overline{U} $ (m s$^{-1})$', fontsize=20)
ax2.set_title('', fontsize=25)
plt.rc('xtick', direction='in', color='k')
plt.rc('ytick', direction='in', color='k')
plt.legend(loc='best',ncol=1,fontsize=16)
g1 = plt.grid(b=True, which='major', color='k', linestyle='-', linewidth=0.35)
g2 = plt.grid(b=True, which='minor', color='k', linestyle='-', linewidth=0.1)
ax2.tick_params(axis='both', labelsize=16)
plt.minorticks_on()
plt.show()


#%% plot OSP1 with Ref station

time_s = datetime(2019,1,1,0,0,0)
time_e = datetime(2019,1,1,14,0,0)
time_s = mdates.num2date(Time[0])
time_e = mdates.num2date(Time[-1])
```

```
plt.close("all")
fig     = plt.figure(figsize=(20, 12))
ax1     = plt.subplot(211)
# format the ticks
locator = mdates.AutoDateLocator(minticks=7, maxticks=9)
formatter = mdates.ConciseDateFormatter(locator)
ax1.xaxis.set_major_locator(locator)
ax1.xaxis.set_major_formatter(formatter)
#ax1.plot(mdates.num2date(Osp1['Time']), Osp1['A_Dir'], 'k-',label='$Dir_{wi}$
$(Osp1\_A)$',markeredgecolor='k',markersize=8)
ax1.plot(Time_Osp1_A_storm, Osp1['A_Dir'][Osp1_A_p],  'r*',label='Storm
(Osp1)',markeredgecolor='r',markersize=12)
#ax1.plot(Time_Ref, data['Wd'],  'b-',label='$Dir_{wi}$
$(Ref)$',markeredgecolor='g',markersize=12,alpha=0.6)
ax1.plot(Time_Ref_storm, data['Wd'][Ref_p],  'g*',label='Storm
(Ref)',markeredgecolor='g',markersize=12,alpha=0.6)
datemin = np.datetime64(time_s, 'm')
datemax = np.datetime64(time_e, 'm') + np.timedelta64(1, 'm')
ax1.set_xlim(time_s, time_e)
ax1.set_ylim(0, 360)
ax1.set_title('Comparison of mean wind direction and velocity between Osp1$\_A$ and Slåtterøy Fyr',
fontsize=25)
plt.ylabel(r'$Dir_{wi}$ $( ^o)$', fontsize=20)
plt.rc('xtick', direction='in', color='k')
plt.rc('ytick', direction='in', color='k')
plt.legend(loc='best',ncol=2,fontsize=16)
g1 = plt.grid(b=True, which='major', color='k', linestyle='-', linewidth=0.35)
g2 = plt.grid(b=True, which='minor', color='k', linestyle='-', linewidth=0.1)
ax1.tick_params(axis='both', labelsize=16)
plt.minorticks_on()
# plt.show()


ax2     = plt.subplot(212)
# format the ticks
locator = mdates.AutoDateLocator(minticks=7, maxticks=9)
formatter = mdates.ConciseDateFormatter(locator)
ax2.xaxis.set_major_locator(locator)
ax2.xaxis.set_major_formatter(formatter)
ax2.plot(mdates.num2date(Osp1['Time']), Osp1['A_U'], 'k-',label=r'$\overline{U} $
$(Osp1\_A)$',markeredgecolor='k',markersize=8)
ax2.plot(Time_Osp1_A_storm, Osp1['A_U'][Osp1_A_p],  'r*',label='Storm
(Osp1)',markeredgecolor='r',markersize=12)
ax2.plot(Time_Ref, data['Ws'],  'b-',label=r'$\overline{U} $
$(Ref)$',markeredgecolor='g',markersize=12,alpha=0.5)
ax2.plot(Time_Ref_storm, data['Ws'][Ref_p],  'g*',label='Storm
(Ref)',markeredgecolor='g',markersize=12,alpha=0.6)
datemin = np.datetime64(time_s, 'm')
datemax = np.datetime64(time_e, 'm') + np.timedelta64(1, 'm')
ax2.set_xlim(datemin, datemax)
plt.ylabel(r'$\overline{U} $ (m s$^{-1})$', fontsize=20)
ax2.set_title('', fontsize=25)
plt.rc('xtick', direction='in', color='k')
plt.rc('ytick', direction='in', color='k')
plt.legend(loc='best',ncol=2,fontsize=16)
g1 = plt.grid(b=True, which='major', color='k', linestyle='-', linewidth=0.35)
g2 = plt.grid(b=True, which='minor', color='k', linestyle='-', linewidth=0.1)
ax2.tick_params(axis='both', labelsize=16)
plt.minorticks_on()
plt.show()
fig.tight_layout(rect=[0, 0, 1, 0.95])
save_tite = 'Comparison of mean wind direction and velocity between Osp1 A and Slåtterøy Fyr.png'
fig.savefig(save_tite)

#%% plot the top sensors of four masts with ref station

time_s = mdates.num2date(min(Osp1['Time'][0],Osp2['Time'][0],Svar['Time'][0],Synn['Time'][0]))
time_e = mdates.num2date(Time[-1])

plt.close("all")
fig     = plt.figure(figsize=(20, 12))
ax1     = plt.subplot(511)
# format the ticks
locator = mdates.AutoDateLocator(minticks=7, maxticks=9)
```

```
formatter = mdates.ConciseDateFormatter(locator)
ax1.xaxis.set_major_locator(locator)
ax1.xaxis.set_major_formatter(formatter)
ax1.plot(mdates.num2date(Osp1['Time']), Osp1['A_U'], 'k-',label=r'$\overline{U} $
$(Osp1\_A)$',markeredgecolor='k',markersize=8)
ax1.plot(Time_Osp1_A_storm, Osp1['A_U'][Osp1_A_p],  'r*',label='Storm
(Osp1)',markeredgecolor='r',markersize=12)
datemin = np.datetime64(time_s, 'm')
datemax = np.datetime64(time_e, 'm') + np.timedelta64(1, 'm')
ax1.set_xlim(datemin, datemax)
ax1.set_title('Comparison of 1-h mean wind velocities among top sensors and Slåtterøy Fyr',
fontsize=25)
plt.ylabel(r'$Dir_{wi}$ $( ^o)$', fontsize=20)
plt.rc('xtick', direction='in', color='k')
plt.rc('ytick', direction='in', color='k')
plt.legend(loc='lower left',ncol=1,fontsize=16)
g1 = plt.grid(b=True, which='major', color='k', linestyle='-', linewidth=0.35)
g2 = plt.grid(b=True, which='minor', color='k', linestyle='-', linewidth=0.1)
ax1.tick_params(axis='both', labelsize=16)
plt.minorticks_on()
# plt.show()


ax2     = plt.subplot(512)
# format the ticks
locator = mdates.AutoDateLocator(minticks=7, maxticks=9)
formatter = mdates.ConciseDateFormatter(locator)
ax2.xaxis.set_major_locator(locator)
ax2.xaxis.set_major_formatter(formatter)
ax2.plot(mdates.num2date(Osp2['Time']), Osp2['A_U'], 'k-',label=r'$\overline{U} $
$(Osp2\_A)$',markeredgecolor='k',markersize=8)
ax2.plot(Time_Osp2_A_storm, Osp2['A_U'][Osp2_A_p],  'r*',label='Storm
(Osp2)',markeredgecolor='r',markersize=12)
datemin = np.datetime64(time_s, 'm')
datemax = np.datetime64(time_e, 'm') + np.timedelta64(1, 'm')
ax2.set_xlim(time_s, time_e)
plt.ylabel(r'$\overline{U} $ (m s$^{-1})$', fontsize=20)
ax2.set_title('', fontsize=25)
plt.rc('xtick', direction='in', color='k')
plt.rc('ytick', direction='in', color='k')
plt.legend(loc='lower left',ncol=1,fontsize=16)
g1 = plt.grid(b=True, which='major', color='k', linestyle='-', linewidth=0.35)
g2 = plt.grid(b=True, which='minor', color='k', linestyle='-', linewidth=0.1)
ax2.tick_params(axis='both', labelsize=16)
plt.minorticks_on()
# plt.show()


ax3     = plt.subplot(513)
# format the ticks
locator = mdates.AutoDateLocator(minticks=7, maxticks=9)
formatter = mdates.ConciseDateFormatter(locator)
ax3.xaxis.set_major_locator(locator)
ax3.xaxis.set_major_formatter(formatter)
ax3.plot(mdates.num2date(Svar['Time']), Svar['A_U'], 'k-',label=r'$\overline{U} $
$(Svar\_A)$',markeredgecolor='k',markersize=8)
ax3.plot(Time_Svar_A_storm, Svar['A_U'][Svar_A_p],  'r*',label='Storm
(Svar)',markeredgecolor='r',markersize=12)
datemin = np.datetime64(time_s, 'm')
datemax = np.datetime64(time_e, 'm') + np.timedelta64(1, 'm')
ax3.set_xlim(time_s, time_e)
plt.ylabel(r'$\overline{U} $ (m s$^{-1})$', fontsize=20)
ax3.set_title('', fontsize=25)
plt.rc('xtick', direction='in', color='k')
plt.rc('ytick', direction='in', color='k')
plt.legend(loc='lower left',ncol=1,fontsize=16)
g1 = plt.grid(b=True, which='major', color='k', linestyle='-', linewidth=0.35)
g2 = plt.grid(b=True, which='minor', color='k', linestyle='-', linewidth=0.1)
ax3.tick_params(axis='both', labelsize=16)
plt.minorticks_on()
# plt.show()

ax4     = plt.subplot(514)
# format the ticks
```

```
locator = mdates.AutoDateLocator(minticks=7, maxticks=9)
formatter = mdates.ConciseDateFormatter(locator)
ax4.xaxis.set_major_locator(locator)
ax4.xaxis.set_major_formatter(formatter)
ax4.plot(mdates.num2date(Synn['Time']), Synn['A_U'], 'k-',label=r'$\overline{U} $
$(Synn\_A)$',markeredgecolor='k',markersize=8)
ax4.plot(Time_Synn_A_storm, Synn['A_U'][Synn_A_p],  'r*',label='Storm
(Synn)',markeredgecolor='r',markersize=12)
datemin = np.datetime64(time_s, 'm')
datemax = np.datetime64(time_e, 'm') + np.timedelta64(1, 'm')
ax4.set_xlim(time_s, time_e)
plt.ylabel(r'$\overline{U} $ (m s$^{-1})$', fontsize=20)
ax4.set_title('', fontsize=25)
ax4.set_ylim(0, 30)
plt.rc('xtick', direction='in', color='k')
plt.rc('ytick', direction='in', color='k')
plt.legend(loc='lower left',ncol=1,fontsize=16)
g1 = plt.grid(b=True, which='major', color='k', linestyle='-', linewidth=0.35)
g2 = plt.grid(b=True, which='minor', color='k', linestyle='-', linewidth=0.1)
ax4.tick_params(axis='both', labelsize=16)
plt.minorticks_on()
# plt.show()


ax5     = plt.subplot(515)
# format the ticks
locator = mdates.AutoDateLocator(minticks=7, maxticks=9)
formatter = mdates.ConciseDateFormatter(locator)
ax5.xaxis.set_major_locator(locator)
ax5.xaxis.set_major_formatter(formatter)
ax5.plot(Time_Ref, data['Ws'],  'b-',label=r'$\overline{U} $
$(Ref)$',markeredgecolor='g',markersize=12,alpha=0.5)
ax5.plot(Time_Ref_storm, data['Ws'][Ref_p],  'g*',label='Storm
(Ref)',markeredgecolor='g',markersize=12,alpha=0.6)
datemin = np.datetime64(time_s, 'm')
datemax = np.datetime64(time_e, 'm') + np.timedelta64(1, 'm')
ax5.set_xlim(time_s, time_e)
plt.ylabel(r'$\overline{U} $ (m s$^{-1})$', fontsize=20)
ax5.set_title('', fontsize=25)
ax5.set_ylim(0, 30)
plt.rc('xtick', direction='in', color='k')
plt.rc('ytick', direction='in', color='k')
plt.legend(loc='lower left',ncol=1,fontsize=16)
g1 = plt.grid(b=True, which='major', color='k', linestyle='-', linewidth=0.35)
g2 = plt.grid(b=True, which='minor', color='k', linestyle='-', linewidth=0.1)
ax5.tick_params(axis='both', labelsize=16)
plt.minorticks_on()
plt.show()

fig.tight_layout(rect=[0, 0, 1, 0.95])
save_tite = 'Comparison of 1-h mean wind velocities among top sensors and Slåtterøy Fyr.png'
fig.savefig(save_tite)
```

A2 Data processing

```
# -*- coding: utf-8 -*-
"""
Created on Thu Sep 21 08:44:42 2020
Storm data visualization and processing

@author: junwan
"""

import numpy as np
import scipy.io as spio
import matplotlib
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
import scipy.signal as signal
from   scipy.optimize import curve_fit
from   scipy.optimize import least_squares
```

```
import re
import pandas as pd
from   datetime import datetime
from   datetime import timedelta
from scipy.stats import binned_statistic
import os.path
import matplotlib.dates as mdates
import json
import pickle
import transformations      as TF

from sklearn.neighbors import KernelDensity
from scipy.signal import argrelextrema

def running_mean(x, N):
    cumsum = np.cumsum(np.insert(x, 0, 0))
    return (cumsum[N:] - cumsum[:-N]) / float(N)
def datenum(d):
    return 366 + d.toordinal() + (d - datetime.fromordinal(d.toordinal())).total_seconds()/(24*60*60)
class NumpyEncoder(json.JSONEncoder):
    def default(self, obj):
        if isinstance(obj, np.ndarray):
            return obj.tolist()
        return json.JSONEncoder.default(self, obj)

# loop over month
def month_year_iter( start_month, start_year, end_month, end_year ):
    ym_start= 12*start_year + start_month - 1
    ym_end= 12*end_year + end_month
    loop=[]
    for ym in range( ym_start, ym_end ):
        loop = np.append(loop,divmod( ym, 12 ))
        #y[ym], m[ym] = divmod( ym, 12 )
    loop=loop.reshape((int(loop.size/2),2))
    return loop

# function to calculate the wind direciton and tranformation matrix from global XYZ to local wind uvw
# local v axis is conti-clockwise 90 deg with respect to u axis
# gloabl X is pointing east and global Y is pointing north
def windXY2dir(X,Y):
    dir_W  =  np.arctan2(X,Y)
    if dir_W<0:
        dir_W=dir_W+2*np.pi
    G_x = [1,0,0]
    G_y = [0,1,0]
    G_z = [0,0,1]
    S_u     = [np.sin(dir_W),        np.cos(dir_W),        0]
    S_v     = [np.sin(dir_W-np.pi/2),np.cos(dir_W-np.pi/2),0]
    S_w     = [0,0,1]
    # transformation matrix from XYZ to UVW
    TT      = TF.T_xyzXYZ(G_x, G_y, G_z, S_u, S_v, S_w, dim = '3x3')
    return (dir_W,TT)

def wind_dir(X,Y):
    dir_W  =  np.arctan2(X,Y)
    dir_W[dir_W<0] = dir_W[dir_W<0]+2*np.pi
    return (dir_W)

# function to calculate the wind direction, mean wind speed, local wind uvw components and the time
tags
def windXYZ2uvw(data):
    # calculate the non-mean for each component of the wind components in global XYZ within starting
and ending time
    data_temp = np.nanmean(data,0)
    if ~np.isnan(data.any()):
        [dir_w,tt_w] = windXY2dir(data_temp[0],data_temp[1])
        U            = np.matmul(tt_w.T , data_temp)[0]
        uvw          = np.matmul(tt_w.T , data.T).T
        return(dir_w,U,uvw)


regex_num  = r"[-+]?[.]?[\d]+(?:\d\d\d)*[\.]?\d*(?:[eE][-+]?\d+)?"

z0 = 0.01   # roughness lengthscale
```

```
delta_ms         = timedelta(milliseconds=100)


#%%
cases = pd.read_excel('Storm events_new.xlsx', sheet_name = 'Clustered wind storms sort')
delta_1d = mdates.date2num(datetime(2,1,2,1,0)) - mdates.date2num(datetime(2,1,1,1,0))
delta_1h = mdates.date2num(datetime(2,1,1,2,0)) - mdates.date2num(datetime(2,1,1,1,0))
delta_10min = mdates.date2num(datetime(2,1,1,1,10)) - mdates.date2num(datetime(2,1,1,1,0))

# for i in range(0,cases['Time_storm'].size):
for i in list(range(0,280)) + list(range(282,cases['Time_storm'].size)):

    # i = 280, 281 no data for OSP2
    # i = 15
    time = datetime.strptime(cases['Time_storm'][i], '%Y-%m-%d %H:%M:%S%z').replace(tzinfo = None)

    date_start  =   mdates.num2date(mdates.date2num(time) - delta_1d) # ±1 day
    date_end    =   mdates.num2date(mdates.date2num(time) + delta_1d - delta_1h)

# ================================================================================
#     date_start  =   mdates.date2num(time) - delta_1d*2
#     date_end    =   mdates.date2num(time) + delta_1d*2 - delta_1h
# ================================================================================

    if os.path.isfile('D:/DATA/osp1/' + 'Osp1_' + str(time.year) + '_' + str(time.month) + '_' +
str(time.day) + '_storm_raw_data.pkl'):
        Osp1_s = pd.read_pickle('D:/DATA/osp1/' + 'Osp1_' + str(time.year) + '_' + str(time.month) +
'_' + str(time.day) + '_storm_raw_data.pkl')
        Osp2_s = pd.read_pickle('D:/DATA/osp2/' + 'Osp2_' + str(time.year) + '_' + str(time.month) +
'_' + str(time.day) + '_storm_raw_data.pkl')
        Svar_s = pd.read_pickle('D:/DATA/svar/' + 'Svar_' + str(time.year) + '_' + str(time.month) +
'_' + str(time.day) + '_storm_raw_data.pkl')
        Synn_s = pd.read_pickle('D:/DATA/synn/' + 'Synn_' + str(time.year) + '_' + str(time.month) +
'_' + str(time.day) + '_storm_raw_data.pkl')

        Osp1_s.loc[abs(Osp1_s['A_X'])>100,'A_X'] = np.nan
        Osp1_s.loc[abs(Osp1_s['A_Y'])>100,'A_Y'] = np.nan
        Osp1_s.loc[abs(Osp1_s['A_Z'])>100,'A_Z'] = np.nan
        Osp1_s.loc[abs(Osp1_s['B_X'])>100,'B_X'] = np.nan
        Osp1_s.loc[abs(Osp1_s['B_Y'])>100,'B_Y'] = np.nan
        Osp1_s.loc[abs(Osp1_s['B_Z'])>100,'B_Z'] = np.nan
        Osp1_s.loc[abs(Osp1_s['C_X'])>100,'C_X'] = np.nan
        Osp1_s.loc[abs(Osp1_s['C_Y'])>100,'C_Y'] = np.nan
        Osp1_s.loc[abs(Osp1_s['C_Z'])>100,'C_Z'] = np.nan

        Osp2_s.loc[abs(Osp2_s['A_X'])>100,'A_X'] = np.nan
        Osp2_s.loc[abs(Osp2_s['A_Y'])>100,'A_Y'] = np.nan
        Osp2_s.loc[abs(Osp2_s['A_Z'])>100,'A_Z'] = np.nan
        Osp2_s.loc[abs(Osp2_s['B_X'])>100,'B_X'] = np.nan
        Osp2_s.loc[abs(Osp2_s['B_Y'])>100,'B_Y'] = np.nan
        Osp2_s.loc[abs(Osp2_s['B_Z'])>100,'B_Z'] = np.nan
        Osp2_s.loc[abs(Osp2_s['C_X'])>100,'C_X'] = np.nan
        Osp2_s.loc[abs(Osp2_s['C_Y'])>100,'C_Y'] = np.nan
        Osp2_s.loc[abs(Osp2_s['C_Z'])>100,'C_Z'] = np.nan

        Svar_s.loc[abs(Svar_s['A_X'])>100,'A_X'] = np.nan
        Svar_s.loc[abs(Svar_s['A_Y'])>100,'A_Y'] = np.nan
        Svar_s.loc[abs(Svar_s['A_Z'])>100,'A_Z'] = np.nan
        Svar_s.loc[abs(Svar_s['B_X'])>100,'B_X'] = np.nan
        Svar_s.loc[abs(Svar_s['B_Y'])>100,'B_Y'] = np.nan
        Svar_s.loc[abs(Svar_s['B_Z'])>100,'B_Z'] = np.nan
        Svar_s.loc[abs(Svar_s['C_X'])>100,'C_X'] = np.nan
        Svar_s.loc[abs(Svar_s['C_Y'])>100,'C_Y'] = np.nan
        Svar_s.loc[abs(Svar_s['C_Z'])>100,'C_Z'] = np.nan

        Synn_s.loc[abs(Synn_s['A_X'])>100,'A_X'] = np.nan
        Synn_s.loc[abs(Synn_s['A_Y'])>100,'A_Y'] = np.nan
        Synn_s.loc[abs(Synn_s['A_Z'])>100,'A_Z'] = np.nan
        Synn_s.loc[abs(Synn_s['B_X'])>100,'B_X'] = np.nan
        Synn_s.loc[abs(Synn_s['B_Y'])>100,'B_Y'] = np.nan
        Synn_s.loc[abs(Synn_s['B_Z'])>100,'B_Z'] = np.nan
        Synn_s.loc[abs(Synn_s['C_X'])>100,'C_X'] = np.nan
        Synn_s.loc[abs(Synn_s['C_Y'])>100,'C_Y'] = np.nan
        Synn_s.loc[abs(Synn_s['C_Z'])>100,'C_Z'] = np.nan
```

```
    if mdates.num2date(Osp1_s.Time[0]).minute < 30:
        date_start = datetime(year = mdates.num2date(Osp1_s.Time[0]).year, month =
mdates.num2date(Osp1_s.Time[0]).month, day = mdates.num2date(Osp1_s.Time[0]).day,
                              hour = mdates.num2date(Osp1_s.Time[0]).hour, minute = 30, second = 0,
microsecond = 0 )
    else:
        date_start = datetime(year = mdates.num2date(Osp1_s.Time[0]).year, month =
mdates.num2date(Osp1_s.Time[0]).month, day = mdates.num2date(Osp1_s.Time[0]).day,
                              hour = (mdates.num2date(Osp1_s.Time[0]).hour)%24, minute = 30, second =
0, microsecond = 0 )

    if mdates.num2date(Osp1_s.Time.iloc[-1]).minute < 30:
        date_end = datetime(year = mdates.num2date(Osp1_s.Time.iloc[-1]).year, month =
mdates.num2date(Osp1_s.Time.iloc[-1]).month, day = mdates.num2date(Osp1_s.Time.iloc[-1]).day,
                              hour = (mdates.num2date(Osp1_s.Time[0]).hour + 1)%24, minute = 30,
second = 0, microsecond = 0 )
    else:
        date_end = datetime(year = mdates.num2date(Osp1_s.Time.iloc[-1]).year, month =
mdates.num2date(Osp1_s.Time.iloc[-1]).month, day = mdates.num2date(Osp1_s.Time.iloc[-1]).day,
                              hour = (mdates.num2date(Osp1_s.Time[0]).hour + 1)%24, minute = 30,
second = 0, microsecond = 0 )

#%% Osp1
    # here we use 1 hour interval
    num_1h = int(np.floor((mdates.date2num(date_end) - mdates.date2num(date_start))/delta_1h))

    # create the startng and ending time array
    time_s = np.array([date_start + timedelta(hours = i) for i in range(0,num_1h)])
    time_e = np.array([date_start + timedelta(hours = i) for i in range(1,num_1h + 1)])

    # find the cloest index of the starting and ending time of each hour
    idxs = Osp1_s.loc[:,'Time'].searchsorted(mdates.date2num(time_s),side = 'right')
    idxe = Osp1_s.loc[:,'Time'].searchsorted(mdates.date2num(time_e),side = 'left')

    time_ref   = []
    A_dir      = []
    A_Dir      = []
    A_U        = []
    A_uvw      = []
    B_dir      = []
    B_Dir      = []
    B_U        = []
    B_uvw      = []
    C_dir      = []
    C_Dir      = []
    C_U        = []
    C_uvw      = []
    time_ss    = []

    for j in range(0,num_1h):
        # if idxs=idxe, then no data avaliable in this hour
        if idxs[j] !=  idxe[j]:
            date_ref = mdates.num2date((mdates.date2num(time_s[j]) + mdates.date2num(time_e[j]))/2)

            [Dir_w_A,U_A,uvw_A] =
windXYZ2uvw(np.asarray(Osp1_s.loc[idxs[j]:idxe[j],['A_X','A_Y','A_Z']],dtype = np.float64))
            dir_w_A             = wind_dir(np.asarray(Osp1_s.loc[idxs[j]:idxe[j],['A_X']],dtype =
np.float64)[:,0],np.asarray(Osp1_s.loc[idxs[j]:idxe[j],['A_Y']],dtype = np.float64)[:,0])
            Dir_w_A = (np.degrees(Dir_w_A) + 180)%360
            dir_w_A = (np.degrees(dir_w_A) + 180)%360
            A_Dir.append(Dir_w_A)
            A_dir.extend(dir_w_A)
            A_U.append(U_A)
            A_uvw.extend(uvw_A)
            time_ref.append(date_ref)
            time_ss.extend(np.asarray(Osp1_s.loc[idxs[j]:idxe[j],['Time']],dtype = np.float64))

            [Dir_w_B,U_B,uvw_B] =
windXYZ2uvw(np.asarray(Osp1_s.loc[idxs[j]:idxe[j],['B_X','B_Y','B_Z']],dtype = np.float64))
            dir_w_B             = wind_dir(np.asarray(Osp1_s.loc[idxs[j]:idxe[j],['B_X']],dtype =
np.float64)[:,0],np.asarray(Osp1_s.loc[idxs[j]:idxe[j],['B_Y']],dtype = np.float64)[:,0])
            Dir_w_B = (np.degrees(Dir_w_B) + 180) %360
            dir_w_B = (np.degrees(dir_w_B) + 180) %360
```

```
            B_Dir.append(Dir_w_B)
            B_dir.extend(dir_w_B)
            B_U.append(U_B)
            B_uvw.extend(uvw_B)

            [Dir_w_C,U_C,uvw_C] =
windXYZ2uvw(np.asarray(Osp1_s.loc[idxs[j]:idxe[j],['C_X','C_Y','C_Z']],dtype = np.float64))
            dir_w_C             = wind_dir(np.asarray(Osp1_s.loc[idxs[j]:idxe[j],['C_X']],dtype =
np.float64)[:,0],np.asarray(Osp1_s.loc[idxs[j]:idxe[j],['C_Y']],dtype = np.float64)[:,0])
            Dir_w_C = (np.degrees(Dir_w_C) + 180)%360
            dir_w_C = (np.degrees(dir_w_C) + 180)%360
            C_Dir.append(Dir_w_C)
            C_dir.extend(dir_w_C)
            C_U.append(U_C)
            C_uvw.extend(uvw_C)

    A_uvw = np.array(A_uvw)
    B_uvw = np.array(B_uvw)
    C_uvw = np.array(C_uvw)
    A_dir = np.array(A_dir)
    B_dir = np.array(B_dir)
    C_dir = np.array(C_dir)
    time_ss = np.array(time_ss)

    # save instaneous wind data into a dataframe and then save as a pkl file
    Osp1_1h_ins = pd.DataFrame(columns =
['Time','A_dir','A_u','A_v','A_w','B_dir','B_u','B_v','B_w','C_dir','C_u','C_v','C_w'])
    Osp1_1h_ins['Time'] = time_ss[:,0]
    Osp1_1h_ins.loc[Osp1_1h_ins.index,['A_dir']] = A_dir
    Osp1_1h_ins.loc[Osp1_1h_ins.index,['A_u','A_v','A_w']] = A_uvw
    Osp1_1h_ins.loc[Osp1_1h_ins.index,['B_dir']] = B_dir
    Osp1_1h_ins.loc[Osp1_1h_ins.index,['B_u','B_v','B_w']] = B_uvw
    Osp1_1h_ins.loc[Osp1_1h_ins.index,['C_dir']] = C_dir
    Osp1_1h_ins.loc[Osp1_1h_ins.index,['C_u','C_v','C_w']] = C_uvw

    # convert object datetypes to float datatype
    Osp1_1h_ins = Osp1_1h_ins.astype({'A_u':'float64','A_v':
'float64','A_w':'float64','B_u':'float64','B_v': 'float64','B_w':'float64','C_u':'float64','C_v':
'float64','C_w':'float64'})

    Osp1_1h_ins.to_pickle('D:/DATA/osp1/' + 'Osp1_' + str(time.year) + '_' + str(time.month) + '_' +
str(time.day) + '_storm_' + 'Osp1_1h' + '_ins.pkl')

    # save mean wind data into a dataframe and then save as a pkl file
    Osp1_1h_mean  = pd.DataFrame(columns=['Time', 'A_Dir','A_U','B_Dir', 'B_U', 'C_Dir','C_U'])
    Osp1_1h_mean['Time'] = time_ref
    Osp1_1h_mean.loc[Osp1_1h_mean.index,['A_Dir']] = A_Dir
    Osp1_1h_mean.loc[Osp1_1h_mean.index,['A_U']] = A_U
    Osp1_1h_mean.loc[Osp1_1h_mean.index,['B_Dir']] = B_Dir
    Osp1_1h_mean.loc[Osp1_1h_mean.index,['B_U']] = B_U
    Osp1_1h_mean.loc[Osp1_1h_mean.index,['C_Dir']] = C_Dir
    Osp1_1h_mean.loc[Osp1_1h_mean.index,['C_U']] = C_U

    Osp1_1h_mean = Osp1_1h_mean.astype({'A_Dir':'float64','A_U':
'float64','B_Dir':'float64','B_U':'float64','C_Dir': 'float64','C_U':'float64'})

    Osp1_1h_mean.to_pickle('D:/DATA/osp1/' + 'Osp1_' + str(time.year) + '_' + str(time.month) + '_' +
str(time.day) + '_storm_' + 'Osp1_1h' + '_mean.pkl')


    Osp1_1h  = {'A_U':  A_U,   'B_U':  B_U,   'C_U':  C_U,   'A_Dir':A_Dir, 'B_Dir':B_Dir,
'C_Dir':C_Dir,\
                'A_uvw':A_uvw, 'B_uvw':B_uvw, 'C_uvw':C_uvw, 'A_dir':A_dir, 'B_dir':B_dir,
'C_dir':C_dir,\
                'time':time_ss, 'Time':time_ref    }

    with open('D:/DATA/osp1/' + 'Osp1_' + str(time.year) + '_' + str(time.month) + '_' + str(time.day)
+ '_storm_' + 'Osp1_1h' +'.pkl', 'wb') as f:
        pickle.dump(Osp1_1h, f, pickle.HIGHEST_PROTOCOL)

# ============================================================================
#     with open('D:/DATA/osp1/' +'Osp1_'+ str(time.year) +'_' + str(time.month)+'_' + str(time.day)+
'_storm_' + 'Osp1_1h' +'.pkl', 'rb') as f:
#         tt=pickle.load( f)
```

```
# ============================================================================

#%% Osp1
    # here we use 10 mins interval
    num_10min        = int(np.floor((mdates.date2num(date_end)-
mdates.date2num(date_start))/delta_10min))
    # create the startng and ending time array
    time_s = np.array([date_start + timedelta(minutes=i*10) for i in range(0,num_10min)])
    time_e = np.array([date_start + timedelta(minutes=i*10) for i in range(1,num_10min+1)])

    # find the cloest index of the starting and ending time of each hour
    idxs=Osp1_s.loc[:,'Time'].searchsorted (mdates.date2num(time_s),side='right')
    idxe=Osp1_s.loc[:,'Time'].searchsorted (mdates.date2num(time_e),side='left')

    time_ref   = []
    A_dir      = []
    A_Dir      = []
    A_U        = []
    A_uvw      = []
    B_dir      = []
    B_Dir      = []
    B_U        = []
    B_uvw      = []
    C_dir      = []
    C_Dir      = []
    C_U        = []
    C_uvw      = []
    time_ss    = []

    for j in range(0,num_10min):
        # if idxs=idxe, then no data avaliable in this hour
        if idxs[j] !=  idxe[j]:
            date_ref  =  mdates.num2date((mdates.date2num(time_s[j]) + mdates.date2num(time_e[j]))/2)

            [Dir_w_A,U_A,uvw_A] =
windXYZ2uvw(np.asarray(Osp1_s.loc[idxs[j]:idxe[j],['A_X','A_Y','A_Z']],dtype=np.float64))
            dir_w_A             =
wind_dir(np.asarray(Osp1_s.loc[idxs[j]:idxe[j],['A_X']],dtype=np.float64)[:,0],np.asarray(Osp1_s.loc[i
dxs[j]:idxe[j],['A_Y']],dtype=np.float64)[:,0])
            Dir_w_A=(np.degrees(Dir_w_A)+180) %360
            dir_w_A=(np.degrees(dir_w_A)+180) %360
            A_Dir.append(Dir_w_A)
            A_dir.extend(dir_w_A)
            A_U.append(U_A)
            A_uvw.extend(uvw_A)
            time_ref.append(date_ref)
            time_ss.extend(np.asarray(Osp1_s.loc[idxs[j]:idxe[j],['Time']],dtype=np.float64))

            [Dir_w_B,U_B,uvw_B] =
windXYZ2uvw(np.asarray(Osp1_s.loc[idxs[j]:idxe[j],['B_X','B_Y','B_Z']],dtype=np.float64))
            dir_w_B             =
wind_dir(np.asarray(Osp1_s.loc[idxs[j]:idxe[j],['B_X']],dtype=np.float64)[:,0],np.asarray(Osp1_s.loc[i
dxs[j]:idxe[j],['B_Y']],dtype=np.float64)[:,0])
            Dir_w_B=(np.degrees(Dir_w_B)+180) %360
            dir_w_B=(np.degrees(dir_w_B)+180) %360
            B_Dir.append(Dir_w_B)
            B_dir.extend(dir_w_B)
            B_U.append(U_B)
            B_uvw.extend(uvw_B)

            [Dir_w_C,U_C,uvw_C] =
windXYZ2uvw(np.asarray(Osp1_s.loc[idxs[j]:idxe[j],['C_X','C_Y','C_Z']],dtype=np.float64))
            dir_w_C             =
wind_dir(np.asarray(Osp1_s.loc[idxs[j]:idxe[j],['C_X']],dtype=np.float64)[:,0],np.asarray(Osp1_s.loc[i
dxs[j]:idxe[j],['C_Y']],dtype=np.float64)[:,0])
            Dir_w_C=(np.degrees(Dir_w_C)+180) %360
            dir_w_C=(np.degrees(dir_w_C)+180) %360
            C_Dir.append(Dir_w_C)
            C_dir.extend(dir_w_C)
            C_U.append(U_C)
            C_uvw.extend(uvw_C)

    A_uvw = np.array(A_uvw)
    B_uvw = np.array(B_uvw)
```

```
    C_uvw = np.array(C_uvw)
    A_dir = np.array(A_dir)
    B_dir = np.array(B_dir)
    C_dir = np.array(C_dir)
    time_ss = np.array(time_ss)


    # save instaneous wind data into a dataframe and then save as a pkl file
    Osp1_10min_ins  = pd.DataFrame(columns=['Time', 'A_dir','A_u', 'A_v','A_w','B_dir', 'B_u',
'B_v','B_w','C_dir','C_u', 'C_v','C_w'])
    Osp1_10min_ins['Time']  =    time_ss[:,0]
    Osp1_10min_ins.loc[Osp1_10min_ins.index,['A_dir']] = A_dir
    Osp1_10min_ins.loc[Osp1_10min_ins.index,['A_u','A_v','A_w']] = A_uvw
    Osp1_10min_ins.loc[Osp1_10min_ins.index,['B_dir']] = B_dir
    Osp1_10min_ins.loc[Osp1_10min_ins.index,['B_u','B_v','B_w']] = B_uvw
    Osp1_10min_ins.loc[Osp1_10min_ins.index,['C_dir']] = C_dir
    Osp1_10min_ins.loc[Osp1_10min_ins.index,['C_u','C_v','C_w']] = C_uvw

    Osp1_10min_ins = Osp1_10min_ins.astype({'A_u':'float64','A_v':
'float64','A_w':'float64','B_u':'float64','B_v': 'float64','B_w':'float64','C_u':'float64','C_v':
'float64','C_w':'float64'})

    Osp1_10min_ins.to_pickle('D:/DATA/osp1/' +'Osp1_'+ str(time.year) +'_' + str(time.month)+'_' +
str(time.day)+ '_storm_' + 'Osp1_10min' +'_ins.pkl')

    # save mean wind data into a dataframe and then save as a pkl file
    Osp1_10min_mean  = pd.DataFrame(columns=['Time', 'A_Dir','A_U','B_Dir', 'B_U', 'C_Dir','C_U'])
    Osp1_10min_mean['Time']  =    time_ref
    Osp1_10min_mean.loc[Osp1_10min_mean.index,['A_Dir']] = A_Dir
    Osp1_10min_mean.loc[Osp1_10min_mean.index,['A_U']] = A_U
    Osp1_10min_mean.loc[Osp1_10min_mean.index,['B_Dir']] = B_Dir
    Osp1_10min_mean.loc[Osp1_10min_mean.index,['B_U']] = B_U
    Osp1_10min_mean.loc[Osp1_10min_mean.index,['C_Dir']] = C_Dir
    Osp1_10min_mean.loc[Osp1_10min_mean.index,['C_U']] = C_U

    Osp1_10min_mean = Osp1_10min_mean.astype({'A_Dir':'float64','A_U':
'float64','B_Dir':'float64','B_U':'float64','C_Dir': 'float64','C_U':'float64'})

    Osp1_10min_mean.to_pickle('D:/DATA/osp1/' +'Osp1_'+ str(time.year) +'_' + str(time.month)+'_' +
str(time.day)+ '_storm_' + 'Osp1_10min' +'_mean.pkl')

# ============================================================================
#     Osp1_10min  = {'A_U':  A_U,   'B_U':  B_U,   'C_U':  C_U,   'A_Dir':A_Dir, 'B_Dir':B_Dir,
'C_Dir':C_Dir,\
#                 'A_uvw':A_uvw, 'B_uvw':B_uvw, 'C_uvw':C_uvw, 'A_dir':A_dir, 'B_dir':B_dir,
'C_dir':C_dir,\
#                 'time':time_ss, 'Time':time_ref    }
#     with open('D:/DATA/osp1/' +'Osp1_'+ str(time.year) +'_' + str(time.month)+'_' + str(time.day)+
'_storm_' + 'Osp1_10min' +'.pkl', 'wb') as f:
#         pickle.dump(Osp1_10min, f, pickle.HIGHEST_PROTOCOL)
# ============================================================================

#%% Osp1 temp
    # here we use 1 hour interval
    num_1h = int(np.floor((mdates.date2num(date_end) - mdates.date2num(date_start))/delta_1h))

    # create the startng and ending time array
    time_s = np.array([date_start + timedelta(hours = i) for i in range(0,num_1h)])
    time_e = np.array([date_start + timedelta(hours = i) for i in range(1,num_1h + 1)])

    # find the cloest index of the starting and ending time of each hour
    idxs = Osp1_s.loc[:,'Time'].searchsorted(mdates.date2num(time_s),side = 'right')
    idxe = Osp1_s.loc[:,'Time'].searchsorted(mdates.date2num(time_e),side = 'left')

    A_temp     = []
    time_ss    = []

    for j in range(0,num_1h):
        # if idxs=idxe, then no data avaliable in this hour
        if idxs[j] !=  idxe[j]:
            date_ref = mdates.num2date((mdates.date2num(time_s[j]) + mdates.date2num(time_e[j]))/2)

            A_temp.extend(np.asarray(Osp1_s.loc[idxs[j]:idxe[j],['A_temp']],dtype = np.float64))
            time_ss.extend(np.asarray(Osp1_s.loc[idxs[j]:idxe[j],['Time']],dtype = np.float64))
```

```
    time_ss = np.array(time_ss)

    # save instaneous wind data into a dataframe and then save as a pkl file
    Osp1_1h_ins = pd.DataFrame(columns = ['Time','A_temp'])
    Osp1_1h_ins['Time'] = time_ss[:,0]
    Osp1_1h_ins.loc[Osp1_1h_ins.index,['A_temp']] = A_temp

    # convert object datetype to float datatype
    Osp1_1h_ins = Osp1_1h_ins.astype({'A_temp':'float64'})

    Osp1_1h_ins.to_pickle('D:/DATA/osp1/' + 'Osp1_' + str(time.year) + '_' + str(time.month) + '_' +
str(time.day) + '_storm_temp_' + 'Osp1_1h' + '_ins.pkl')

#%% Storm time history

    with open('D:/DATA/osp1/' + 'Osp1_' + str(time.year) + '_' + str(time.month) + '_' + str(time.day)
+ '_storm_' + 'Osp1_1h' +'.pkl', 'rb') as f:
          Osp1_1h = pickle.load( f)
    with open('D:/DATA/osp2/' +'Osp2_'+ str(time.year) +'_' + str(time.month)+'_' + str(time.day)+
'_storm_' + 'Osp2_1h' +'.pkl', 'rb') as f:
        Osp2_1h=pickle.load( f)
    with open('D:/DATA/svar/' +'Svar_'+ str(time.year) +'_' + str(time.month)+'_' + str(time.day)+
'_storm_' + 'Svar_1h' +'.pkl', 'rb') as f:
        Svar_1h=pickle.load( f)
    with open('D:/DATA/synn/' +'Synn_'+ str(time.year) +'_' + str(time.month)+'_' + str(time.day)+
'_storm_' + 'Synn_1h' +'.pkl', 'rb') as f:
        Synn_1h=pickle.load( f)

    Osp1_1h['A_Dir']= np.asarray(Osp1_1h['A_Dir'])

    if  round(max(Osp1_1h['A_dir']))-round(min(Osp1_1h['A_dir'])) == 360:
          Osp1_1h['A_dir'][Osp1_1h['A_dir']<20] =  Osp1_1h['A_dir'][Osp1_1h['A_dir']<20]+360
          Osp1_1h['A_Dir'][Osp1_1h['A_Dir']<20] =  Osp1_1h['A_Dir'][Osp1_1h['A_Dir']<20]+360

    dt = 0.1
    Nwin  = round(60*60./dt);
      # plot the wind direction, mean wind speed and along wind turbulence intensity
    plt.close("all")
    fig     = plt.figure(figsize=(20, 12))
    ax1     = plt.subplot(311)
      # format the ticks
    locator = mdates.AutoDateLocator(minticks=7, maxticks=9)
    formatter = mdates.ConciseDateFormatter(locator)
    ax1.xaxis.set_major_locator(locator)
    ax1.xaxis.set_major_formatter(formatter)
    ax1.plot(Osp1_1h['time'], Osp1_1h['A_dir'], 'k-',label='$Dir_{wi}$
$(Osp1\_A)$',markeredgecolor='k',markersize=8,alpha=0.75)
    # movingdir_Osp1   =
np.asarray(pd.DataFrame(Osp1_1h['A_dir']).rolling(Nwin,min_periods=1).mean())
    # ax1.plot(Osp1_1h['time'], movingdir_Osp1, 'r-',label='$Dir_{wi}$
$(Osp1\_A)$',markeredgecolor='k',markersize=8,alpha=0.75)
    ax1.plot(Osp1_1h['Time'], Osp1_1h['A_Dir'], 'r-',label='$Dir_{wi}$
$(Osp1\_A)$',markeredgecolor='k',markersize=8)
    datemin = np.datetime64(mdates.num2date(Osp1_1h['time'][0][0], tz = None), 'h') #.replace(tzinfo =
None)
    datemax = np.datetime64(mdates.num2date(Osp1_1h['time'][-1][0], tz = None), 'm') +
np.timedelta64(1, 'h') #.replace(tzinfo = None)
    ax1.set_xlim(datemin, datemax)
    plt.ylabel(r'$Dir_{wi}$ $( ^o)$', fontsize=20)
    fig.suptitle(str(time.year) + '_' + str(time.month)+ '_' + str(time.day)+ ' wind event time
history', fontsize=25)
    plt.rc('xtick', direction='in', color='k')
    plt.rc('ytick', direction='in', color='k')
    plt.legend(loc='best',ncol=1,fontsize=16)
    g1 = plt.grid(b=True, which='major', color='k', linestyle='-', linewidth=0.35)
    g2 = plt.grid(b=True, which='minor', color='k', linestyle='-', linewidth=0.1)
    ax1.tick_params(axis='both', labelsize=16)
    plt.minorticks_on()


    ax2     = plt.subplot(312)
      # format the ticks
    locator = mdates.AutoDateLocator(minticks=7, maxticks=9)
```

```
    formatter = mdates.ConciseDateFormatter(locator)
    ax2.xaxis.set_major_locator(locator)
    ax2.xaxis.set_major_formatter(formatter)
    ax2.plot(Osp1_1h['time'], Osp1_1h['A_uvw'][:,0], 'b-',label='$\overline{U} + u $
$(Osp1\_A)$',markeredgecolor='k',markersize=8,alpha=0.5)
    #movingU_Osp1   =
np.asarray(pd.DataFrame(Osp1_1h['A_uvw'][:,0]).rolling(Nwin,min_periods=1).mean())
    #ax2.plot(Osp1_1h['time'], movingU_Osp1, 'r-',label='$\overline{U} $
$(Osp1\_A)$',markeredgecolor='k',markersize=8,alpha=0.5)
    ax2.plot(Osp1_1h['Time'], Osp1_1h['A_U'], 'r-',label='$\overline{U} $
$(Osp1\_A)$',markeredgecolor='k',markersize=8)
    datemin = np.datetime64(mdates.num2date(Osp1_s.loc[0,'Time'], tz = None), 'h') #.replace(tzinfo =
None)
    datemax = np.datetime64(mdates.num2date(Osp1_s.loc[Osp1_s.index[-1],'Time'], tz = None), 'm') +
np.timedelta64(1, 'h') #.replace(tzinfo = None)
    ax2.set_xlim(datemin, datemax)
    plt.ylabel(r'$\overline{U} + u$ (m s$^{-1})$', fontsize=20)
    ax2.set_title('', fontsize=25)
    plt.rc('xtick', direction='in', color='k')
    plt.rc('ytick', direction='in', color='k')
    plt.legend(loc='best',ncol=1,fontsize=16)
    g1 = plt.grid(b=True, which='major', color='k', linestyle='-', linewidth=0.35)
    g2 = plt.grid(b=True, which='minor', color='k', linestyle='-', linewidth=0.1)
    ax2.tick_params(axis='both', labelsize=16)
    plt.minorticks_on()
    # plt.show()


    ax3     = plt.subplot(313)
      # format the ticks
    locator = mdates.AutoDateLocator(minticks=7, maxticks=9)
    formatter = mdates.ConciseDateFormatter(locator)
    ax3.xaxis.set_major_locator(locator)
    ax3.xaxis.set_major_formatter(formatter)
    ax3.plot(Osp1_1h['Time'], Osp1_1h['A_U'], 'ro-',label='$\overline{U} $
$(Osp1\_A)$',markeredgecolor='r',markersize=8,alpha=0.75)
    ax3.plot(Osp2_1h['Time'], Osp2_1h['A_U'], 'ko-',label='$\overline{U} $
$(Osp2\_A)$',markeredgecolor='k',markersize=8,alpha=0.75)
    ax3.plot(Svar_1h['Time'], Svar_1h['A_U'], 'go-',label='$\overline{U} $
$(Svar\_A)$',markeredgecolor='g',markersize=8,alpha=0.75)
    ax3.plot(Synn_1h['Time'], Synn_1h['A_U'], 'bo-',label='$\overline{U} $
$(Synn\_A)$',markeredgecolor='b',markersize=8,alpha=0.75)

    datemin = np.datetime64(mdates.num2date(Osp1_s.loc[0,'Time'], tz = None), 'h') #.replace(tzinfo =
None)
    datemax = np.datetime64(mdates.num2date(Osp1_s.loc[Osp1_s.index[-1],'Time'], tz = None), 'm') +
np.timedelta64(1, 'h') #.replace(tzinfo = None)
    ax3.set_xlim(datemin, datemax)
    plt.ylabel(r'$\overline{U}$ (m s$^{-1})$', fontsize=20)
    ax3.set_title('', fontsize=25)
    plt.rc('xtick', direction='in', color='k')
    plt.rc('ytick', direction='in', color='k')
    plt.legend(loc='best',ncol=1,fontsize=16)
    g1 = plt.grid(b=True, which='major', color='k', linestyle='-', linewidth=0.35)
    g2 = plt.grid(b=True, which='minor', color='k', linestyle='-', linewidth=0.1)
    ax3.tick_params(axis='both', labelsize=16)
    plt.minorticks_on()
    # plt.show()
    fig.tight_layout(rect=[0, 0, 1, 0.95])
    path = 'D:/DATA/Plots/Storm time histories/'
    save_tite = str(time.year) +'_' + str(time.month)+'_' + str(time.day)+ '_storm_time_history.png'
    fig.savefig(path + save_tite)
```

A3 Integral length scale estimation

```
# -*- coding: utf-8 -*-
"""
Created on Thu Sep 21 08:44:42 2020
Storm data spectral fitting

@author: junwan
```

```
"""

import numpy as np
import scipy.io as spio
import matplotlib
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
import scipy.signal as signal
from   scipy.optimize import curve_fit
from   scipy.optimize import least_squares
import re
import pandas as pd
from   datetime import datetime
from   datetime import timedelta
from scipy.stats import binned_statistic
import os.path
import matplotlib.dates as mdates
import json
import pickle
import transformations      as TF
import statsmodels.api as sm
from scipy import integrate

from sklearn.neighbors import KernelDensity
from scipy.signal import argrelextrema

def running_mean(x, N):
    cumsum = np.cumsum(np.insert(x, 0, 0))
    return (cumsum[N:] - cumsum[:-N]) / float(N)
def datenum(d):
    return 366 + d.toordinal() + (d - datetime.fromordinal(d.toordinal())).total_seconds()/(24*60*60)
class NumpyEncoder(json.JSONEncoder):
    def default(self, obj):
        if isinstance(obj, np.ndarray):
            return obj.tolist()
        return json.JSONEncoder.default(self, obj)

# loop over month
def month_year_iter( start_month, start_year, end_month, end_year ):
    ym_start= 12*start_year + start_month - 1
    ym_end= 12*end_year + end_month
    loop=[]
    for ym in range( ym_start, ym_end ):
        loop = np.append(loop,divmod( ym, 12 ))
        #y[ym], m[ym] = divmod( ym, 12 )
    loop=loop.reshape((int(loop.size/2),2))
    return loop

# function to calculate the wind direciton and tranformation matrix from global XYZ to local wind uvw
# local v axis is conti-clockwise 90 deg with respect to u axis
# gloabl X is pointing east and global Y is pointing north
def windXY2dir(X,Y):
    dir_W  =  np.arctan2(X,Y)
    if dir_W<0:
        dir_W=dir_W+2*np.pi
    G_x = [1,0,0]
    G_y = [0,1,0]
    G_z = [0,0,1]
    S_u     = [np.sin(dir_W),        np.cos(dir_W),        0]
    S_v     = [np.sin(dir_W-np.pi/2),np.cos(dir_W-np.pi/2),0]
    S_w     = [0,0,1]
    # transformation matrix from XYZ to UVW
    TT      = TF.T_xyzXYZ(G_x, G_y, G_z, S_u, S_v, S_w, dim='3x3')
    return (dir_W,TT)

def wind_dir(X,Y):
    dir_W  =  np.arctan2(X,Y)
    dir_W[dir_W<0] = dir_W[dir_W<0]+2*np.pi
    return (dir_W)

# function to calculate the wind direction, mean wind speed, local wind uvw components and the time
tags
def windXYZ2uvw(data):
```

```
    # calculate the non-mean for each component of the wind components in global XYZ within starting
and ending time
    data_temp = np.nanmean(data,0)
    if ~np.isnan(data.any()):
        [dir_w,tt_w] = windXY2dir(data_temp[0],data_temp[1])
        U            = np.matmul(tt_w.T , data_temp)[0]
        uvw          = np.matmul(tt_w.T , data.T).T
        return(dir_w,U,uvw)

# round time to 1 minute
def round_time(dt=None, round_to=60):
   if dt == None:
       dt = datetime.now()
   seconds = (dt - dt.min).seconds
   rounding = (seconds+round_to/2) // round_to * round_to
   return dt + timedelta(0,rounding-seconds,-dt.microsecond)

# Find zero-crossings
def zero_crossing(data):
    return np.where(np.diff(np.sign(data)))[0]

regex_num  = r"[-+]?[.]?[\d]+(?:\d\d\d)*[\.]?\d*(?:[eE][-+]?\d+)?"

z0=0.01   # roughness lengthscale
delta_ms         = timedelta(milliseconds=100)
delta_2d         = timedelta(days=2)
delta_30min      = timedelta(minutes=30)
delta_5min       = timedelta(minutes=5)

#%%
cases = pd.read_excel('Storm events_new.xlsx', sheet_name = 'Clustered wind storms sort')
delta_1d         = mdates.date2num(datetime(2,1,2,1,0))-mdates.date2num(datetime(2,1,1,1,0))
delta_1h         = mdates.date2num(datetime(2,1,1,2,0))-mdates.date2num(datetime(2,1,1,1,0))
delta_10min      = mdates.date2num(datetime(2,1,1,1,10))-mdates.date2num(datetime(2,1,1,1,0))

#%%
# for event in range(0,cases['Time_storm'].size):
# for event in (53, 68, 162, 166, 216, 273, 280, 283, 286, 290, 291, 292, 293, 294, 299, 303):
for event in list(range(0,53)) + list(range(54,68)) + list(range(69,162)) + list(range(163,166)) +
list(range(167,216)) + list(range(217,273)) + list(range(274,280)) + list(range(282,283)) +
list(range(284,286)) + list(range(287,290)) + list(range(295,299)) + list(range(300,303)) +
list(range(304,306)):
    # event = 284
    time = datetime.strptime(cases['Time_storm'][event], '%Y-%m-%d %H:%M:%S%z').replace(tzinfo = None)

    # starting and ending time of analyzed data window (based on storm mean wind speed time history)
    time_s1 = datetime.strptime(cases['Time_s1'][event], '%Y-%m-%d %H:%M:%S%z').replace(tzinfo = None)
    time_e1 = datetime.strptime(cases['Time_e1'][event], '%Y-%m-%d %H:%M:%S%z').replace(tzinfo = None)

    # Osp1 height info
    info_osp1 = pd.read_excel('D:/DATA/Bjørnafjorden/files_osp1.xls', sheet_name = 'Height')

    if os.path.isfile('D:/DATA/osp1/' + 'Osp1_' + str(time.year) + '_' + str(time.month) + '_' +
str(time.day) + '_storm_raw_data.pkl'):
        with open('D:/DATA/osp1/' + 'Osp1_' + str(time.year) + '_' + str(time.month) + '_' +
str(time.day) + '_storm_' + 'Osp1_10min' + '_ins.pkl', 'rb') as f:
            Osp1_10min_ins  = pickle.load( f )
        with open('D:/DATA/osp1/' + 'Osp1_' + str(time.year) + '_' + str(time.month) + '_' +
str(time.day) + '_storm_' + 'Osp1_10min' + '_mean.pkl', 'rb') as f:
            Osp1_10min_mean = pickle.load( f )

        # number of hourly events
        num_case = int(np.round((mdates.date2num(time_e1) - mdates.date2num(time_s1))/delta_10min))
        # num_case = 24

        U           = np.zeros((num_case,3)) # lateral mean wind speed
        Dir         = np.zeros((num_case,3)) # mean wind direction
        stdu        = np.zeros((num_case,3)) # standard deviation of turbulence component u
        Iu          = np.zeros((num_case,3)) # lateral turbulence intensity
        Iv          = np.zeros((num_case,3)) # longitudinal turbulence intensity
        Iw          = np.zeros((num_case,3)) # vertical turbulence intensity
        Lux         = np.zeros((num_case,3)) # length scale along-wind direction
        Lvx         = np.zeros((num_case,3)) # length scale across-wind direction
        Lwx         = np.zeros((num_case,3)) # length scale vertical direction
```

```
#%%
        for flag_case in range(1,num_case):
            # flag_case = 1
            # make sure the starting and ending date is half a hour before and after the time_tag (to
be consistent with mean values)
            date_begin = round_time (mdates.num2date(mdates.date2num(time_s1)-0.5*delta_10min +
delta_10min*flag_case   ).replace(tzinfo=None))
            date_stop  = round_time (mdates.num2date(mdates.date2num(date_begin) + delta_10min
).replace(tzinfo=None))
            # integer hour tag
            date       = mdates.num2date(mdates.date2num(time_s1) +
delta_10min*flag_case).replace(tzinfo=None).replace(microsecond=0)
            # get the index of the starting and ending hour index in time

            # the original date number is based on matplotlib 3.1.3, so we have to create new time
series here
            # the time_new is created based on the time tag of the mean wind storm data
            date_new_start =  Osp1_10min_mean.iloc[0,0]  - delta_5min
            date_new_end   =  Osp1_10min_mean.iloc[-1,0] + delta_5min  -  delta_ms
            time_new = pd.date_range(date_new_start,date_new_end,freq = '100L')
            time_new = pd.Series(mdates.date2num(time_new))

            # corresponding time tag of each hourly event
            num_s     = time_new.index[time_new == mdates.date2num(date_begin)].tolist()
            num_e     = time_new.index[time_new == mdates.date2num(date_stop)].tolist()
            Num       = Osp1_10min_mean.index[Osp1_10min_mean['Time'].dt.tz_localize(None) ==
date].tolist()

            # get the target time series
            tt    = time_new.iloc[num_s[0]:num_e[0]].reset_index(drop=True)

            # check the ratio of NANs in data, if it's more than 10%, we do not work on it. otherwise,
we interpolate
            nan_ratio =
Osp1_10min_ins.loc[num_s[0]:num_e[0],['A_u','B_u','C_u']].isnull().sum().sum()/Osp1_10min_ins.loc[num_
s[0]:num_e[0],['A_u','B_u','C_u']].size.sum()
            if  nan_ratio < 0.05:

                # get the raw data (interpolate the data for NAN using linear method)
                dd    = np.asarray(Osp1_10min_ins.loc[num_s[0]:num_e[0] -
1,['A_dir','B_dir','C_dir']].interpolate(axis = 0,limit_direction = 'both'))
                UU    = np.asarray(Osp1_10min_ins.loc[num_s[0]:num_e[0] -
1,['A_u','B_u','C_u']].interpolate(axis = 0,limit_direction = 'both'))
                VV    = np.asarray(Osp1_10min_ins.loc[num_s[0]:num_e[0] -
1,['A_v','B_v','C_v']].interpolate(axis = 0,limit_direction = 'both'))
                WW    = np.asarray(Osp1_10min_ins.loc[num_s[0]:num_e[0] -
1,['A_w','B_w','C_w']].interpolate(axis = 0,limit_direction = 'both'))

                # get the mean value (10 MINS AVERAGING)
                dt          = 0.1
                Nwin        = round(60./dt)
                DD          = np.asarray(pd.DataFrame(dd).rolling(Nwin,min_periods = 1).mean())
                movingU     = np.asarray(pd.DataFrame(UU).rolling(Nwin,min_periods = 1).mean())
                movingV     = np.asarray(pd.DataFrame(VV).rolling(Nwin,min_periods = 1).mean())
                movingW     = np.asarray(pd.DataFrame(WW).rolling(Nwin,min_periods = 1).mean())

                # get the turbulent components
                uu          = UU - movingU
                vv          = VV - movingV
                ww          = WW - movingW

                # statistical data
                meanU    = np.mean(UU,axis = 0).reshape((1,3))
                meanW    = np.mean(WW,axis = 0).reshape((1,3))
                meanDir  = np.mean(DD,axis = 0).reshape((1,3))
                stdu_mea = np.std(uu,axis = 0).reshape((1,3))
                stdv_mea = np.std(vv,axis = 0).reshape((1,3))
                stdw_mea = np.std(ww,axis = 0).reshape((1,3))
                Iu_mea   = np.divide(stdu_mea,meanU)
                Iv_mea   = np.divide(stdv_mea,meanU)
                Iw_mea   = np.divide(stdw_mea,meanU)

                U[flag_case,:]      = meanU
```

```
                Dir[flag_case,:]    = meanDir
                stdu[flag_case,:]   = stdu_mea
                Iu[flag_case,:]     = Iu_mea
                Iv[flag_case,:]     = Iv_mea
                Iw[flag_case,:]     = Iw_mea

                autocorr_u = np.zeros(uu.shape)
                autocorr_v = np.zeros(vv.shape)
                autocorr_w = np.zeros(ww.shape)

                for i in range (0,3):
                    autocorr_u[:,i] = sm.tsa.acf(uu[:,i], nlags = 6000)
                    autocorr_v[:,i] = sm.tsa.acf(vv[:,i], nlags = 6000)
                    autocorr_w[:,i] = sm.tsa.acf(ww[:,i], nlags = 6000)

                indu_zc_A = zero_crossing(autocorr_u[:,0])[0]
                indu_zc_B = zero_crossing(autocorr_u[:,1])[0]
                indu_zc_C = zero_crossing(autocorr_u[:,2])[0]

                indv_zc_A = zero_crossing(autocorr_v[:,0])[0]
                indv_zc_B = zero_crossing(autocorr_v[:,1])[0]
                indv_zc_C = zero_crossing(autocorr_v[:,2])[0]

                indw_zc_A = zero_crossing(autocorr_w[:,0])[0]
                indw_zc_B = zero_crossing(autocorr_w[:,1])[0]
                indw_zc_C = zero_crossing(autocorr_w[:,2])[0]

                y_u_A = autocorr_u[0:indu_zc_A+1,0].reshape((indu_zc_A+1,1))
                y_u_B = autocorr_u[0:indu_zc_B+1,1].reshape((indu_zc_B+1,1))
                y_u_C = autocorr_u[0:indu_zc_C+1,2].reshape((indu_zc_C+1,1))

                y_v_A = autocorr_v[0:indv_zc_A+1,0].reshape((indv_zc_A+1,1))
                y_v_B = autocorr_v[0:indv_zc_B+1,1].reshape((indv_zc_B+1,1))
                y_v_C = autocorr_v[0:indv_zc_C+1,2].reshape((indv_zc_C+1,1))

                y_w_A = autocorr_w[0:indw_zc_A+1,0].reshape((indw_zc_A+1,1))
                y_w_B = autocorr_w[0:indw_zc_B+1,1].reshape((indw_zc_B+1,1))
                y_w_C = autocorr_w[0:indw_zc_C+1,2].reshape((indw_zc_C+1,1))

                plt.close("all")
                fig = plt.figure(figsize=(12, 10))
                ax = plt.subplot(111)
                ax.plot(np.arange(6000),autocorr_u[:,0], 'b-')
                ref = np.linspace(0,10000,100)
                zero_ref = np.zeros(ref.shape)
                ax.plot(ref,zero_ref, 'k-')
                ax.plot(617,0,'ro')
                plt.xlabel(r'Time lags', fontsize=20)
                plt.ylabel(r'$R_u(\tau)$', fontsize=20)
                plt.legend(loc = 'upper right')
                ax.set_xlim(0,6001)
                ax.set_ylim(autocorr_u[:,0].min()*1.1,autocorr_u[:,0].max()*1.1)
                g1 = plt.grid(b=True, which='major', color='k', linestyle='-', linewidth=0.35)
                g2 = plt.grid(b=True, which='minor', color='k', linestyle='-', linewidth=0.1)
                ax.tick_params(axis='both', labelsize=16)
                plt.minorticks_on()
                plt.show()

                path      = 'D:/DATA/Plots/Results summary/Length scales/'
                save_tite = 'Auto-correlation u Osp1 sensor No.' + str(1) + '.png'
                fig.tight_layout(rect=[0, 0, 1, 1])
                fig.savefig(path + save_tite)

                Lux[flag_case,0] = abs(meanU[0,0])*np.trapz(np.transpose(y_u_A))
                Lux[flag_case,1] = abs(meanU[0,1])*np.trapz(np.transpose(y_u_B))
                Lux[flag_case,2] = abs(meanU[0,2])*np.trapz(np.transpose(y_u_C))

                Lvx[flag_case,0] = abs(meanU[0,0])*np.trapz(np.transpose(y_v_A))
                Lvx[flag_case,1] = abs(meanU[0,1])*np.trapz(np.transpose(y_v_B))
                Lvx[flag_case,2] = abs(meanU[0,2])*np.trapz(np.transpose(y_v_C))

                Lwx[flag_case,0] = abs(meanU[0,0])*np.trapz(np.transpose(y_w_A))
                Lwx[flag_case,1] = abs(meanU[0,1])*np.trapz(np.transpose(y_w_B))
                Lwx[flag_case,2] = abs(meanU[0,2])*np.trapz(np.transpose(y_w_C))
```

```
#%% summarize the results for each event
        result={}
        result['date_start'] = time_s1.strftime("%Y-%m-%d %H:%M")
        result['date_end']   = time_e1.strftime("%Y-%m-%d %H:%M")
        result['U']          = U
        result['Dir']        = Dir
        result['stdu']       = stdu
        result['Iu']         = Iu
        result['Iv']         = Iv
        result['Iw']         = Iw
        result['Lux']        = Lux
        result['Lvx']        = Lvx
        result['Lwx']        = Lwx

        path      = 'D:/DATA/Results/Osp1 Length scale 10min/'

        save_tite = 'Storm No.' + str(event+1) + '_length scale Osp1 sensor No.' + str(3) + '_'\
                     + str(time_s1.year) + '_' + str(time_s1.month) + '_' + str(time_s1.day)\
                         + '_' + str(time_s1.hour)

        with open(path + save_tite + '_smooth_json.txt', "w") as outfile:
            json.dump(result, outfile, cls=NumpyEncoder)
```

A4 Spectra fitting

```
# -*- coding: utf-8 -*-
"""
Created on Thu Sep 21 08:44:42 2020
Storm data spectral fitting

@author: junwan
"""

import numpy as np
import scipy.io as spio
import matplotlib
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
import scipy.signal as signal
from   scipy.optimize import curve_fit
from   scipy.optimize import least_squares
import re
import pandas as pd
from   datetime import datetime
from   datetime import timedelta
from scipy.stats import binned_statistic
import os.path
import matplotlib.dates as mdates
import json
import pickle
import transformations      as TF
import statsmodels.api as sm
from scipy import integrate

from sklearn.neighbors import KernelDensity
from scipy.signal import argrelextrema

def running_mean(x, N):
    cumsum = np.cumsum(np.insert(x, 0, 0))
    return (cumsum[N:] - cumsum[:-N]) / float(N)
def datenum(d):
    return 366 + d.toordinal() + (d - datetime.fromordinal(d.toordinal())).total_seconds()/(24*60*60)
class NumpyEncoder(json.JSONEncoder):
    def default(self, obj):
        if isinstance(obj, np.ndarray):
            return obj.tolist()
        return json.JSONEncoder.default(self, obj)

# loop over month
def month_year_iter( start_month, start_year, end_month, end_year ):
```

```
    ym_start= 12*start_year + start_month - 1
    ym_end= 12*end_year + end_month
    loop=[]
    for ym in range( ym_start, ym_end ):
        loop = np.append(loop,divmod( ym, 12 ))
        #y[ym], m[ym] = divmod( ym, 12 )
    loop=loop.reshape((int(loop.size/2),2))
    return loop

# function to calculate the wind direciton and tranformation matrix from global XYZ to local wind uvw
# local v axis is conti-clockwise 90 deg with respect to u axis
# gloabl X is pointing east and global Y is pointing north
def windXY2dir(X,Y):
    dir_W  =  np.arctan2(X,Y)
    if dir_W<0:
        dir_W=dir_W+2*np.pi
    G_x = [1,0,0]
    G_y = [0,1,0]
    G_z = [0,0,1]
    S_u     = [np.sin(dir_W),        np.cos(dir_W),        0]
    S_v     = [np.sin(dir_W-np.pi/2),np.cos(dir_W-np.pi/2),0]
    S_w     = [0,0,1]
    # transformation matrix from XYZ to UVW
    TT      = TF.T_xyzXYZ(G_x, G_y, G_z, S_u, S_v, S_w, dim='3x3')
    return (dir_W,TT)

def wind_dir(X,Y):
    dir_W  =  np.arctan2(X,Y)
    dir_W[dir_W<0] = dir_W[dir_W<0]+2*np.pi
    return (dir_W)

# function to calculate the wind direction, mean wind speed, local wind uvw components and the time
tags
def windXYZ2uvw(data):
    # calculate the non-mean for each component of the wind components in global XYZ within starting
and ending time
    data_temp = np.nanmean(data,0)
    if ~np.isnan(data.any()):
        [dir_w,tt_w] = windXY2dir(data_temp[0],data_temp[1])
        U            = np.matmul(tt_w.T , data_temp)[0]
        uvw          = np.matmul(tt_w.T , data.T).T
        return(dir_w,U,uvw)

# round time to 1 minute
def round_time(dt=None, round_to=60):
   if dt == None:
       dt = datetime.now()
   seconds = (dt - dt.min).seconds
   rounding = (seconds+round_to/2) // round_to * round_to
   return dt + timedelta(0,rounding-seconds,-dt.microsecond)

# Find zero-crossings
def zero_crossing(data):
    return np.where(np.diff(np.sign(data)))[0]

regex_num  = r"[-+]?[.]?[\d]+(?:\d\d\d)*[\.]?\d*(?:[eE][-+]?\d+)?"

z0=0.01   # roughness lengthscale
delta_ms         = timedelta(milliseconds=100)
delta_2d         = timedelta(days=2)
delta_30min      = timedelta(minutes=30)

#%%
cases = pd.read_excel('Storm events_new.xlsx', sheet_name = 'Clustered wind storms sort')
delta_1d         = mdates.date2num(datetime(2,1,2,1,0))-mdates.date2num(datetime(2,1,1,1,0))
delta_1h         = mdates.date2num(datetime(2,1,1,2,0))-mdates.date2num(datetime(2,1,1,1,0))
delta_10min      = mdates.date2num(datetime(2,1,1,1,10))-mdates.date2num(datetime(2,1,1,1,0))

#%%
# for event in range(0,cases['Time_storm'].size):
# for event in (53, 68, 162, 166, 216, 273, 280, 283, 286, 290, 291, 292, 293, 294, 299, 303):
for event in list(range(0,53)) + list(range(54,68)) + list(range(69,162)) + list(range(163,166)) +
list(range(167,216)) + list(range(217,273)) + list(range(274,280)) + list(range(282,283)) +
```

```
list(range(284,286)) + list(range(287,290)) + list(range(295,299)) + list(range(300,303)) +
list(range(304,306)):
    # event = 284
    time = datetime.strptime(cases['Time_storm'][event], '%Y-%m-%d %H:%M:%S%z').replace(tzinfo = None)

    # starting and ending time of analyzed data window (based on storm mean wind speed time history)
    time_s1 = datetime.strptime(cases['Time_s1'][event], '%Y-%m-%d %H:%M:%S%z').replace(tzinfo = None)
    time_e1 = datetime.strptime(cases['Time_e1'][event], '%Y-%m-%d %H:%M:%S%z').replace(tzinfo = None)

    # Osp1 height info
    info_osp1 = pd.read_excel('D:/DATA/files_osp1.xls', sheet_name = 'Height')

    if os.path.isfile('D:/DATA/osp1/' + 'Osp1_' + str(time.year) + '_' + str(time.month) + '_' +
str(time.day) + '_storm_raw_data.pkl'):
        with open('D:/DATA/osp1/' + 'Osp1_' + str(time.year) + '_' + str(time.month) + '_' +
str(time.day) + '_storm_' + 'Osp1_1h' + '_ins.pkl', 'rb') as f:
            Osp1_1h_ins  = pickle.load( f )
        with open('D:/DATA/osp1/' + 'Osp1_' + str(time.year) + '_' + str(time.month) + '_' +
str(time.day) + '_storm_' + 'Osp1_1h' + '_mean.pkl', 'rb') as f:
            Osp1_1h_mean = pickle.load( f )
        with open('D:/DATA/osp1/' + 'Osp1_' + str(time.year) + '_' + str(time.month) + '_' +
str(time.day) + '_storm_temp_' + 'Osp1_1h' + '_ins.pkl', 'rb') as f:
            Osp1_1h_temp_ins  = pickle.load( f )

        # number of hourly events
        num_case = int(np.round((mdates.date2num(time_e1) - mdates.date2num(time_s1))/delta_1h))
        # num_case = 24

        U           = np.zeros((num_case,3)) # lateral mean wind speed
        Dir         = np.zeros((num_case,3)) # mean wind direction
        stdu        = np.zeros((num_case,3)) # standard deviation of turbulence component u
        Iu          = np.zeros((num_case,3)) # lateral turbulence intensity
        Iv          = np.zeros((num_case,3)) # longitudinal turbulence intensity
        Iw          = np.zeros((num_case,3)) # vertical turbulence intensity
        u_f         = np.zeros((num_case,3)) # friction velocity
        beta        = np.zeros((num_case,3)) # ratio of standard deviation to friction velocity
        alpha       = np.zeros((num_case,3)) # angle of attack
        zeta        = np.zeros((num_case,1)) # atmospheric stability

        # define the size of the fitted spectral parameters
        ANu    = np.zeros((num_case,3))
        ANu_n  = np.zeros((num_case,3))
        ANu2   = np.zeros((num_case,3))
        ANu_n2 = np.zeros((num_case,3))
        BNu2   = np.zeros((num_case,3))
        BNu_n2 = np.zeros((num_case,3))
        AHu    = np.zeros((num_case,3))
        BHu    = np.zeros((num_case,3))
        AHu_n  = np.zeros((num_case,3))
        BHu_n  = np.zeros((num_case,3))

        ANv    = np.zeros((num_case,3))
        ANv_n  = np.zeros((num_case,3))
        ANv2   = np.zeros((num_case,3))
        ANv_n2 = np.zeros((num_case,3))
        BNv2   = np.zeros((num_case,3))
        BNv_n2 = np.zeros((num_case,3))

        ANw    = np.zeros((num_case,3))
        ANw_n  = np.zeros((num_case,3))
        ANw2   = np.zeros((num_case,3))
        ANw_n2 = np.zeros((num_case,3))
        BNw2   = np.zeros((num_case,3))
        BNw_n2 = np.zeros((num_case,3))
        ABw    = np.zeros((num_case,3))
        BBw    = np.zeros((num_case,3))
        ABw_n  = np.zeros((num_case,3))
        BBw_n  = np.zeros((num_case,3))

        AKu    = np.zeros((num_case,3))
        BKu    = np.zeros((num_case,3))
        AKv    = np.zeros((num_case,3))
        BKv    = np.zeros((num_case,3))
        AKw    = np.zeros((num_case,3))
```

```
        BKw    = np.zeros((num_case,3))

        AKu_n  = np.zeros((num_case,3))
        BKu_n  = np.zeros((num_case,3))
        AKv_n  = np.zeros((num_case,3))
        BKv_n  = np.zeros((num_case,3))
        AKw_n  = np.zeros((num_case,3))
        BKw_n  = np.zeros((num_case,3))

#%%
        for flag_case in range(1,num_case):
            # flag_case = 1
            # make sure the starting and ending date is half a hour before and after the time_tag (to
be consistent with mean values)
            date_begin = round_time (mdates.num2date(mdates.date2num(time_s1)-0.5*delta_1h +
delta_1h*flag_case   ).replace(tzinfo=None))
            date_stop  = round_time (mdates.num2date(mdates.date2num(date_begin) + delta_1h
).replace(tzinfo=None))
            # integer hour tag
            date       = mdates.num2date(mdates.date2num(time_s1) +
delta_1h*flag_case).replace(tzinfo=None).replace(microsecond=0)
            # get the index of the starting and ending hour index in time

            # the original date number is based on matplotlib 3.1.3, so we have to create new time
series here
            # the time_new is created based on the time tag of the mean wind storm data
            date_new_start =  Osp1_1h_mean.iloc[0,0]  - delta_30min
            date_new_end   =  Osp1_1h_mean.iloc[-1,0] + delta_30min  -  delta_ms
            time_new = pd.date_range(date_new_start,date_new_end,freq = '100L')
            time_new = pd.Series(mdates.date2num(time_new))

            # corresponding time tag of each hourly event
            num_s     = time_new.index[time_new == mdates.date2num(date_begin)].tolist()
            num_e     = time_new.index[time_new == mdates.date2num(date_stop)].tolist()
            Num       = Osp1_1h_mean.index[Osp1_1h_mean['Time'].dt.tz_localize(None) == date].tolist()


            # get the target time series
            tt    = time_new.iloc[num_s[0]:num_e[0]].reset_index(drop=True)

            # check the ratio of NANs in data, if it's more than 10%, we do not work on it. otherwise,
we interpolate
            nan_ratio =
Osp1_1h_ins.loc[num_s[0]:num_e[0],['A_u','B_u','C_u']].isnull().sum().sum()/Osp1_1h_ins.loc[num_s[0]:n
um_e[0],['A_u','B_u','C_u']].size.sum()
            if  nan_ratio < 0.05:

                # get the raw data (interpolate the data for NAN using linear method)

                dd    = np.asarray(Osp1_1h_ins.loc[num_s[0]:num_e[0] -
1,['A_dir','B_dir','C_dir']].interpolate(axis = 0,limit_direction = 'both'))
                UU    = np.asarray(Osp1_1h_ins.loc[num_s[0]:num_e[0] -
1,['A_u','B_u','C_u']].interpolate(axis = 0,limit_direction = 'both'))
                VV    = np.asarray(Osp1_1h_ins.loc[num_s[0]:num_e[0] -
1,['A_v','B_v','C_v']].interpolate(axis = 0,limit_direction = 'both'))
                WW    = np.asarray(Osp1_1h_ins.loc[num_s[0]:num_e[0] -
1,['A_w','B_w','C_w']].interpolate(axis = 0,limit_direction = 'both'))

                TT    = np.asarray(Osp1_1h_temp_ins.loc[num_s[0]:num_e[0] -
1,['A_temp']].interpolate(axis = 0,limit_direction = 'both')) + 273.15

                # get the mean value (10 MINS AVERAGING)
                dt          = 0.1
                Nwin        = round(10*60./dt)
                DD          = np.asarray(pd.DataFrame(dd).rolling(Nwin,min_periods = 1).mean())
                movingU     = np.asarray(pd.DataFrame(UU).rolling(Nwin,min_periods = 1).mean())
                movingV     = np.asarray(pd.DataFrame(VV).rolling(Nwin,min_periods = 1).mean())
                movingW     = np.asarray(pd.DataFrame(WW).rolling(Nwin,min_periods = 1).mean())

                T           = np.asarray(pd.DataFrame(TT).rolling(Nwin,min_periods = 1).mean()) +
273.15

                # get the turbulent components
                uu          =  UU - movingU
```

```
                vv          =  VV - movingV
                ww          =  WW - movingW

                # friction velocity
                for i in range(0, 3):
                    u_f[flag_case,i] = np.power(np.power(np.cov(uu[:,i],ww[:,i])[0,1],2) +
np.power(np.cov(vv[:,i],ww[:,i])[0,1],2),1/4)

                # atmospheric stability
                w           = ww - np.mean(ww)
                theta       = TT - np.mean(TT)

                flux = np.mean(w[:,0].reshape(36000,1)*theta)
                zeta[flag_case,0] = np.divide(-
9.81*0.4*info_osp1.iloc[0,0]*flux,np.mean(T)*np.power(u_f[flag_case,0],3))

                # statistical data
                meanU    = np.mean(UU,axis = 0).reshape((1,3))
                meanW    = np.mean(WW,axis = 0).reshape((1,3))
                meanDir  = np.mean(DD,axis = 0).reshape((1,3))
                stdu_mea = np.std(uu,axis = 0).reshape((1,3))
                stdv_mea = np.std(vv,axis = 0).reshape((1,3))
                stdw_mea = np.std(ww,axis = 0).reshape((1,3))
                Iu_mea   = np.divide(stdu_mea,meanU)
                Iv_mea   = np.divide(stdv_mea,meanU)
                Iw_mea   = np.divide(stdw_mea,meanU)
                temp_b   = np.divide(stdu_mea,u_f[flag_case,:])
                temp_a   = np.arctan2(meanW,meanU)*(180/np.pi)

                U[flag_case,:]      = meanU
                Dir[flag_case,:]    = meanDir
                stdu[flag_case,:]   = stdu_mea
                Iu[flag_case,:]     = Iu_mea
                Iv[flag_case,:]     = Iv_mea
                Iw[flag_case,:]     = Iw_mea
                beta[flag_case,:]   = temp_b
                alpha[flag_case,:]  = temp_a

                # reference data
                z0       = 0.01;
                Lux_ref  = 100*np.power((np.asarray(info_osp1.iloc[0,:])/10),0.3);
                Lvx_ref  = 1/4*Lux_ref;
                Lwx_ref  = 1/12*Lux_ref;
                Lux_ref  = Lux_ref.reshape(1,(np.size(Lux_ref)))
                Lvx_ref  = Lvx_ref.reshape(1,(np.size(Lux_ref)))
                Lwx_ref  = Lwx_ref.reshape(1,(np.size(Lux_ref)))
                Iu_ref   =
(1./np.log(np.asarray(info_osp1.iloc[0,:])/z0)).reshape((1,np.size(Lux_ref)))
                Iv_ref   = 3/4.*Iu_ref;
                Iw_ref   = 1/2.*Iu_ref;
                stdu_ref = np.multiply(meanU,Iu_ref)
                stdv_ref = np.multiply(meanU,Iv_ref)
                stdw_ref = np.multiply(meanU,Iw_ref)

                # spectral analysis of the wind time series
                Nblock   = 6; # Nblock = number of overlapping segment. Friday 07/06/2019, I have used
Nblock = 2. here I use 6 blocks to have segments of 30 min
                N        = np.shape(uu)[0];
                Nfft     = np.round(N/Nblock);
                fs       = 1/dt; # sampling frequency

                f, Su = signal.welch(np.transpose(uu), fs, window = 'hann',  nperseg = Nfft, noverlap
= None, nfft = None, detrend = 'constant', return_onesided = True)
                f, Sv = signal.welch(np.transpose(vv), fs, window = 'hann',  nperseg = Nfft, noverlap
= None, nfft = None, detrend = 'constant', return_onesided = True)
                f, Sw = signal.welch(np.transpose(ww), fs, window = 'hann',  nperseg = Nfft, noverlap
= None, nfft = None, detrend = 'constant', return_onesided = True)

                f  =  f.reshape((np.size(f),1))
                Su =  np.transpose(Su)
                Sv =  np.transpose(Sv)
                Sw =  np.transpose(Sw)

                # bin average the spectra in logspace
```

```
                num_log = 60
                f_s_start = np.log10(1/600)
                f_s_end   = np.log10(5)

                f_s  = np.logspace(f_s_start,f_s_end,num_log + 1).reshape(num_log + 1,1)
                Su_s = np.zeros((np.size(f_s) - 1,3))
                Sv_s = np.zeros((np.size(f_s) - 1,3))
                Sw_s = np.zeros((np.size(f_s) - 1,3))

                for i in range(0, 3):
                    Su_s[:,i], edges, _ = binned_statistic(f[:,0],Su[:,i], statistic = 'mean', bins =
f_s[:,0])
                    Sv_s[:,i], edges, _ = binned_statistic(f[:,0],Sv[:,i], statistic = 'mean', bins =
f_s[:,0])
                    Sw_s[:,i], edges, _ = binned_statistic(f[:,0],Sw[:,i], statistic = 'mean', bins =
f_s[:,0])
                f_s  = edges[:-1] + np.diff(edges)/2

                nan_index = np.isnan(Su_s)
                f_s       = f_s [~nan_index[:,0]]
                Su_s      = Su_s[~nan_index].reshape((np.size(f_s),3))
                Sv_s      = Sv_s[~nan_index].reshape((np.size(f_s),3))
                Sw_s      = Sw_s[~nan_index].reshape((np.size(f_s),3))
                f_s       = f_s.reshape((np.size(f_s),1))

                # fit the wind spectra u<0.1Hz,v<0.1Hz,w<0.5Hz
                fu_s = 0.01
                fu_e = 1
                fv_s = 0.01
                fv_e = 1
                fw_s = 0.1
                fw_e = 1

                indfu_e  =  np.where(abs(f_s-fu_e) == min(abs(f_s-fu_e)))[0]
                indfv_e  =  np.where(abs(f_s-fv_e) == min(abs(f_s-fv_e)))[0]
                indfw_e  =  np.where(abs(f_s-fw_e) == min(abs(f_s-fw_e)))[0]
                indfu_s  =  np.where(abs(f_s-fu_s) == min(abs(f_s-fu_s)))[0]
                indfv_s  =  np.where(abs(f_s-fv_s) == min(abs(f_s-fv_s)))[0]
                indfw_s  =  np.where(abs(f_s-fw_s) == min(abs(f_s-fw_s)))[0]

                fre_u_n  =  np.dot(f_s,Lux_ref/meanU)
                fre_v_n  =  np.dot(f_s,Lvx_ref/meanU)
                fre_w_n  =  np.dot(f_s,Lwx_ref/meanU)

                Sf_u_n   = np.multiply(np.dot(f_s,1/np.power(stdu_mea,2)),Su_s)
                Sf_v_n   = np.multiply(np.dot(f_s,1/np.power(stdv_mea,2)),Sv_s)
                Sf_w_n   = np.multiply(np.dot(f_s,1/np.power(stdw_mea,2)),Sw_s)

                # range of the moving mean velocity
                delta_U = (np.max(movingU,0) - np.min(movingU,0)).reshape((1,3))
                delta_Dir = (np.max(DD,0) - np.min(DD,0)).reshape((1,3))

                # i here represents the sensor number for each station
                for i in range(0, 3):
                    if delta_U[0,i] < 0.3*meanU[0,i] and delta_Dir[0,i] < 20 and meanU[0,i] > 6:    #
this is to check the stationarity range of moving mean should be smaller than +-10% of the static mean

                        f_n = np.dot(f_s,info_osp1.iloc[0,i]/meanU)

                        #%
                        # define the wind spectra
                        global temp, temp1, tempu, tempu1, tempv, tempv1, tempw, tempw1
                        temp       = info_osp1.iloc[0,i]*np.power(u_f[flag_case,i],2)/meanU[0,i];
# z*u_f^2/U
                        temp1      = info_osp1.iloc[0,i]/meanU[0,i];
# z/U
                        tempu      = Lux_ref[0,i]*np.power(stdu_mea[0,i],2)/meanU[0,i];
# Lu*stdu^2/U
                        tempu1     = Lux_ref[0,i]/meanU[0,i];
# L/U
                        tempu_ref  = Lux_ref[0,i]*np.power(stdu_ref[0,i],2)/meanU[0,i];
# L*std^2/U
                        tempu1_ref = Lux_ref[0,i]/meanU[0,i];
# L/U
```

```
                        tempv      = Lvx_ref[0,i]*np.power(stdv_mea[0,i],2)/meanU[0,i];
# Lv*stdv^2/U
                        tempv1     = Lvx_ref[0,i]/meanU[0,i];
# L/U
                        tempv_ref  = Lvx_ref[0,i]*np.power(stdv_ref[0,i],2)/meanU[0,i];
# L*std^2/U
                        tempv1_ref = Lvx_ref[0,i]/meanU[0,i];
# L/U
                        tempw      = Lwx_ref[0,i]*np.power(stdw_mea[0,i],2)/meanU[0,i];
# Lw*stdw^2/U
                        tempw1     = Lwx_ref[0,i]/meanU[0,i];
# L/U
                        tempw_ref  = Lwx_ref[0,i]*np.power(stdw_ref[0,i],2)/meanU[0,i];
# L*std^2/U
                        tempw1_ref = Lwx_ref[0,i]/meanU[0,i];
# L/U


                        def N400u_ref(f, A):
                            return np.divide(A*tempu_ref,np.power((1+1.5*A*tempu1_ref*f),5/3))
                        def N400v_ref(f, A):
                            return np.divide(A*tempv_ref,np.power(1+1.5*A*tempv1_ref*f,5/3))
                        def N400w_ref(f, A):
                            return np.divide(A*tempw_ref,np.power(1+1.5*A*tempw1_ref*f,5/3))

                        def N400u_1par(f, A):
                            return np.divide(A*tempu,np.power((1+1.5*A*tempu1*f),5/3))
                        def N400_n_1par(f_hat, A):
                            return np.divide(A*f_hat,np.power(1+1.5*A*f_hat,5/3))
                        def N400u_2par(f, A, B):
                            return np.divide(A*tempu,np.power(1+1.5*B*tempu1*f,5/3))
                        def N400_n_2par(f_hat, A, B):
                            return np.divide(A*f_hat,np.power(1+1.5*B*f_hat,5/3))

                        def Harris_u(f, A, B):
                            return
np.divide(A*tempu,np.float_power(1+B*np.float_power(f*tempu1,2),5/6))
                        def Harris_u_n(f_hat, A, B):
                            return np.divide(A*f_hat,np.power(1+B*np.power(f_hat,2),5/6))

                        def N400v_1par(f, A):
                            return np.divide(A*tempv,np.power(1+1.5*A*tempv1*f,5/3))
                        def N400v_2par(f, A, B):
                            return np.divide(A*tempv,np.power(1+1.5*B*tempv1*f,5/3))
                        def N400w_1par(f, A):
                            return np.divide(A*tempw,np.power(1+1.5*A*tempw1*f,5/3))
                        def N400w_2par(f, A, B):
                            return np.divide(A*tempw,np.power(1+1.5*B*tempw1*f,5/3))

                        def Kaimal_u(f, A, B):
                            return np.divide(A*temp,np.power(1+B*temp1*f,5/3))
                        def Kaimal_v(f, A, B):
                            return np.divide(A*temp,np.power(1+B*temp1*f,5/3))
                        def Kaimal_w(f, A, B):
                            return np.divide(A*temp,1+np.multiply(B,np.power(temp1*f,5/3)))
                        def Kaimal_u_n(f_hat, A, B):
                            return np.divide(A*f_hat,np.power(1+B*f_hat,5/3))
                        def Kaimal_v_n(f_hat, A, B):
                            return np.divide(A*f_hat,np.power(1+B*f_hat,5/3))
                        def Kaimal_w_n(f_hat, A, B):
                            return np.divide(A*f_hat,1+np.multiply(B,np.power(f_hat,5/3)))

                        def Froya_u(f, Uref, z):
                            n     = 0.468
                            f_hat = 172*np.power((z/10),2/3)*np.power((Uref/10),-0.75)*f
                            Su    =
320*np.divide(np.power((Uref/10),2)*np.power((z/10),0.45),np.power(1+np.power(f_hat,n),5/(3*n)))
                            return Su

                        def Busch_Panofsky_w_ref(f,A,B):
                            return np.divide(A*tempw_ref,1+B*np.power(tempw1_ref*f,5/3))
                        def Busch_Panofsky_w_n_ref(f_hat,A,B):
                            return np.divide(A*f_hat,1+B*np.power(f_hat,5/3))
```

```
                        def Sfn_from_Sf(f,Sf):
                            var= np.trapz(Sf,f,axis=0)
                            Sfn= np.multiply(np.dot(f, 1/var).reshape(np.size(f),1),Sf)
                            return Sfn

                        # collect the fitted parameters
                        ANu[flag_case,i], pcov = curve_fit(N400u_1par, f_s[indfu_s[0]:indfu_e[0],0],
Su_s[indfu_s[0]:indfu_e[0],i], p0=6, maxfev=10000, bounds=(0, np.inf))
                        ANu_n[flag_case,i], pcovn = curve_fit(N400_n_1par,
fre_u_n[indfu_s[0]:indfu_e[0],i], Sf_u_n [indfu_s[0]:indfu_e[0],i], p0=6, maxfev=10000, bounds=(0,
np.inf))
                        [ANu2[flag_case,i],BNu2[flag_case,i]], pcov = curve_fit(N400u_2par,
f_s[indfu_s[0]:indfu_e[0],0], Su_s[indfu_s[0]:indfu_e[0],i], p0=(6,6), maxfev=10000, bounds=(0,
np.inf))
                        [ANu_n2[flag_case,i],BNu_n2[flag_case,i]], pcovn = curve_fit(N400_n_2par,
fre_u_n[indfu_s[0]:indfu_e[0],i], Sf_u_n [indfu_s[0]:indfu_e[0],i], p0=(6,6), maxfev=10000, bounds=(0,
np.inf))
                        [AHu[flag_case,i],BHu[flag_case,i]], pcovn = curve_fit(Harris_u,
f_s[indfu_s[0]:indfu_e[0],0], Su_s[indfu_s[0]:indfu_e[0],i], p0=(4,70), maxfev=10000, bounds=(0,
np.inf))
                        [AHu_n[flag_case,i],BHu_n[flag_case,i]], pcovn = curve_fit(Harris_u_n,
fre_u_n[indfu_s[0]:indfu_e[0],i], Sf_u_n [indfu_s[0]:indfu_e[0],i], p0=(4,70), maxfev=10000,
bounds=(0, np.inf))

                        ANv[flag_case,i], pcov = curve_fit(N400v_1par, f_s[indfv_s[0]:indfv_e[0],0],
Sv_s[indfv_s[0]:indfv_e[0],i], p0=6, maxfev=10000, bounds=(0, np.inf))
                        ANv_n[flag_case,i], pcovn = curve_fit(N400_n_1par,
fre_v_n[indfv_s[0]:indfv_e[0],i], Sf_v_n [indfv_s[0]:indfv_e[0],i], p0=6, maxfev=10000, bounds=(0,
np.inf))
                        [ANv2[flag_case,i],BNv2[flag_case,i]], pcov = curve_fit(N400v_2par,
f_s[indfv_s[0]:indfv_e[0],0], Sv_s[indfv_s[0]:indfv_e[0],i], p0=(6,6), maxfev=10000, bounds=(0,
np.inf))
                        [ANv_n2[flag_case,i],BNv_n2[flag_case,i]], pcovn = curve_fit(N400_n_2par,
fre_v_n[indfv_s[0]:indfv_e[0],i], Sf_v_n [indfv_s[0]:indfv_e[0],i], p0=(6,6), maxfev=10000, bounds=(0,
np.inf))

                        ANw[flag_case,i], pcov = curve_fit(N400w_1par, f_s[indfw_s[0]:indfw_e[0],0],
Sw_s[indfw_s[0]:indfw_e[0],i], p0=6, maxfev=10000, bounds=(0, np.inf))
                        ANw_n[flag_case,i], pcovn = curve_fit(N400_n_1par,
fre_w_n[indfw_s[0]:indfw_e[0],i], Sf_w_n [indfw_s[0]:indfw_e[0],i], p0=6, maxfev=10000, bounds=(0,
np.inf))
                        [ANw2[flag_case,i],BNw2[flag_case,i]], pcov = curve_fit(N400w_2par,
f_s[indfw_s[0]:indfw_e[0],0], Sw_s[indfw_s[0]:indfw_e[0],i], p0=(6,6), maxfev=10000, bounds=(0,
np.inf))
                        [ANw_n2[flag_case,i],BNw_n2[flag_case,i]], pcovn = curve_fit(N400_n_2par,
fre_w_n[indfw_s[0]:indfw_e[0],i], Sf_w_n [indfw_s[0]:indfw_e[0],i], p0=(6,6), maxfev=10000, bounds=(0,
np.inf))

                        [AKu[flag_case,i],BKu[flag_case,i]], pcov = curve_fit(Kaimal_u,
f_s[indfu_s[0]:indfu_e[0],0], Su_s[indfu_s[0]:indfu_e[0],i], p0=(105,33), maxfev=10000, bounds=(0,
np.inf))
                        [AKv[flag_case,i],BKv[flag_case,i]], pcov = curve_fit(Kaimal_v,
f_s[indfv_s[0]:indfv_e[0],0], Sv_s[indfv_s[0]:indfv_e[0],i], p0=(17,9.5), maxfev=10000, bounds=(0,
np.inf))
                        [AKw[flag_case,i],BKw[flag_case,i]], pcov = curve_fit(Kaimal_w,
f_s[indfw_s[0]:indfw_e[0],0], Sw_s[indfw_s[0]:indfw_e[0],i], p0=(2,5.3), maxfev=10000, bounds=(0,
np.inf))

                        [AKu_n[flag_case,i],BKu_n[flag_case,i]], pcovn = curve_fit(Kaimal_u_n,
f_n[indfu_s[0]:indfu_e[0],i], Sf_u_n [indfu_s[0]:indfu_e[0],i], p0=(105,33), maxfev=10000, bounds=(0,
np.inf))
                        [AKv_n[flag_case,i],BKv_n[flag_case,i]], pcovn = curve_fit(Kaimal_v_n,
f_n[indfv_s[0]:indfv_e[0],i], Sf_v_n [indfv_s[0]:indfv_e[0],i], p0=(17,9.5), maxfev=10000, bounds=(0,
np.inf))
                        [AKw_n[flag_case,i],BKw_n[flag_case,i]], pcovn = curve_fit(Kaimal_w_n,
f_n[indfw_s[0]:indfw_e[0],i], Sf_w_n [indfw_s[0]:indfw_e[0],i], p0=(2,5.3), maxfev=10000, bounds=(0,
np.inf))

                        # plot the spectra in the normalized format
                        matplotlib.use('Agg')

                        # calculate the reference wind speed at 10 m height
                        f_hor      = 1/56.06*0.9
                        f_ver      = 1/6.89*0.9
```

```
                        Uref       =
meanU[0,i]*(1+np.log(np.asarray(info_osp1.iloc[0,:])[i]/10)/np.log(10/z0))
                        Su_n_froya =  Sfn_from_Sf(f_s,Froya_u(f_s[:,0], Uref,
np.asarray(info_osp1.iloc[0,:])[i]).reshape(np.size(f_s),1))
                        plt.close("all")
                        fig     = plt.figure(figsize=(16, 12))
                        ax1     = plt.subplot(311)
                        ax1.plot([fu_s, fu_s],[np.min(Su[:,i]), np.max(Su[:,i])],'k-', linewidth=2,
markeredgecolor='k',markersize=8)
                        ax1.plot([fu_e, fu_e],[np.min(Su[:,i]), np.max(Su[:,i])],'k-', linewidth=2,
markeredgecolor='k',markersize=8)
                        ax1.semilogx(f_s[:,0],Su_s[:,i],
'ks', label='$Measured$',linewidth=1, markeredgecolor='k',markersize=6)
                        ax1.semilogx(f[:,0],N400u_1par(f[:,0], ANu[flag_case,i]),
'r-', label='$Fitted$ $ N400$ $( 1 para)$',linewidth=2, markeredgecolor='k',markersize=8)
                        ax1.semilogx(f[:,0],N400u_2par(f[:,0], ANu2[flag_case,i],BNu2[flag_case,i]),
'g-', label='$Fitted$ $ N400$ $( 2 para)$',linewidth=2, markeredgecolor='k',markersize=8)
                        ax1.semilogx(f[:,0],Harris_u(f[:,0], AHu[flag_case,i], BHu[flag_case,i]),
'y-', label='$Fitted$ $ Harris$ $( 2 para)$',linewidth=2, markeredgecolor='k',markersize=8)
                        ax1.semilogx(f[:,0],Kaimal_u(f[:,0], AKu[flag_case,i],BKu[flag_case,i]),   'm-
', label='$Fitted$ $ Kaimal$ $( 2 para)$',linewidth=2, markeredgecolor='k',markersize=8)
                        ax1.semilogx(f[:,0],N400u_ref(f[:,0], 6.8),
'b--',label='$N400$ $ ref$',linewidth=2, markeredgecolor='k',markersize=8)
                        ax1.semilogx(f[:,0],Froya_u(f[:,0], Uref, np.asarray(info_osp1.iloc[0,:])[i]),
'c--',label='$Frøya$ $ ref$',linewidth=2, markeredgecolor='k',markersize=8)
                        ax1.fill_between(f[:,0],0,1000,where= f[:,0]>f_hor,facecolor='grey',
alpha=0.5)
                        ax1.set_xlim(0.005, 2)
                        ax1.set_ylim(0, 1.2*np.max(Su_s[2:-1,i]))
                        plt.xlabel(r'$f$ ($ Hz)$', fontsize=20)
                        plt.ylabel(r'$S_{u}$ ($m^2 s^{-1})$', fontsize=20)
                        fig.suptitle('Event No.' + str(event+1)+ ' Dir=' +
str(np.round(meanDir[0,i],1)) + '$^o$'+'  Osp1 at height ' +
str(np.round(np.asarray(info_osp1.iloc[0,:])[i],1)) + 'm ' \
                                    + 'from '  + date_begin.strftime("%Y-%m-%d %H:%M") + ' to ' +
date_stop.strftime("%Y-%m-%d %H:%M"), fontsize=25)
                        plt.rc('xtick', direction='in', color='k')
                        plt.rc('ytick', direction='in', color='k')
                        plt.legend(loc='upper right',ncol=2,fontsize=16)
                        g1 = plt.grid(b=True, which='major', color='k', linestyle='-', linewidth=0.35)
                        g2 = plt.grid(b=True, which='minor', color='k', linestyle='-', linewidth=0.1)
                        ax1.tick_params(axis='both', labelsize=16)
                        plt.minorticks_on()
                        #plt.show()

                        ax2     = plt.subplot(312)
                        ax2.plot([fv_s, fv_s],[np.min(Sv[:,i]), np.max(Sv[:,i])],'k-', linewidth=2,
markeredgecolor='k',markersize=8)
                        ax2.plot([fv_e, fv_e],[np.min(Sv[:,i]), np.max(Sv[:,i])],'k-', linewidth=2,
markeredgecolor='k',markersize=8)
                        ax2.semilogx(f_s[:,0],Sv_s[:,i],
'ks', label='$Measured$',linewidth=1, markeredgecolor='k',markersize=6)
                        ax2.semilogx(f[:,0],N400v_1par(f[:,0], ANv[flag_case,i]),
'r-', label='$Fitted$ $ N400$ $( 1 para)$',linewidth=2, markeredgecolor='k',markersize=8)
                        ax2.semilogx(f[:,0],N400v_2par(f[:,0], ANv2[flag_case,i],BNv2[flag_case,i]),
'g-', label='$Fitted$ $ N400$ $( 2 para)$',linewidth=2, markeredgecolor='k',markersize=8)
                        ax2.semilogx(f[:,0],Kaimal_v(f[:,0], AKv[flag_case,i],BKv[flag_case,i]),
'm-', label='$Fitted$ $ Kaimal$ $( 2 para)$',linewidth=2, markeredgecolor='k',markersize=8)
                        ax2.semilogx(f[:,0],N400v_ref(f[:,0], 9.4),
'b--',label='$N400$ $ ref$',linewidth=2, markeredgecolor='k',markersize=8)
                        ax2.fill_between(f[:,0],0,1000,where= f[:,0]>f_hor,facecolor='grey',
alpha=0.5)
                        ax2.set_xlim(0.005, 2)
                        ax2.set_ylim(0, 1.2*np.max(Sv_s[:,i]))
                        plt.xlabel(r'$f$ ($ Hz)$', fontsize=20)
                        plt.ylabel(r'$S_{v}$ ($m^2 s^{-1})$', fontsize=20)
                        plt.rc('xtick', direction='in', color='k')
                        plt.rc('ytick', direction='in', color='k')
                        plt.legend(loc='upper right',ncol=2,fontsize=16)
                        g1 = plt.grid(b=True, which='major', color='k', linestyle='-', linewidth=0.35)
                        g2 = plt.grid(b=True, which='minor', color='k', linestyle='-', linewidth=0.1)
                        ax2.tick_params(axis='both', labelsize=16)
                        plt.minorticks_on()
```

```
                        #plt.show()

                        ax3     = plt.subplot(313)
                        ax3.plot([fw_s, fw_s],[np.min(Sw[:,i]), np.max(Sw[:,i])],'k-', linewidth=2,
markeredgecolor='k',markersize=8)
                        ax3.plot([fw_e, fw_e],[np.min(Sw[:,i]), np.max(Sw[:,i])],'k-', linewidth=2,
markeredgecolor='k',markersize=8)
                        ax3.semilogx(f_s[:,0],Sw_s[:,i],
'ks', label='$Measured$',linewidth=1, markeredgecolor='k',markersize=6)
                        ax3.plot(f[:,0],N400w_1par(f[:,0], ANw[flag_case,i]),                       'r-
', label='$Fitted$ $ N400$ $( 1 para)$',linewidth=2, markeredgecolor='k',markersize=8)
                        ax3.plot(f[:,0],N400w_2par(f[:,0], ANw2[flag_case,i],BNw2[flag_case,i]),   'g-
', label='$Fitted$ $ N400$ $( 2 para)$',linewidth=2, markeredgecolor='k',markersize=8)
                        ax3.semilogx(f[:,0],Kaimal_w(f[:,0], AKw[flag_case,i],BKw[flag_case,i]),
'm-', label='$Fitted$ $ Kaimal$ $( 2 para)$',linewidth=2, markeredgecolor='k',markersize=8)
                        ax3.plot(f[:,0],Busch_Panofsky_w_ref(f[:,0], 2.15,11.16),                  'y-
',label='$Busch-Panofsky$ $ ref$',linewidth=2, markeredgecolor='k',markersize=8)
                        ax3.plot(f[:,0],N400w_ref(f[:,0], 6.8),                                    'b--
',label='$N400$ $ ref$',linewidth=2, markeredgecolor='k',markersize=8)
                        ax3.fill_between(f[:,0],0,1000,where= f[:,0]>f_ver,facecolor='grey',
alpha=0.5)
                        ax3.set_xlim(0.01, 2)
                        ax3.set_ylim(0, 1.2*np.max(Sw_s[4:-1,i]))
                        plt.xlabel(r'$f$ ($ Hz)$', fontsize=20)
                        plt.ylabel(r'$S_{w}$ ($m^2 s^{-1})$', fontsize=20)
                        plt.rc('xtick', direction='in', color='k')
                        plt.rc('ytick', direction='in', color='k')
                        plt.legend(loc='upper right',ncol=2,fontsize=16)
                        g1 = plt.grid(b=True, which='major', color='k', linestyle='-', linewidth=0.35)
                        g2 = plt.grid(b=True, which='minor', color='k', linestyle='-', linewidth=0.1)
                        ax3.tick_params(axis='both', labelsize=16)
                        plt.minorticks_on()
                        #plt.show()

                        fig.tight_layout(rect=[0, 0, 1, 0.95])

                        path      = 'D:/DATA/Plots/Osp1 Spectra fitting/'
                        save_tite = 'Storm No.' + str(event+1) + '_spectrum fitting Osp1 sensor No.' +
str(i+1) + '_'\
                                    + str(date_begin.year) + '_' + str(date_begin.month) + '_' +
str(date_begin.day)\
                                        + '_' + str(date_begin.hour) + 'smooth.png'
                        fig.savefig(path + save_tite)

                        # N400 spectrum
                        plt.close("all")
                        fig     = plt.figure(figsize=(20, 12))
                        ax1     = plt.subplot(321)
                        ax1.plot([fu_s, fu_s],[np.min(Su[:,i]), np.max(Su[:,i])],'k-', linewidth=2,
markeredgecolor='k',markersize=8)
                        ax1.plot([fu_e, fu_e],[np.min(Su[:,i]), np.max(Su[:,i])],'k-', linewidth=2,
markeredgecolor='k',markersize=8)
                        ax1.semilogx(f_s[:,0],Su_s[:,i],
'ks', label='$Measured$',linewidth=1, markeredgecolor='k',markersize=6)
                        ax1.semilogx(f[:,0],N400u_1par(f[:,0], ANu[flag_case,i]),
'r-', label='$Fitted$ $ N400$ $( 1 para)$',linewidth=2, markeredgecolor='k',markersize=8)
                        ax1.semilogx(f[:,0],N400u_2par(f[:,0], ANu2[flag_case,i],BNu2[flag_case,i]),
'g-', label='$Fitted$ $ N400$ $( 2 para)$',linewidth=2, markeredgecolor='k',markersize=8)
                        ax1.semilogx(f[:,0],N400u_ref(f[:,0], 6.8),
'b--',label='$N400$ $ ref$',linewidth=2, markeredgecolor='k',markersize=8)
                        ax1.fill_between(f[:,0],0,1000,where= f[:,0]>f_hor,facecolor='grey',
alpha=0.5)
                        ax1.set_xlim(0.005, 2)
                        ax1.set_ylim(0, 1.2*np.max(Su_s[2:-1,i]))
                        plt.xlabel(r'$f$ ($ Hz)$', fontsize=20)
                        plt.ylabel(r'$S_{u}$ ($m^2 s^{-1})$', fontsize=20)
                        fig.suptitle('Event No.' + str(event+1)+ ' Dir=' +
str(np.round(meanDir[0,i],1)) + '$^o$'+'  Osp1 at height ' +
str(np.round(np.asarray(info_osp1.iloc[0,:])[i],1)) + 'm ' \
                                    + 'from '  + date_begin.strftime("%Y-%m-%d %H:%M") + ' to ' +
date_stop.strftime("%Y-%m-%d %H:%M"), fontsize=25)
                        plt.rc('xtick', direction='in', color='k')
                        plt.rc('ytick', direction='in', color='k')
                        plt.legend(loc='upper right',ncol=2,fontsize=16)
```

```
                        g1 = plt.grid(b=True, which='major', color='k', linestyle='-', linewidth=0.35)
                        g2 = plt.grid(b=True, which='minor', color='k', linestyle='-', linewidth=0.1)
                        ax1.tick_params(axis='both', labelsize=16)
                        plt.minorticks_on()
                        #plt.show()

                        ax2     = plt.subplot(322)
                        ax2.plot([fre_u_n[indfu_s[0],i], fre_u_n[indfu_s[0],i]],[0, 1000],'k-',
linewidth=2, markeredgecolor='k',markersize=8)
                        ax2.plot([fre_u_n[indfu_e[0],i], fre_u_n[indfu_e[0],i]],[0, 1000],'k-',
linewidth=2, markeredgecolor='k',markersize=8)
                        ax2.loglog(fre_u_n[:,i],Sf_u_n[:,i],                                    'ks',
label='$Measured$',linewidth=1, markeredgecolor='k',markersize=6)
                        ax2.loglog(fre_u_n[:,i],N400_n_1par(fre_u_n[:,i], ANu_n[flag_case,i]),
'r-', label='$Fitted$ $ N400$ $( 1 para)$',linewidth=2, markeredgecolor='k',markersize=8)
                        ax2.loglog(fre_u_n[:,i],N400_n_2par(fre_u_n[:,i],
ANu_n2[flag_case,i],BNu_n2[flag_case,i]), 'g-', label='$Fitted$ $ N400$ $( 2 para)$',linewidth=2,
markeredgecolor='k',markersize=8)
                        ax2.loglog(fre_u_n[:,i],N400_n_1par(fre_u_n[:,i], 6.8),                 'b--
',label='$N400$ $ ref$',linewidth=2, markeredgecolor='k',markersize=8)
                        ax2.fill_between(fre_u_n[:,i],0,1000,where=
fre_u_n[:,i]>f_hor*Lux_ref[0,i]/meanU[0,i],facecolor='grey', alpha=0.5)
                        ax2.set_ylim(0.6*Sf_u_n[-1,i],  1.3*np.max(Sf_u_n[:,i]))
                        ax2.set_xlim(0.01, 10)
                        plt.xlabel(r'$f*L_{ux}/\overline{U}$', fontsize=20)
                        plt.ylabel(r'$f*S_{u}$ $/\sigma_{u\_measure}^2$', fontsize=20)
                        plt.rc('xtick', direction='in', color='k')
                        plt.rc('ytick', direction='in', color='k')
                        plt.legend(loc='lower left',ncol=2,fontsize=16)
                        g1 = plt.grid(b=True, which='major', color='k', linestyle='-', linewidth=0.35)
                        g2 = plt.grid(b=True, which='minor', color='k', linestyle='-', linewidth=0.1)
                        ax2.tick_params(axis='both', labelsize=16)
                        plt.minorticks_on()
                        #plt.show()

                        ax1     = plt.subplot(323)
                        ax1.plot([fv_s, fv_s],[np.min(Sv[:,i]), np.max(Sv[:,i])],'k-', linewidth=2,
markeredgecolor='k',markersize=8)
                        ax1.plot([fv_e, fv_e],[np.min(Sv[:,i]), np.max(Sv[:,i])],'k-', linewidth=2,
markeredgecolor='k',markersize=8)
                        ax1.semilogx(f_s[:,0],Sv_s[:,i],
'ks', label='$Measured$',linewidth=1, markeredgecolor='k',markersize=6)
                        ax1.semilogx(f[:,0],N400v_1par(f[:,0], ANv[flag_case,i]),
'r-', label='$Fitted$ $ N400$ $( 1 para)$',linewidth=2, markeredgecolor='k',markersize=8)
                        ax1.semilogx(f[:,0],N400v_2par(f[:,0], ANv2[flag_case,i],BNv2[flag_case,i]),
'g-', label='$Fitted$ $ N400$ $( 2 para)$',linewidth=2, markeredgecolor='k',markersize=8)
                        ax1.semilogx(f[:,0],N400v_ref(f[:,0], 9.4),
'b--',label='$N400$ $ ref$',linewidth=2, markeredgecolor='k',markersize=8)
                        ax1.fill_between(f[:,0],0,1000,where= f[:,0]>f_hor,facecolor='grey',
alpha=0.5)
                        ax1.set_xlim(0.005, 2)
                        ax1.set_ylim(0, 1.2*np.max(Sv_s[:,i]))
                        plt.xlabel(r'$f$ ($ Hz)$', fontsize=20)
                        plt.ylabel(r'$S_{v}$ ($m^2 s^{-1})$', fontsize=20)
                        plt.rc('xtick', direction='in', color='k')
                        plt.rc('ytick', direction='in', color='k')
                        plt.legend(loc='upper right',ncol=2,fontsize=16)
                        g1 = plt.grid(b=True, which='major', color='k', linestyle='-', linewidth=0.35)
                        g2 = plt.grid(b=True, which='minor', color='k', linestyle='-', linewidth=0.1)
                        ax1.tick_params(axis='both', labelsize=16)
                        plt.minorticks_on()
                        #plt.show()

                        ax2     = plt.subplot(324)
                        ax2.plot([fre_v_n[indfv_s[0],i], fre_v_n[indfv_s[0],i]],[0, 1000],'k-',
linewidth=2, markeredgecolor='k',markersize=8)
                        ax2.plot([fre_v_n[indfv_e[0],i], fre_v_n[indfv_e[0],i]],[0, 1000],'k-',
linewidth=2, markeredgecolor='k',markersize=8)
                        ax2.loglog(fre_v_n[:,i],Sf_v_n[:,i],                                    'ks',
label='$Measured$',linewidth=1, markeredgecolor='k',markersize=6)
                        ax2.loglog(fre_v_n[:,i],N400_n_1par(fre_v_n[:,i], ANv_n[flag_case,i]),
'r-', label='$Fitted$ $ N400$ $( 1 para)$',linewidth=2, markeredgecolor='k',markersize=8)
```

```
                        ax2.loglog(fre_v_n[:,i],N400_n_2par(fre_v_n[:,i],
ANv_n2[flag_case,i],BNv_n2[flag_case,i]), 'g-', label='$Fitted$ $ N400$ $( 2 para)$',linewidth=2,
markeredgecolor='k',markersize=8)
                        ax2.loglog(fre_v_n[:,i],N400_n_1par(fre_v_n[:,i], 6.8),                 'b--
',label='$N400$ $ ref$',linewidth=2, markeredgecolor='k',markersize=8)
                        ax2.fill_between(fre_v_n[:,i],0,1000,where=
fre_v_n[:,i]>f_hor*Lvx_ref[0,i]/meanU[0,i],facecolor='grey', alpha=0.5)
                        ax2.set_ylim(0.6*Sf_v_n[-1,i], 1.3*np.max(Sf_v_n[:,i]))
                        ax2.set_xlim(0.01, 10)
                        plt.xlabel(r'$f*L_{vx}/\overline{U}$', fontsize=20)
                        plt.ylabel(r'$f*S_{v}$ $/\sigma_{v\_measure}^2$', fontsize=20)
                        plt.rc('xtick', direction='in', color='k')
                        plt.rc('ytick', direction='in', color='k')
                        plt.legend(loc='lower left',ncol=2,fontsize=16)
                        g1 = plt.grid(b=True, which='major', color='k', linestyle='-', linewidth=0.35)
                        g2 = plt.grid(b=True, which='minor', color='k', linestyle='-', linewidth=0.1)
                        ax2.tick_params(axis='both', labelsize=16)
                        plt.minorticks_on()
                        #plt.show()

                        ax1     = plt.subplot(325)
                        ax1.plot([fw_s, fw_s],[np.min(Sw[:,i]), np.max(Sw[:,i])],'k-', linewidth=2,
markeredgecolor='k',markersize=8)
                        ax1.plot([fw_e, fw_e],[np.min(Sw[:,i]), np.max(Sw[:,i])],'k-', linewidth=2,
markeredgecolor='k',markersize=8)
                        ax1.semilogx(f_s[:,0],Sw_s[:,i],
'ks', label='$Measured$',linewidth=1, markeredgecolor='k',markersize=6)
                        ax1.plot(f[:,0],N400w_1par(f[:,0], ANw[flag_case,i]),                    'r-
', label='$Fitted$ $ N400$ $( 1 para)$',linewidth=2, markeredgecolor='k',markersize=8)
                        ax1.plot(f[:,0],N400w_2par(f[:,0], ANw2[flag_case,i],BNw2[flag_case,i]),   'g-
', label='$Fitted$ $ N400$ $( 2 para)$',linewidth=2, markeredgecolor='k',markersize=8)
                        ax1.plot(f[:,0],N400w_ref(f[:,0], 6.8),                                  'b--
',label='$N400$ $ ref$',linewidth=2, markeredgecolor='k',markersize=8)
                        ax1.fill_between(f[:,0],0,1000,where= f[:,0]>f_ver,facecolor='grey',
alpha=0.5)
                        ax1.set_xlim(0.01, 2)
                        ax1.set_ylim(0, 1.2*np.max(Sw_s[4:-1,i]))
                        plt.xlabel(r'$f$ ($ Hz)$', fontsize=20)
                        plt.ylabel(r'$S_{w}$ ($m^2 s^{-1})$', fontsize=20)
                        plt.rc('xtick', direction='in', color='k')
                        plt.rc('ytick', direction='in', color='k')
                        plt.legend(loc='upper right',ncol=2,fontsize=16)
                        g1 = plt.grid(b=True, which='major', color='k', linestyle='-', linewidth=0.35)
                        g2 = plt.grid(b=True, which='minor', color='k', linestyle='-', linewidth=0.1)
                        ax1.tick_params(axis='both', labelsize=16)
                        plt.minorticks_on()
                        #plt.show()

                        ax2     = plt.subplot(326)
                        ax2.plot([fre_w_n[indfw_s[0],i], fre_w_n[indfw_s[0],i]],[0, 1000],'k-',
linewidth=2, markeredgecolor='k',markersize=8)
                        ax2.plot([fre_w_n[indfw_e[0],i], fre_w_n[indfw_e[0],i]],[0, 1000],'k-',
linewidth=2, markeredgecolor='k',markersize=8)
                        ax2.loglog(fre_w_n[:,i],Sf_w_n[:,i],                                     'ks',
label='$Measured$',linewidth=1, markeredgecolor='k',markersize=6)
                        ax2.loglog(fre_w_n[:,i],N400_n_1par(fre_w_n[:,i], ANw_n[flag_case,i]),
'r-', label='$Fitted$ $ N400$ $( 1 para)$',linewidth=2, markeredgecolor='k',markersize=8)
                        ax2.loglog(fre_w_n[:,i],N400_n_2par(fre_w_n[:,i],
ANw_n2[flag_case,i],BNw_n2[flag_case,i]), 'g-', label='$Fitted$ $ N400$ $( 2 para)$',linewidth=2,
markeredgecolor='k',markersize=8)
                        ax2.loglog(fre_w_n[:,i],N400_n_1par(fre_w_n[:,i], 6.8),                 'b--
',label='$N400$ $ ref$',linewidth=2, markeredgecolor='k',markersize=8)
                        ax2.fill_between(fre_w_n[:,i],0,1000,where=
fre_w_n[:,i]>f_ver*Lwx_ref[0,i]/meanU[0,i],facecolor='grey', alpha=0.5)
                        ax2.set_ylim(0.6*Sf_w_n[-1,i], 1.3*np.max(Sf_w_n[:,i]))
                        ax2.set_xlim(0.01, 2)
                        plt.xlabel(r'$f*L_{wx}/\overline{U}$', fontsize=20)
                        plt.ylabel(r'$f*S_{w}$ $/\sigma_{w\_measure}^2$', fontsize=20)
                        plt.rc('xtick', direction='in', color='k')
                        plt.rc('ytick', direction='in', color='k')
                        plt.legend(loc='lower left',ncol=2,fontsize=16)
                        g1 = plt.grid(b=True, which='major', color='k', linestyle='-', linewidth=0.35)
                        g2 = plt.grid(b=True, which='minor', color='k', linestyle='-', linewidth=0.1)
                        ax2.tick_params(axis='both', labelsize=16)
```

```
                        plt.minorticks_on()
                        plt.show()
                        plt.close(fig)

                        fig.tight_layout(rect=[0, 0, 1, 0.95])

                        path      = 'D:/DATA/Plots/Osp1 Spectra fitting/'
                        save_tite = 'Storm No.' + str(event+1) + '_spectrum fitting N400 Osp1 sensor
No.' + str(i+1) + '_'\
                                    + str(date_begin.year) + '_' + str(date_begin.month) + '_' +
str(date_begin.day)\
                                        + '_' + str(date_begin.hour) + 'smooth.png'
                        fig.savefig(path + save_tite)


#%% summarize the results for each event
        result={}
        result['date_start'] = time_s1.strftime("%Y-%m-%d %H:%M")
        result['date_end']   = time_e1.strftime("%Y-%m-%d %H:%M")
        result['U']          = U
        result['Dir']        = Dir
        result['stdu']       = stdu
        result['Iu']         = Iu
        result['Iv']         = Iv
        result['Iw']         = Iw
        result['u_f']        = u_f
        result['alpha']      = alpha
        result['beta']       = beta
        result['zeta']       = zeta

        result['ANu']        = ANu
        result['ANu_n']      = ANu_n
        result['ANu2']       = ANu2
        result['ANu_n2']     = ANu_n2
        result['BNu2']       = BNu2
        result['BNu_n2']     = BNu_n2
        result['AHu']        = AHu
        result['BHu']        = BHu
        result['AHu_n']      = AHu_n
        result['BHu_n']      = BHu_n

        result['ANv']        = ANv
        result['ANv_n']      = ANv_n
        result['ANv2']       = ANv2
        result['ANv_n2']     = ANv_n2
        result['BNv2']       = BNv2
        result['BNv_n2']     = BNv_n2

        result['ANw']        = ANw
        result['ANw_n']      = ANw_n
        result['ANw2']       = ANw2
        result['ANw_n2']     = ANw_n2
        result['BNw2']       = BNw2
        result['BNw_n2']     = BNw_n2

        result['AKu']        = AKu
        result['BKu']        = BKu
        result['AKv']        = AKv
        result['BKv']        = BKv
        result['AKw']        = AKw
        result['BKw']        = BKw

        result['AKu_n']      = AKu_n
        result['BKu_n']      = BKu_n
        result['AKv_n']      = AKv_n
        result['BKv_n']      = BKv_n
        result['AKw_n']      = AKw_n
        result['BKw_n']      = BKw_n

        path      = 'D:/DATA/Results/Osp1 Spectra fitting/'

        save_tite = 'Storm No.' + str(event+1) + '_spectrum fitting Osp1 sensor No.' + str(3) + '_'\
                     + str(time_s1.year) + '_' + str(time_s1.month) + '_' + str(time_s1.day)\
                         + '_' + str(time_s1.hour)
```

```
        with open(path + save_tite + '_smooth_json.txt', "w") as outfile:
            json.dump(result, outfile, cls=NumpyEncoder)
```

A5 Coherence fitting

```
# -*- coding: utf-8 -*-
"""
Created on Mon Feb  18 13:56:06 2019
Read the processed storm events
and calculate the coherence parameters

@author: JUNWAN
"""

import numpy as np
import scipy.io as spio
import matplotlib
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
import scipy.signal as signal
from   scipy.optimize import curve_fit
from   scipy.optimize import least_squares
import re
import pandas as pd
from   datetime import datetime
from   datetime import timedelta
from scipy.stats import binned_statistic
import os.path
import matplotlib.dates as mdates
import json
import pickle
import transformations      as TF

from sklearn.neighbors import KernelDensity
from scipy.signal import argrelextrema

# ================================================================================
# boom_direction_mag_from_mast_deg    = '358.0 deg (154003), 178.0 deg (154004), 358.0 deg (154005)'
# instrument_north_direction_mag_deg  = '178.0 deg (154003), 358.0 deg (154004), 178.0 deg (154005)'
# instrument_north_direction_true_deg = '178.57 deg (154003), 358.57 deg (154004), 178.57 deg
(154005)'
# ================================================================================

def running_mean(x, N):
    cumsum = np.cumsum(np.insert(x, 0, 0))
    return (cumsum[N:] - cumsum[:-N]) / float(N)
def datenum(d):
    return 366 + d.toordinal() + (d - datetime.fromordinal(d.toordinal())).total_seconds()/(24*60*60)
class NumpyEncoder(json.JSONEncoder):
    def default(self, obj):
        if isinstance(obj, np.ndarray):
            return obj.tolist()
        return json.JSONEncoder.default(self, obj)

# loop over month
def month_year_iter( start_month, start_year, end_month, end_year ):
    ym_start= 12*start_year + start_month - 1
    ym_end= 12*end_year + end_month
    loop=[]
    for ym in range( ym_start, ym_end ):
        loop = np.append(loop,divmod( ym, 12 ))
        #y[ym], m[ym] = divmod( ym, 12 )
    loop=loop.reshape((int(loop.size/2),2))
    return loop

# function to calculate the wind direciton and tranformation matrix from global XYZ to local wind uvw
# local v axis is conti-clockwise 90 deg with respect to u axis
# gloabl X is pointing east and global Y is pointing north
def windXY2dir(X,Y):
    dir_W  =  np.arctan2(X,Y)
```

```
    if dir_W<0:
        dir_W=dir_W+2*np.pi
    G_x = [1,0,0]
    G_y = [0,1,0]
    G_z = [0,0,1]
    S_u     = [np.sin(dir_W),        np.cos(dir_W),        0]
    S_v     = [np.sin(dir_W-np.pi/2),np.cos(dir_W-np.pi/2),0]
    S_w     = [0,0,1]
    # transformation matrix from XYZ to UVW
    TT      = TF.T_xyzXYZ(G_x, G_y, G_z, S_u, S_v, S_w, dim='3x3')
    return (dir_W,TT)

def wind_dir(X,Y):
    dir_W  =  np.arctan2(X,Y)
    dir_W[dir_W<0] = dir_W[dir_W<0]+2*np.pi
    return (dir_W)

# function to calculate the wind direction, mean wind speed, local wind uvw components and the time
tags
def windXYZ2uvw(data):
    # calculate the non-mean for each component of the wind components in global XYZ within starting
and ending time
    data_temp = np.nanmean(data,0)
    if ~np.isnan(data.any()):
        [dir_w,tt_w] = windXY2dir(data_temp[0],data_temp[1])
        U            = np.matmul(tt_w.T , data_temp)[0]
        uvw          = np.matmul(tt_w.T , data.T).T
        return(dir_w,U,uvw)

# round time to 1 minute
def round_time(dt=None, round_to=60):
   if dt == None:
       dt = datetime.now()
   seconds = (dt - dt.min).seconds
   rounding = (seconds+round_to/2) // round_to * round_to
   return dt + timedelta(0,rounding-seconds,-dt.microsecond)

regex_num  = r"[-+]?[.]?[\d]+(?:\d\d\d)*[\.]?\d*(?:[eE][-+]?\d+)?"

z0=0.01   # roughness lengthscale
delta_ms         = timedelta(milliseconds=100)
delta_2d         = timedelta(days=2)
delta_30min      = timedelta(minutes=30)

#%%
cases = pd.read_excel('Storm events_new.xlsx', sheet_name='Clustered wind storms sort')
delta_1d         = mdates.date2num(datetime(2,1,2,1,0))-mdates.date2num(datetime(2,1,1,1,0))
delta_1h         = mdates.date2num(datetime(2,1,1,2,0))-mdates.date2num(datetime(2,1,1,1,0))
delta_10min      = mdates.date2num(datetime(2,1,1,1,10))-mdates.date2num(datetime(2,1,1,1,0))

#%%
# for event in range(0,cases['Time_storm'].size):
# for event in (53, 68, 162, 166, 216, 273, 280, 281, 283, 286, 290, 291, 292, 293, 294, 299, 303):
for event in list(range(0,53)) + list(range(54,68)) + list(range(69,162)) + list(range(163,166)) +
list(range(167,216)) + list(range(217,273)) + list(range(274,280)) + list(range(282,283)) +
list(range(284,286)) + list(range(287,290)) + list(range(295,299)) + list(range(300,303)) +
list(range(304,305)):
    # event = 16
    time = datetime.strptime(cases['Time_storm'][event], '%Y-%m-%d %H:%M:%S%z').replace(tzinfo=None)

    # starting and ending time of analyzed data window (based on storm mean wind speed time history)
    time_s1 = datetime.strptime(cases['Time_s1'][event], '%Y-%m-%d %H:%M:%S%z').replace(tzinfo=None)
    time_e1 = datetime.strptime(cases['Time_e1'][event], '%Y-%m-%d %H:%M:%S%z').replace(tzinfo=None)

    # number of hourly events
    num_case = int(np.round((mdates.date2num(time_e1)-mdates.date2num(time_s1))/delta_1h))
    # num_case = 24

    # define the size of the fitted spectral parameters
    Cux    = np.zeros((num_case,1))
    Cuy    = np.zeros((num_case,1))
    Cuz    = np.zeros((num_case,1))
    Cvx    = np.zeros((num_case,1))
    Cvy    = np.zeros((num_case,1))
```

```
    Cvz    = np.zeros((num_case,1))
    Cwx    = np.zeros((num_case,1))
    Cwy    = np.zeros((num_case,1))
    Cwz    = np.zeros((num_case,1))

    U      = np.zeros((num_case,3))
    Dir    = np.zeros((num_case,3))
    Iu     = np.zeros((num_case,3))
    Iv     = np.zeros((num_case,3))
    Iw     = np.zeros((num_case,3))
    ddx    = np.zeros((num_case,3))
    ddy    = np.zeros((num_case,3))

    # Osp1 height info
    info_osp1 = pd.read_excel('D:/DATA/files_osp1.xls', sheet_name='Height')

    if os.path.isfile('D:/DATA/osp1/' + 'Osp1_' + str(time.year) + '_' + str(time.month) + '_' +
str(time.day) + '_storm_raw_data.pkl'):
        with open('D:/DATA/osp1/' + 'Osp1_' + str(time.year) + '_' + str(time.month) + '_' +
str(time.day) + '_storm_' + 'Osp1_1h' + '_ins.pkl', 'rb') as f:
            Osp1_1h_ins  = pickle.load( f )
        with open('D:/DATA/osp1/' + 'Osp1_' + str(time.year) + '_' + str(time.month) + '_' +
str(time.day) + '_storm_' + 'Osp1_1h' + '_mean.pkl', 'rb') as f:
            Osp1_1h_mean = pickle.load( f )

#%%
        for flag_case in range(1,num_case):
            # flag_case = 7
            # make sure the starting and ending date is half a hour before and after the time_tag (to
be consistent with mean values)
            date_begin = round_time(mdates.num2date(mdates.date2num(time_s1) - 0.5*delta_1h +
delta_1h*flag_case).replace(tzinfo = None))
            date_stop  = round_time(mdates.num2date(mdates.date2num(date_begin) +
delta_1h).replace(tzinfo = None))
            # integer hour tag
            date = mdates.num2date(mdates.date2num(time_s1) + delta_1h*flag_case).replace(tzinfo =
None).replace(microsecond=0)
            # get the index of the starting and ending hour index in time

            # the original date number is based on matplotlib 3.1.3, so we have to create new time
series here
            # the time_new is created based on the time tag of the mean wind storm data
            date_new_start =  Osp1_1h_mean.iloc[0,0]  - delta_30min
            date_new_end   =  Osp1_1h_mean.iloc[-1,0] + delta_30min - delta_ms
            time_new = pd.date_range(date_new_start,date_new_end,freq = '100L')
            time_new = pd.Series(mdates.date2num(time_new))

            # corresponding time tag of each hourly event
            num_s     = time_new.index[time_new == mdates.date2num(date_begin)].tolist()
            num_e     = time_new.index[time_new == mdates.date2num(date_stop)].tolist()
            Num       = Osp1_1h_mean.index[Osp1_1h_mean['Time'].dt.tz_localize(None) == date].tolist()

            # get the target time series
            tt    = time_new.iloc[num_s[0]:num_e[0]].reset_index(drop = True)

            # check the ratio of NANs in data, if it's more than 10%, we do not work on it. otherwise,
we interpolate
            nan_ratio =
Osp1_1h_ins.loc[num_s[0]:num_e[0],['A_u','B_u','C_u']].isnull().sum().sum()/Osp1_1h_ins.loc[num_s[0]:n
um_e[0],['A_u','B_u','C_u']].size.sum()
            if  nan_ratio < 0.05:

                # get the raw data (interpolate the data for NAN using linear method)

                dd    = np.asarray(Osp1_1h_ins.loc[num_s[0]:num_e[0]-
1,['A_dir','B_dir','C_dir']].interpolate(axis=0,limit_direction='both'))
                UU    = np.asarray(Osp1_1h_ins.loc[num_s[0]:num_e[0]-
1,['A_u','B_u','C_u']].interpolate(axis=0,limit_direction='both'))
                VV    = np.asarray(Osp1_1h_ins.loc[num_s[0]:num_e[0]-
1,['A_v','B_v','C_v']].interpolate(axis=0,limit_direction='both'))
                WW    = np.asarray(Osp1_1h_ins.loc[num_s[0]:num_e[0]-
1,['A_w','B_w','C_w']].interpolate(axis=0,limit_direction='both'))

                # get the mean value (10 MINS AVERAGING)
```

```
                dt          = 0.1
                Nwin        = round(10*60./dt)
                DD          = np.asarray(pd.DataFrame(dd).rolling(Nwin,min_periods=1).mean())
                movingU     = np.asarray(pd.DataFrame(UU).rolling(Nwin,min_periods=1).mean())
                movingV     = np.asarray(pd.DataFrame(VV).rolling(Nwin,min_periods=1).mean())
                movingW     = np.asarray(pd.DataFrame(WW).rolling(Nwin,min_periods=1).mean())

                # get the turbulent components
                # dd[dd<100]  =  dd[dd<100]+360;
                uu          =  UU - movingU
                vv          =  VV - movingV
                ww          =  WW - movingW

                # statistical data
                meanU    = np.mean(UU,axis=0).reshape((1,3))
                meanDir  = np.mean(DD,axis=0).reshape((1,3))
                stdu_mea = np.std(uu,axis=0).reshape((1,3))
                stdv_mea = np.std(vv,axis=0).reshape((1,3))
                stdw_mea = np.std(ww,axis=0).reshape((1,3))
                Iu_mea   = np.divide(stdu_mea,meanU)
                Iv_mea   = np.divide(stdv_mea,meanU)
                Iw_mea   = np.divide(stdw_mea,meanU)

                U[flag_case,:]   = meanU
                Dir[flag_case,:] = meanDir
                Iu[flag_case,:]  = Iu_mea
                Iv[flag_case,:]  = Iv_mea
                Iw[flag_case,:]  = Iw_mea

                # reference data
                z0       = 0.01;
                Lux_ref  = 100*np.power((np.asarray(info_osp1.iloc[0,:])/10),0.3);
                Lvx_ref  = 1/4* Lux_ref;
                Lwx_ref  = 1/12*Lux_ref;
                Lux_ref  = Lux_ref.reshape(1,(np.size(Lux_ref)))
                Lvx_ref  = Lvx_ref.reshape(1,(np.size(Lux_ref)))
                Lwx_ref  = Lwx_ref.reshape(1,(np.size(Lux_ref)))
                Iu_ref   =
(1./np.log(np.asarray(info_osp1.iloc[0,:])/z0)).reshape((1,np.size(Lux_ref)))
                Iv_ref   = 3/4.*Iu_ref;
                Iw_ref   = 1/2.*Iu_ref;
                stdu_ref = np.multiply(meanU,Iu_ref)
                stdv_ref = np.multiply(meanU,Iv_ref)
                stdw_ref = np.multiply(meanU,Iw_ref)

                # range of the moving mean velocity
                delta_U = (np.max(movingU,0)-np.min(movingU,0)).reshape((1,3))
                delta_Dir = (np.max(DD,0) - np.min(DD,0)).reshape((1,3))

                # this is to check the stationarity range of moving mean should be smaller than +-15%
of the static mean
                if delta_U[0,0]<0.3*meanU[0,0] and delta_Dir[0,0]<20 and meanU[0,0] > 6 and meanU[0,1]
> 6 and meanU[0,2] > 6:    # this is to check the stationarity range of moving mean should be smaller
than +-10% of the static mean
                # spectral analysis of the wind time series
                    Nblock   = 6*3; # Nblock = number of overlapping segment. Friday 07/06/2019, I
have used Nblock = 2. here I use 6 blocks to have segments of 10 min
                    N        = np.shape(uu)[0];
                    Nfft     = np.round(N/Nblock);
                    fs       = 1/dt; # sampling frequency

                    f, Su1 = signal.csd(np.transpose(uu[:,0]), np.transpose(uu[:,0]),fs,
window='hanning',  nperseg=Nfft, noverlap=None, nfft=None, detrend='constant', return_onesided=True)
                    f, Su2 = signal.csd(np.transpose(uu[:,1]), np.transpose(uu[:,1]),fs,
window='hanning',  nperseg=Nfft, noverlap=None, nfft=None, detrend='constant', return_onesided=True)
                    f, Su3 = signal.csd(np.transpose(uu[:,2]), np.transpose(uu[:,2]),fs,
window='hanning',  nperseg=Nfft, noverlap=None, nfft=None, detrend='constant', return_onesided=True)
                    f, Su12= signal.csd(np.transpose(uu[:,0]), np.transpose(uu[:,1]),fs,
window='hanning',  nperseg=Nfft, noverlap=None, nfft=None, detrend='constant', return_onesided=True)
                    f, Su13= signal.csd(np.transpose(uu[:,0]), np.transpose(uu[:,2]),fs,
window='hanning',  nperseg=Nfft, noverlap=None, nfft=None, detrend='constant', return_onesided=True)

                    f, Sv1 = signal.csd(np.transpose(vv[:,0]), np.transpose(vv[:,0]),fs,
window='hanning',  nperseg=Nfft, noverlap=None, nfft=None, detrend='constant', return_onesided=True)
```

```
                    f, Sv2 = signal.csd(np.transpose(vv[:,1]), np.transpose(vv[:,1]),fs,
window='hanning',  nperseg=Nfft, noverlap=None, nfft=None, detrend='constant', return_onesided=True)
                    f, Sv3 = signal.csd(np.transpose(vv[:,2]), np.transpose(vv[:,2]),fs,
window='hanning',  nperseg=Nfft, noverlap=None, nfft=None, detrend='constant', return_onesided=True)
                    f, Sv12= signal.csd(np.transpose(vv[:,0]), np.transpose(vv[:,1]),fs,
window='hanning',  nperseg=Nfft, noverlap=None, nfft=None, detrend='constant', return_onesided=True)
                    f, Sv13= signal.csd(np.transpose(vv[:,0]), np.transpose(vv[:,2]),fs,
window='hanning',  nperseg=Nfft, noverlap=None, nfft=None, detrend='constant', return_onesided=True)

                    f, Sw1 = signal.csd(np.transpose(ww[:,0]), np.transpose(ww[:,0]),fs,
window='hanning',  nperseg=Nfft, noverlap=None, nfft=None, detrend='constant', return_onesided=True)
                    f, Sw2 = signal.csd(np.transpose(ww[:,1]), np.transpose(ww[:,1]),fs,
window='hanning',  nperseg=Nfft, noverlap=None, nfft=None, detrend='constant', return_onesided=True)
                    f, Sw3 = signal.csd(np.transpose(ww[:,2]), np.transpose(ww[:,2]),fs,
window='hanning',  nperseg=Nfft, noverlap=None, nfft=None, detrend='constant', return_onesided=True)
                    f, Sw12= signal.csd(np.transpose(ww[:,0]), np.transpose(ww[:,1]),fs,
window='hanning',  nperseg=Nfft, noverlap=None, nfft=None, detrend='constant', return_onesided=True)
                    f, Sw13= signal.csd(np.transpose(ww[:,0]), np.transpose(ww[:,2]),fs,
window='hanning',  nperseg=Nfft, noverlap=None, nfft=None, detrend='constant', return_onesided=True)

                    f, Suw = signal.csd(np.transpose(uu[:,0]), np.transpose(ww[:,0]),fs,
window='hanning',  nperseg=Nfft, noverlap=None, nfft=None, detrend='constant', return_onesided=True)

# ========================================================================
#                     plt.close("all")
#                     fig     = plt.figure(figsize=(20, 12))
#                     ax1     = plt.subplot(111)
#                     ax1.semilogx(f, Suw, 'b-', label='$Suw$',linewidth=1,
markeredgecolor='k',markersize=6,alpha=0.5)
# ========================================================================

                    Coh_u  = np.real(np.divide(Su12,np.power(np.multiply(Su1,Su2),0.5)))
                    Coh_v  = np.real(np.divide(Sv12,np.power(np.multiply(Sv1,Sv2),0.5)))
                    Coh_w  = np.real(np.divide(Sw12,np.power(np.multiply(Sw1,Sw2),0.5)))

                    Coh_uz  = np.real(np.divide(Su13,np.power(np.multiply(Su1,Su3),0.5)))
                    Coh_vz  = np.real(np.divide(Sv13,np.power(np.multiply(Sv1,Sv3),0.5)))
                    Coh_wz  = np.real(np.divide(Sw13,np.power(np.multiply(Sw1,Sw3),0.5)))

                    Coh_uw  = np.real(np.divide(Suw,np.power(np.multiply(Su1,Sw1),0.5)))

                    dist  = 8   # distance between two top sensors are 8m
                    yaw_s = -2   # for ospøya, the boom is -2 degress from true north or true south
                    yaw_w = np.mean(meanDir[0,0:2])-yaw_s
                    if yaw_w>180:
                        yaw_w=np.abs(yaw_w-270)
                    else:
                        yaw_w=np.abs(yaw_w-90)
                    # dx the along-wind speration distance; dy is the lateral seperation distance ; dz
is the vertical seperation distance
                    # dx ad dy are for based on sensor 1 and 2, and dz is based on sensor 1 and 3
                    global dx,dy,dz,U_coh
                    dx = dist*np.sin(np.radians(yaw_w))
                    dy = dist*np.cos(np.radians(yaw_w))
                    dz = np.asarray(info_osp1.iloc[0,:])[0]-np.asarray(info_osp1.iloc[0,:])[2]
                    U_coh = np.mean(meanU[0,0:2])
                    ddx[flag_case,0]  = dx
                    ddy[flag_case,0]  = dy

    # ========================================================================
    #                     def Coh_lateral(f,Cx,Cy):
    #                         return np.exp(-
1*np.divide(f,U_coh)*np.power(np.power(Cx*dx,2)+np.power(Cy*dy,2),0.5))
    #
    # ========================================================================
                    # based on Etienne's comments
                    def Coh_lateral(f,Cx,Cy):
                        return np.exp(-
1*np.divide(f,U_coh)*np.power(np.power(Cx*dx,2)+np.power(Cy*dy,2),0.5))*np.cos(dx*2*np.pi*f/U_coh)

                    def Coh_lateralu(f,Cy):
                        return np.exp(-
1*np.divide(f,U_coh)*np.power(np.power(1/3*Cy*dx,2)+np.power(Cy*dy,2),0.5))*np.cos(dx*2*np.pi*f/U_coh)
```

```
                    def Coh_lateralv(f,Cy):
                        return np.exp(-
1*np.divide(f,U_coh)*np.power(np.power(1/4*Cy*dx,2)+np.power(Cy*dy,2),0.5))*np.cos(dx*2*np.pi*f/U_coh)

                    def Coh_lateralw(f,Cy):
                        return np.exp(-
1*np.divide(f,U_coh)*np.power(np.power(1/6*Cy*dx,2)+np.power(Cy*dy,2),0.5))*np.cos(dx*2*np.pi*f/U_coh)

                    def Coh_vertical(f,Cz):
                        return np.exp(-1*np.divide(f,U_coh)*dz*Cz)

                    # bin average the spectra in logspace
                    num_log = 60
                    f_s_start = np.log10(f[1])
                    f_s_end   = np.log10(f[-1])
                    f_s  = np.logspace(f_s_start,f_s_end,num_log+1).reshape(num_log+1,1)
                    Coh_u_s = np.zeros((np.size(f_s)-1,1))
                    Coh_v_s = np.zeros((np.size(f_s)-1,1))
                    Coh_w_s = np.zeros((np.size(f_s)-1,1))
                    Coh_uz_s = np.zeros((np.size(f_s)-1,1))
                    Coh_vz_s = np.zeros((np.size(f_s)-1,1))
                    Coh_wz_s = np.zeros((np.size(f_s)-1,1))

                    Coh_u_s, edges, _ = binned_statistic(f,Coh_u, statistic='mean', bins=f_s[:,0])
                    Coh_v_s, edges, _ = binned_statistic(f,Coh_v, statistic='mean', bins=f_s[:,0])
                    Coh_w_s, edges, _ = binned_statistic(f,Coh_w, statistic='mean', bins=f_s[:,0])
                    Coh_uz_s, edges, _ = binned_statistic(f,Coh_uz, statistic='mean', bins=f_s[:,0])
                    Coh_vz_s, edges, _ = binned_statistic(f,Coh_vz, statistic='mean', bins=f_s[:,0])
                    Coh_wz_s, edges, _ = binned_statistic(f,Coh_wz, statistic='mean', bins=f_s[:,0])
                    f_s  = edges[:-1]+np.diff(edges)/2
                    nan_index = np.isnan(Coh_u_s)
                    f_s       = f_s[~nan_index]
                    Coh_u_s      = Coh_u_s[~nan_index].reshape((np.size(f_s),1))
                    Coh_v_s      = Coh_v_s[~nan_index].reshape((np.size(f_s),1))
                    Coh_w_s      = Coh_w_s[~nan_index].reshape((np.size(f_s),1))
                    Coh_uz_s      = Coh_uz_s[~nan_index].reshape((np.size(f_s),1))
                    Coh_vz_s      = Coh_vz_s[~nan_index].reshape((np.size(f_s),1))
                    Coh_wz_s      = Coh_wz_s[~nan_index].reshape((np.size(f_s),1))
                    f_s          = f_s.reshape((np.size(f_s),1))

                    # fit the coherence 0.01Hz<f<0.5Hz
                    fu_s = 0.01
                    fu_e = 1

                    indfu_e  =  np.where(abs(f_s-fu_e)==min(abs(f_s-fu_e)))[0]
                    indfu_s  =  np.where(abs(f_s-fu_s)==min(abs(f_s-fu_s)))[0]

                    Cuy[flag_case,0],  pcov = curve_fit(Coh_lateralu,f_s[indfu_s[0]:indfu_e[0],0],
Coh_u_s[indfu_s[0]:indfu_e[0],0],p0=(10),bounds=(0, [ 20]),maxfev=100000)
                    Cvy[flag_case,0],  pcov = curve_fit(Coh_lateralv,f_s[indfu_s[0]:indfu_e[0],0],
Coh_v_s[indfu_s[0]:indfu_e[0],0],p0=(6.5),bounds=(0, [ 20]),maxfev=100000)
                    Cwy[flag_case,0],  pcov = curve_fit(Coh_lateralw,f_s[indfu_s[0]:indfu_e[0],0],
Coh_w_s[indfu_s[0]:indfu_e[0],0],p0=(6.5),bounds=(0, [ 20]),maxfev=100000)
                    Cux[flag_case,0]        = 1/3*Cuy[flag_case,0]
                    Cvx[flag_case,0]        = 1/4*Cvy[flag_case,0]
                    Cwx[flag_case,0]        = 1/6*Cwy[flag_case,0]

                    if Cux[flag_case,0]<0:
                        Cux[flag_case,0]=np.abs(Cux[flag_case,0])
                    if Cuy[flag_case,0]<0:
                        Cuy[flag_case,0]=np.abs(Cuy[flag_case,0])
                    if Cvx[flag_case,0]<0:
                        Cvx[flag_case,0]=np.abs(Cvx[flag_case,0])
                    if Cvy[flag_case,0]<0:
                        Cvy[flag_case,0]=np.abs(Cvy[flag_case,0])
                    if Cwx[flag_case,0]<0:
                        Cwx[flag_case,0]=np.abs(Cwx[flag_case,0])
                    if Cwy[flag_case,0]<0:
                        Cwy[flag_case,0]=np.abs(Cwy[flag_case,0])

                    Cuz[flag_case,0],  pcov = curve_fit(Coh_vertical,f_s[indfu_s[0]:indfu_e[0],0],
Coh_uz_s[indfu_s[0]:indfu_e[0],0],p0=(6),maxfev=100000)
                    Cvz[flag_case,0],  pcov = curve_fit(Coh_vertical,f_s[indfu_s[0]:indfu_e[0],0],
Coh_vz_s[indfu_s[0]:indfu_e[0],0],p0=(6),maxfev=100000)
```

```
                    Cwz[flag_case,0],  pcov = curve_fit(Coh_vertical,f_s[indfu_s[0]:indfu_e[0],0],
Coh_wz_s[indfu_s[0]:indfu_e[0],0],p0=(6),maxfev=100000)

#%%
                    f_hor      = 1/56.06*0.9
                    f_ver      = 1/6.89*0.9
                    matplotlib.use('Agg')

                    plt.close("all")
                    fig     = plt.figure(figsize=(20, 12))
                    ax1     = plt.subplot(321)
                    ax1.plot([fu_s, fu_s],[-10, 10],'k-', linewidth=2,
markeredgecolor='k',markersize=8)
                    ax1.plot([fu_e, fu_e],[-10, 10],'k-', linewidth=2,
markeredgecolor='k',markersize=8)
                    ax1.semilogx(f_s,Coh_u_s,  'ks', label='$Measured$',linewidth=1,
markeredgecolor='k',markersize=6)
                    ax1.semilogx(f,Coh_lateral(f,Cux[flag_case,0],Cuy[flag_case,0]),  'r-',
label='$Fitted$ (Cux='+
str(np.round(Cux[flag_case,0],2))+';Cuy='+str(np.round(Cuy[flag_case,0],2))+')' \
                                ,linewidth=1, markeredgecolor='k',markersize=6)
                    ax1.semilogx(f,Coh_lateral(f,3,10),  'b--', label='$Ref$ (Cux='+
str(3)+';Cuy='+str(10)+')' \
                                ,linewidth=1, markeredgecolor='k',markersize=6)
                    ax1.fill_between(f,-10,10,where= f>f_hor,facecolor='grey', alpha=0.5)
                    ax1.set_xlim(0.005, 2)
                    ax1.set_ylim(np.min(Coh_u_s)*1.1, 1.1)
                    plt.xlabel(r'$f$ ($ Hz)$', fontsize=20)
                    plt.ylabel(r'$Coh_u$ $horizontal$', fontsize=20)
                    fig.suptitle('Event No.' + str(event+1)+ ' Dir=' + str(np.round(meanDir[0,0],1)) +
'$^o$'+'  Osp1 at height ' + str(np.round(np.asarray(info_osp1.iloc[0,:])[0],1)) + 'm ' \
                                + 'from '  + date_begin.strftime("%Y-%m-%d %H:%M") + ' to ' +
date_stop.strftime("%Y-%m-%d %H:%M") +'\n'+ 'dx=' +str(np.round(dx,2))+'m; '+'dy=' \
                                    +str(np.round(dy,2))+'m; ' + 'dz=' +str(np.round(dz,2))+'m',
fontsize=25)
                    plt.rc('xtick', direction='in', color='k')
                    plt.rc('ytick', direction='in', color='k')
                    plt.legend(loc='lower left',ncol=1,fontsize=14)
                    g1 = plt.grid(b=True, which='major', color='k', linestyle='-', linewidth=0.35)
                    g2 = plt.grid(b=True, which='minor', color='k', linestyle='-', linewidth=0.1)
                    ax1.tick_params(axis='both', labelsize=16)
                    plt.minorticks_on()
                    #plt.show()

                    ax1     = plt.subplot(323)
                    ax1.plot([fu_s, fu_s],[-10, 10],'k-', linewidth=2,
markeredgecolor='k',markersize=8)
                    ax1.plot([fu_e, fu_e],[-10, 10],'k-', linewidth=2,
markeredgecolor='k',markersize=8)
                    ax1.semilogx(f_s,Coh_v_s,  'ks', label='$Measured$',linewidth=1,
markeredgecolor='k',markersize=6)
                    ax1.semilogx(f,Coh_lateral(f,Cvx[flag_case,0],Cvy[flag_case,0]),  'r-',
label='$Fitted$ (Cvx='+
str(np.round(Cvx[flag_case,0],2))+';Cvy='+str(np.round(Cvy[flag_case,0],2))+')' \
                                ,linewidth=1, markeredgecolor='k',markersize=6)
                    ax1.semilogx(f,Coh_lateral(f,6,6.5),  'b--', label='$Ref$ (Cvx='+
str(6)+';Cvy='+str(6.5)+')' \
                                ,linewidth=1, markeredgecolor='k',markersize=6)
                    ax1.fill_between(f,-10,10,where= f>f_hor,facecolor='grey', alpha=0.5)
                    ax1.set_xlim(0.005, 2)
                    ax1.set_ylim(np.min(Coh_v_s)*1.1, 1.1)
                    plt.xlabel(r'$f$ ($ Hz)$', fontsize=20)
                    plt.ylabel(r'$Coh_v$ $horizontal$', fontsize=20)
                    plt.rc('xtick', direction='in', color='k')
                    plt.rc('ytick', direction='in', color='k')
                    plt.legend(loc='lower left',ncol=1,fontsize=14)
                    g1 = plt.grid(b=True, which='major', color='k', linestyle='-', linewidth=0.35)
                    g2 = plt.grid(b=True, which='minor', color='k', linestyle='-', linewidth=0.1)
                    ax1.tick_params(axis='both', labelsize=16)
                    plt.minorticks_on()
                    #plt.show()


                    ax1     = plt.subplot(325)
```

```
                    ax1.plot([fu_s, fu_s],[-10, 10],'k-', linewidth=2,
markeredgecolor='k',markersize=8)
                    ax1.plot([fu_e, fu_e],[-10, 10],'k-', linewidth=2,
markeredgecolor='k',markersize=8)
                    ax1.semilogx(f_s,Coh_w_s,  'ks', label='$Measured$',linewidth=1,
markeredgecolor='k',markersize=6)
                    ax1.semilogx(f,Coh_lateral(f,Cwx[flag_case,0],Cwy[flag_case,0]),  'r-',
label='$Fitted$ (Cwx='+
str(np.round(Cwx[flag_case,0],2))+';Cwy='+str(np.round(Cwy[flag_case,0],2))+')' \
                                ,linewidth=1, markeredgecolor='k',markersize=6)
                    ax1.semilogx(f,Coh_lateral(f,3,6.5),  'b--', label='$Ref$ (Cwx='+
str(3)+';Cwy='+str(6.5)+')' \
                                ,linewidth=1, markeredgecolor='k',markersize=6)
                    ax1.fill_between(f,-10,10,where= f>f_hor,facecolor='grey', alpha=0.5)
                    ax1.set_xlim(0.005, 2)
                    ax1.set_ylim(np.min(Coh_w_s)*1.1, 1.1)
                    plt.xlabel(r'$f$ ($ Hz)$', fontsize=20)
                    plt.ylabel(r'$Coh_w$ $horizontal$', fontsize=20)
                    plt.rc('xtick', direction='in', color='k')
                    plt.rc('ytick', direction='in', color='k')
                    plt.legend(loc='lower left',ncol=1,fontsize=14)
                    g1 = plt.grid(b=True, which='major', color='k', linestyle='-', linewidth=0.35)
                    g2 = plt.grid(b=True, which='minor', color='k', linestyle='-', linewidth=0.1)
                    ax1.tick_params(axis='both', labelsize=16)
                    plt.minorticks_on()
                    #plt.show()


                    ax1     = plt.subplot(322)
                    ax1.plot([fu_s, fu_s],[-10, 10],'k-', linewidth=2,
markeredgecolor='k',markersize=8)
                    ax1.plot([fu_e, fu_e],[-10, 10],'k-', linewidth=2,
markeredgecolor='k',markersize=8)
                    ax1.semilogx(f_s,Coh_uz_s,  'ks', label='$Measured$',linewidth=1,
markeredgecolor='k',markersize=6)
                    ax1.semilogx(f,Coh_vertical(f,Cuz[flag_case,0]),  'r-', label='$Fitted$ (Cuz='+
str(np.round(Cuz[flag_case,0],2)) +')' \
                                ,linewidth=1, markeredgecolor='k',markersize=6)
                    ax1.semilogx(f,Coh_vertical(f,10),  'b--', label='$Ref$ (Cuz='+ str(10) +')' \
                                ,linewidth=1, markeredgecolor='k',markersize=6)
                    ax1.fill_between(f,-10,10,where= f>f_ver,facecolor='grey', alpha=0.5)
                    ax1.set_xlim(0.005, 2)
                    ax1.set_ylim(np.min(Coh_uz_s)*1.1, 1.1)
                    plt.xlabel(r'$f$ ($ Hz)$', fontsize=20)
                    plt.ylabel(r'$Coh_u$ $vertical$', fontsize=20)
                    plt.rc('xtick', direction='in', color='k')
                    plt.rc('ytick', direction='in', color='k')
                    plt.legend(loc='lower left',ncol=1,fontsize=14)
                    g1 = plt.grid(b=True, which='major', color='k', linestyle='-', linewidth=0.35)
                    g2 = plt.grid(b=True, which='minor', color='k', linestyle='-', linewidth=0.1)
                    ax1.tick_params(axis='both', labelsize=16)
                    plt.minorticks_on()
                    #plt.show()

                    ax1     = plt.subplot(324)
                    ax1.plot([fu_s, fu_s],[-10, 10],'k-', linewidth=2,
markeredgecolor='k',markersize=8)
                    ax1.plot([fu_e, fu_e],[-10, 10],'k-', linewidth=2,
markeredgecolor='k',markersize=8)
                    ax1.semilogx(f_s,Coh_vz_s,  'ks', label='$Measured$',linewidth=1,
markeredgecolor='k',markersize=6)
                    ax1.semilogx(f,Coh_vertical(f,Cvz[flag_case,0]),  'r-', label='$Fitted$ (Cvz='+
str(np.round(Cvz[flag_case,0],2)) +')' \
                                ,linewidth=1, markeredgecolor='k',markersize=6)
                    ax1.semilogx(f,Coh_vertical(f,6.5),  'b--', label='$Ref$ (Cvz='+ str(6.5) +')' \
                                ,linewidth=1, markeredgecolor='k',markersize=6)
                    ax1.fill_between(f,-10,10,where= f>f_ver,facecolor='grey', alpha=0.5)
                    ax1.set_xlim(0.005, 2)
                    ax1.set_ylim(np.min(Coh_vz_s)*1.1, 1.1)
                    plt.xlabel(r'$f$ ($ Hz)$', fontsize=20)
                    plt.ylabel(r'$Coh_v$ $vertical$', fontsize=20)
                    plt.rc('xtick', direction='in', color='k')
                    plt.rc('ytick', direction='in', color='k')
                    plt.legend(loc='lower left',ncol=1,fontsize=14)
```

```
                    g1 = plt.grid(b=True, which='major', color='k', linestyle='-', linewidth=0.35)
                    g2 = plt.grid(b=True, which='minor', color='k', linestyle='-', linewidth=0.1)
                    ax1.tick_params(axis='both', labelsize=16)
                    plt.minorticks_on()
                    #plt.show()

                    ax1     = plt.subplot(326)
                    ax1.plot([fu_s, fu_s],[-10, 10],'k-', linewidth=2,
markeredgecolor='k',markersize=8)
                    ax1.plot([fu_e, fu_e],[-10, 10],'k-', linewidth=2,
markeredgecolor='k',markersize=8)
                    ax1.semilogx(f_s,Coh_wz_s,  'ks', label='$Measured$',linewidth=1,
markeredgecolor='k',markersize=6)
                    ax1.semilogx(f,Coh_vertical(f,Cwz[flag_case,0]),  'r-', label='$Fitted$ (Cwz='+
str(np.round(Cwz[flag_case,0],2)) +')' \
                                ,linewidth=1, markeredgecolor='k',markersize=6)
                    ax1.semilogx(f,Coh_vertical(f,3),  'b--', label='$Ref$ (Cwz='+ str(3) +')' \
                                ,linewidth=1, markeredgecolor='k',markersize=6)
                    ax1.fill_between(f,-10,10,where= f>f_ver,facecolor='grey', alpha=0.5)
                    ax1.set_xlim(0.005, 2)
                    ax1.set_ylim(np.min(Coh_wz_s)*1.1, 1.1)
                    plt.xlabel(r'$f$ ($ Hz)$', fontsize=20)
                    plt.ylabel(r'$Coh_w$ $vertical$', fontsize=20)
                    plt.rc('xtick', direction='in', color='k')
                    plt.rc('ytick', direction='in', color='k')
                    plt.legend(loc='lower left',ncol=1,fontsize=14)
                    g1 = plt.grid(b=True, which='major', color='k', linestyle='-', linewidth=0.35)
                    g2 = plt.grid(b=True, which='minor', color='k', linestyle='-', linewidth=0.1)
                    ax1.tick_params(axis='both', labelsize=16)
                    plt.minorticks_on()
                    #plt.show()
                    plt.close(fig)

                    fig.tight_layout(rect=[0, 0, 1, 0.92])

                    path      = 'D:/DATA/Plots/Osp1 Coherence fitting xy ratio//'
                    save_tite = 'Storm No.' + str(event+1) + '_coherence fitting Osp1_'\
                        + str(date_begin.year) + '_' + str(date_begin.month) + '_' +
str(date_begin.day)\
                            + '_' + str(date_begin.hour) + 'smooth.png'
                    fig.savefig(path + save_tite)

#%%
    result={}
    result['date_start'] = time_s1.strftime("%Y-%m-%d %H:%M")
    result['date_end']   = time_e1.strftime("%Y-%m-%d %H:%M")
    result['U']          = U
    result['Dir']        = Dir
    result['Iu']         = Iu
    result['Iv']         = Iv
    result['Iw']         = Iw
    result['dx']         = ddx
    result['dy']         = ddy
    result['Cux']       = Cux
    result['Cuy']       = Cuy
    result['Cuz']       = Cuz
    result['Cvx']       = Cvx
    result['Cvy']       = Cvy
    result['Cvz']       = Cvz
    result['Cwx']       = Cwx
    result['Cwy']       = Cwy
    result['Cwz']       = Cwz

    save_tite = 'Storm No.' + str(event+1) + '_coherence fitting Osp1_'\
        + str(date_begin.year) + '_' + str(date_begin.month) + '_' + str(date_begin.day)\
            + '_' + str(date_begin.hour)
    path      = 'D:/DATA/Results/Osp1 Coherence fitting xy ratio/'
    with open(path + save_tite + '_smooth_json.txt', "w") as outfile:
        json.dump(result, outfile, cls=NumpyEncoder)
```

B: Distribution of fitted parameters with stability

B1 Spectral parameters

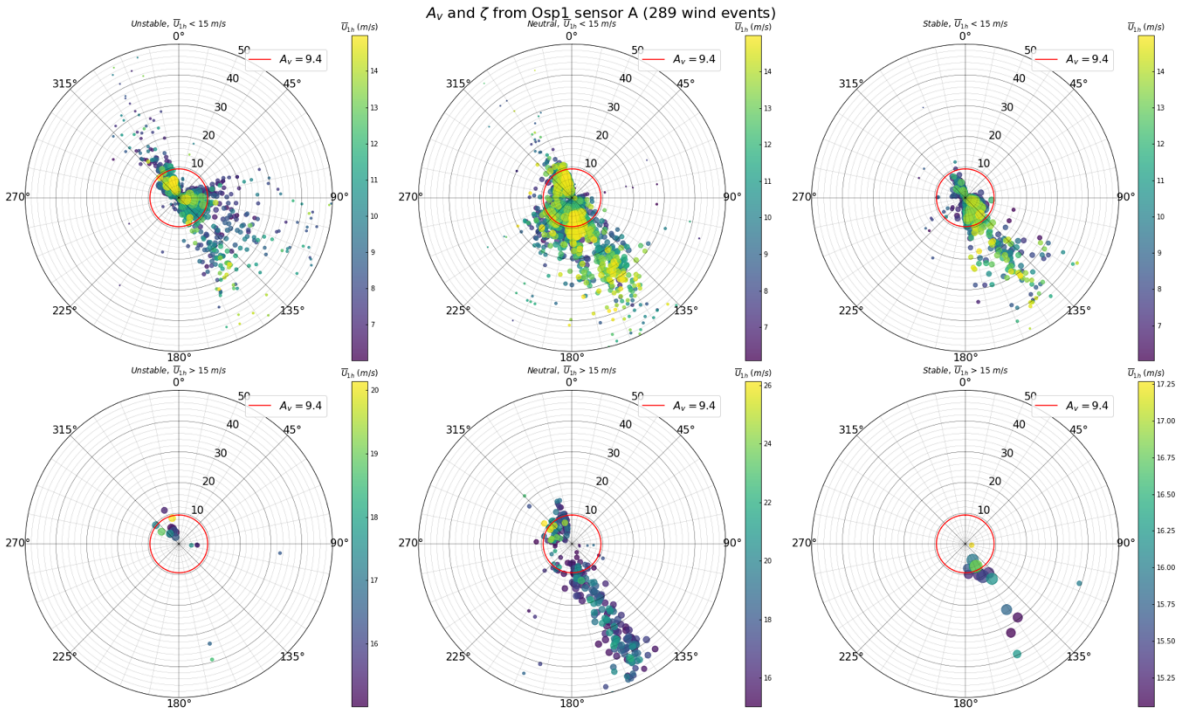

Figure B1.1: Distribution of the fitted A_v with stability at Osp1.

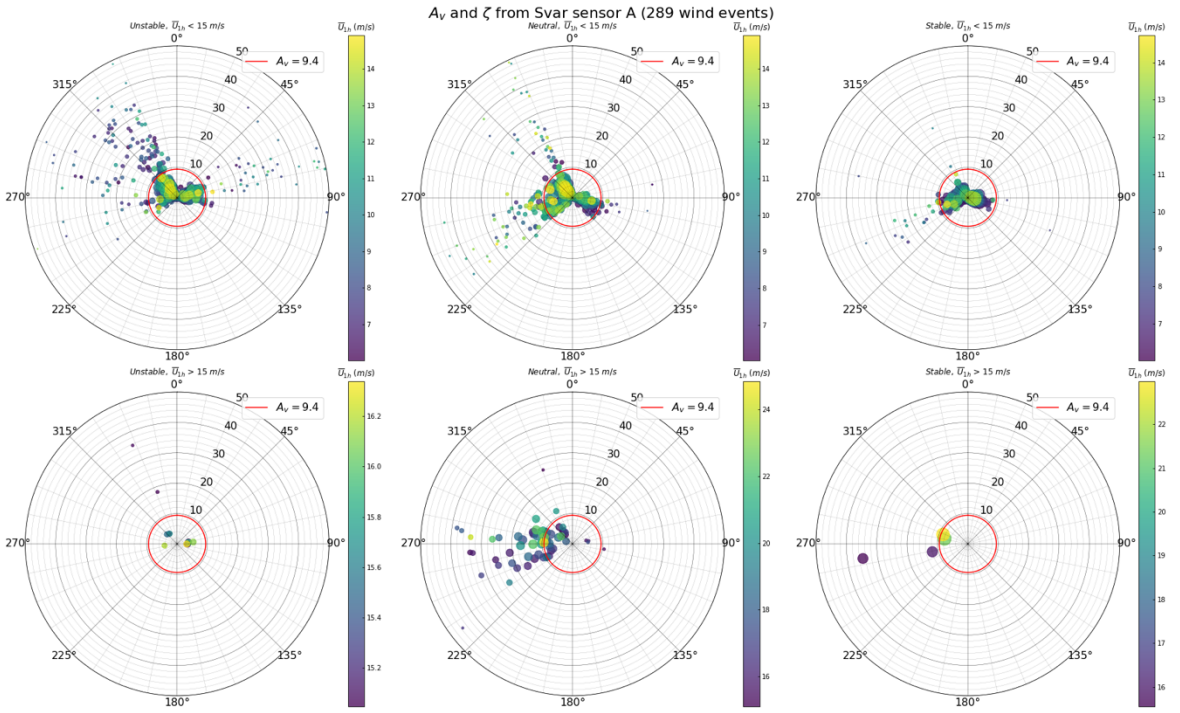

Figure B1.2: Distribution of the fitted A_v with stability at Svar.

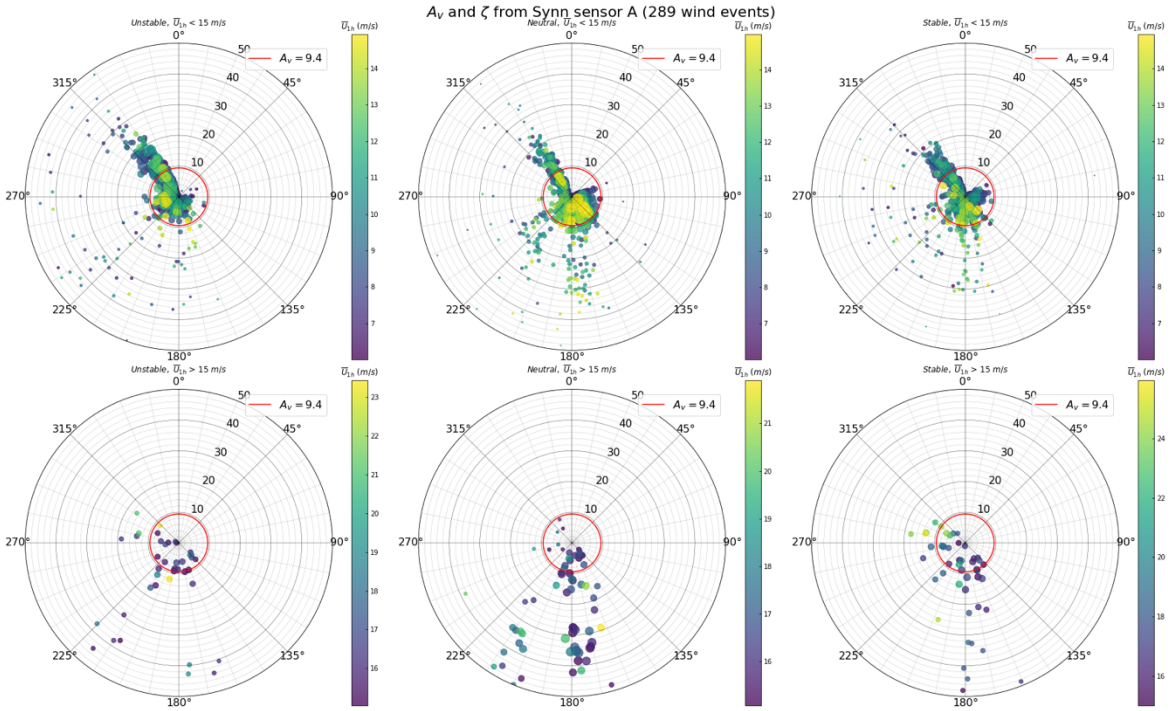

Figure B1.3: Distribution of the fitted A_v with stability at Synn.

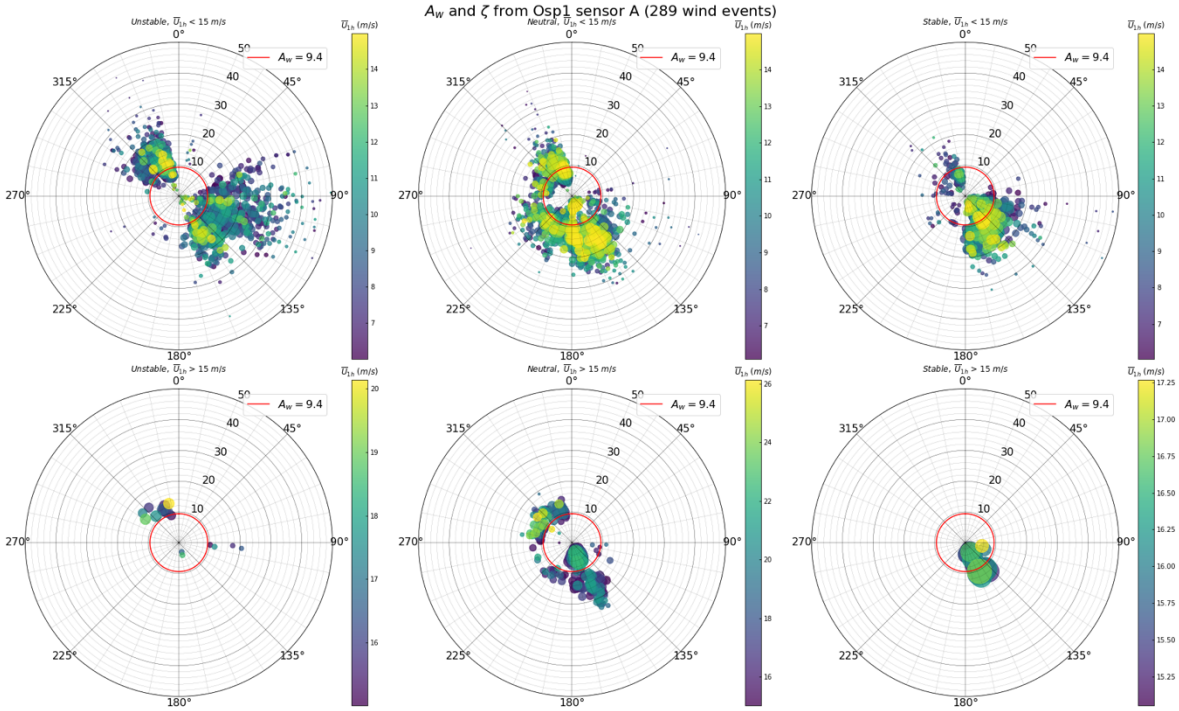

Figure B1.4: Distribution of the fitted A_w with stability at Osp1.

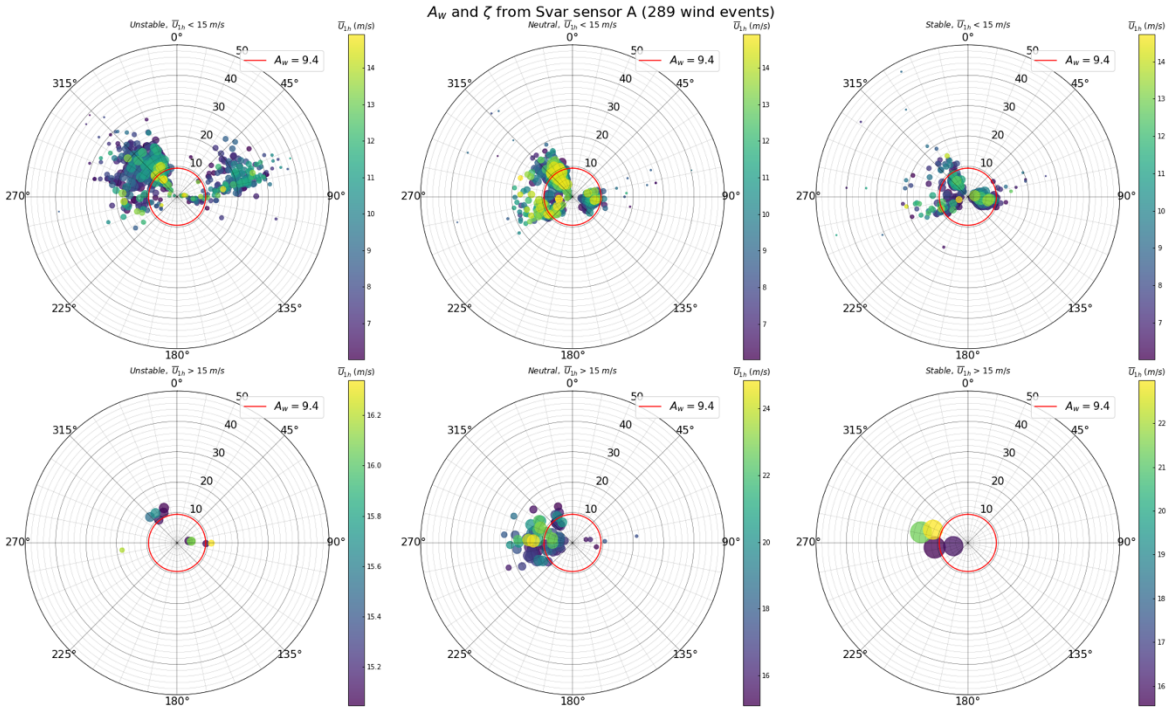

Figure B1.5: Distribution of the fitted A_w with stability at Svar.

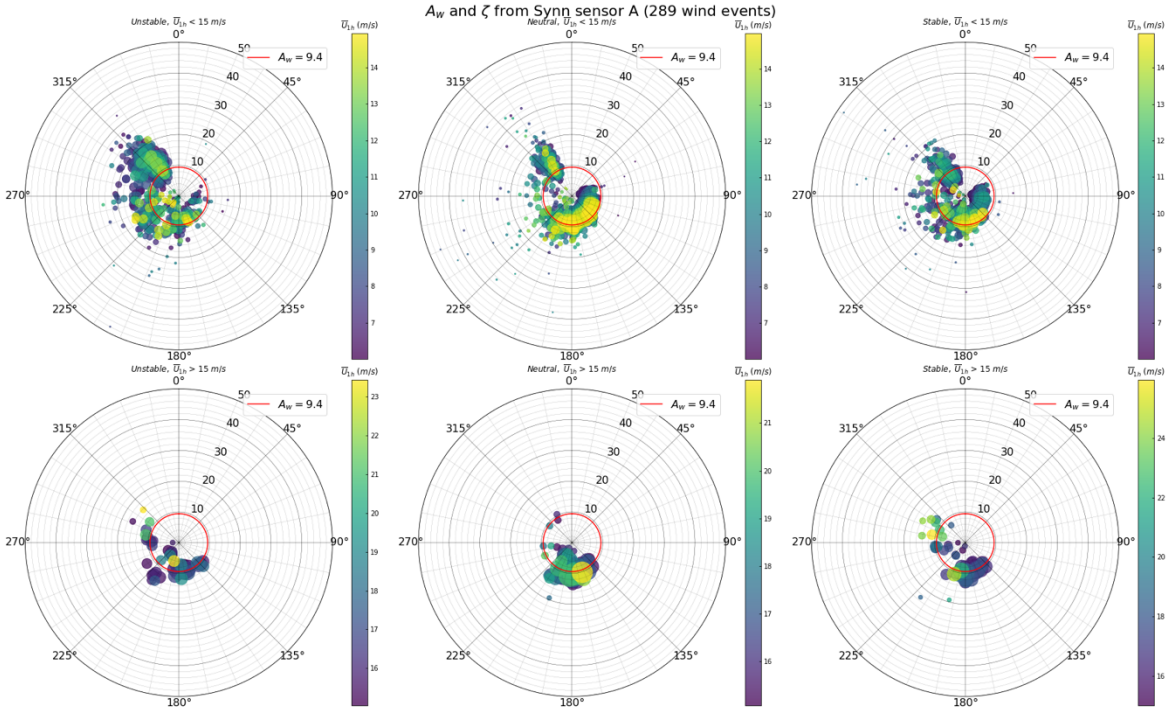

Figure B1.6: Distribution of the fitted A_w with stability at Synn.

B2 Coherence parameters

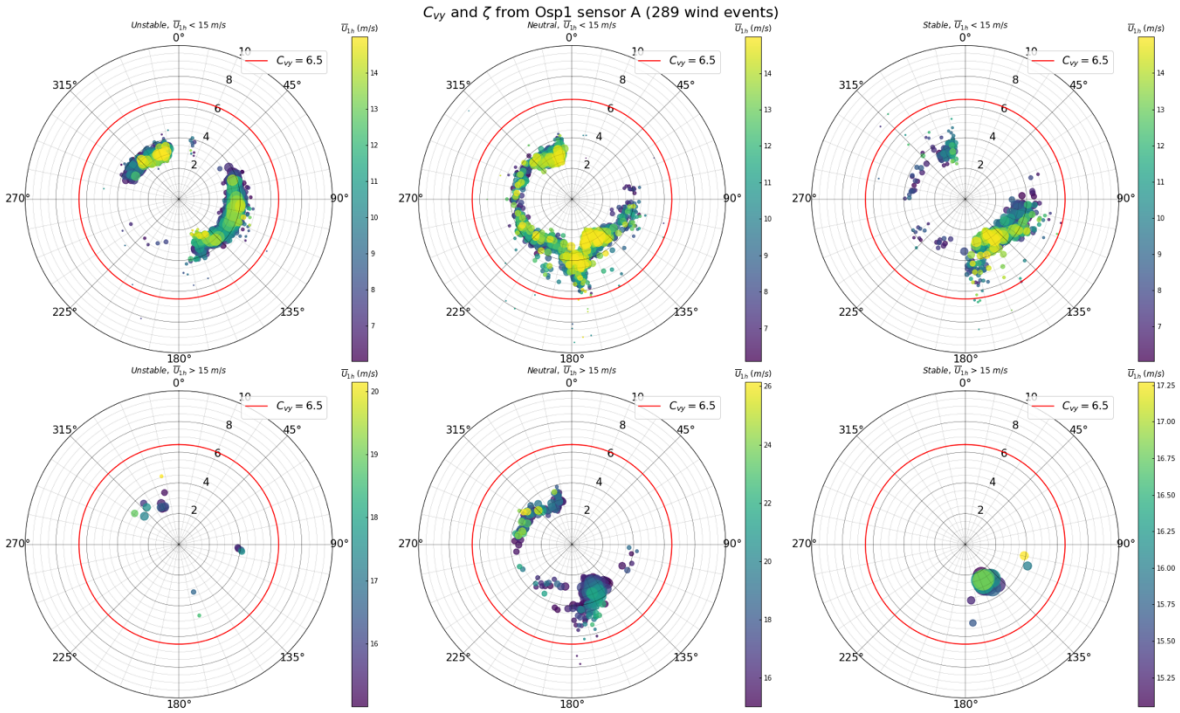

Figure B2.1: Distribution of the fitted C_{vy} with stability at Osp1.

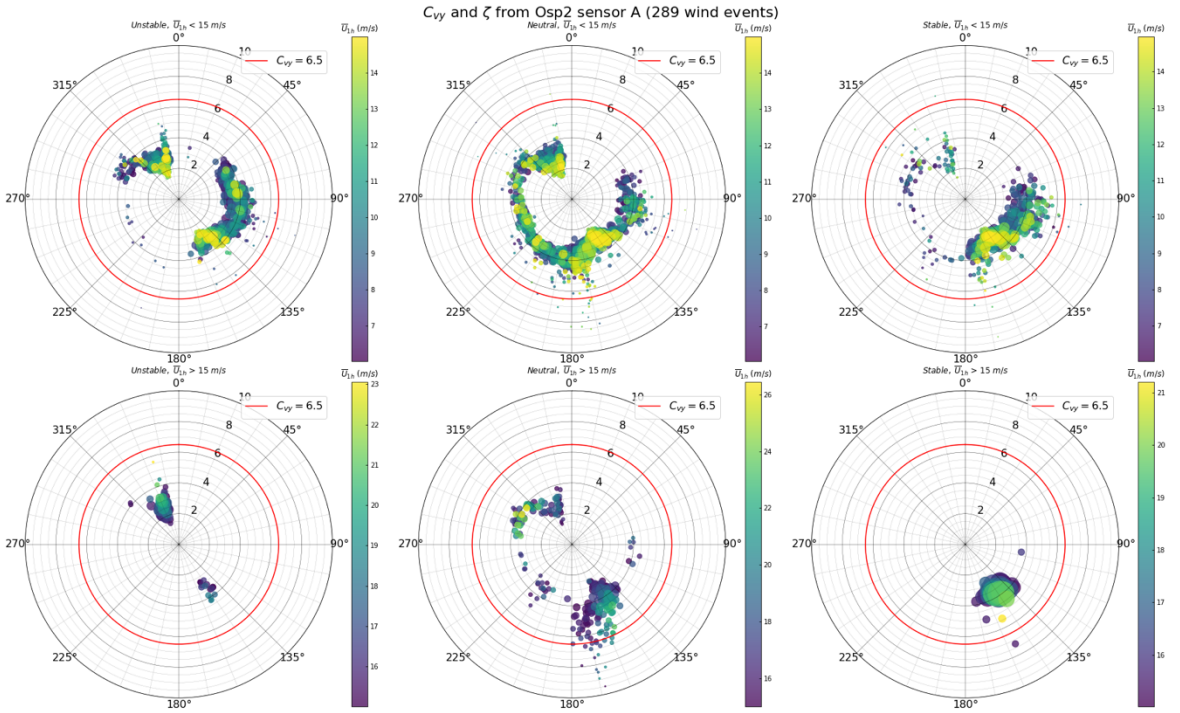

Figure B2.2: Distribution of the fitted C_{vy} with stability at Osp2.

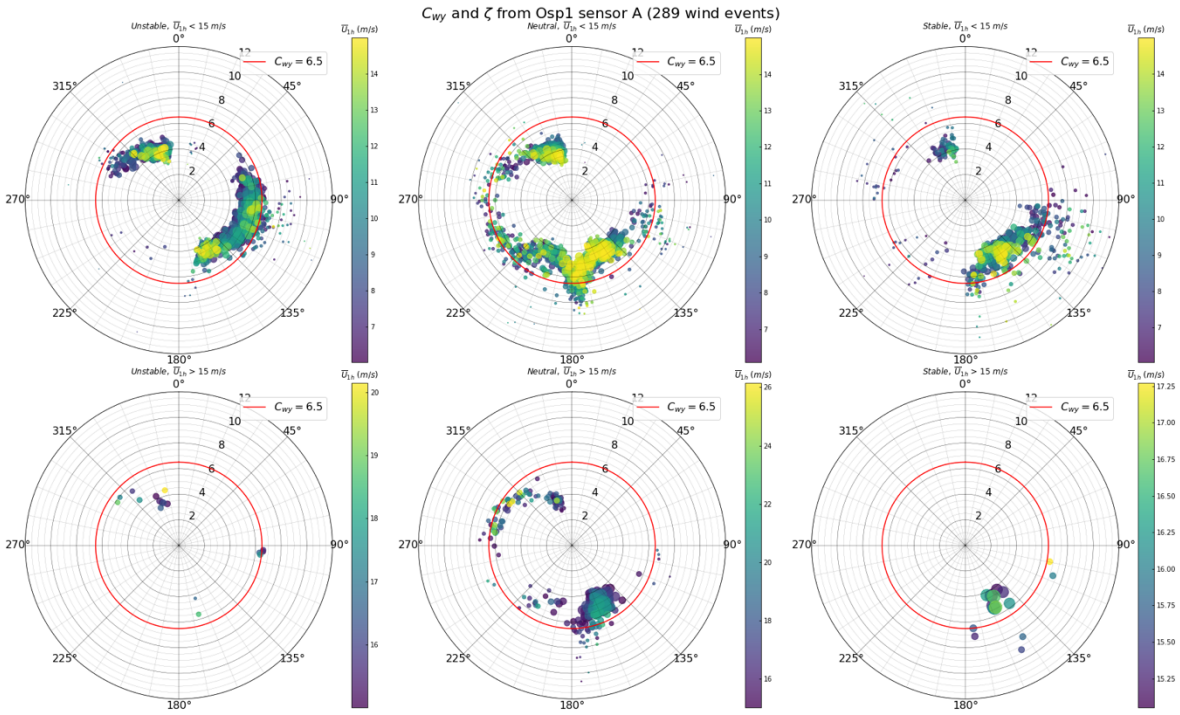

Figure B2.3: Distribution of the fitted C_{wy} with stability at Osp1.

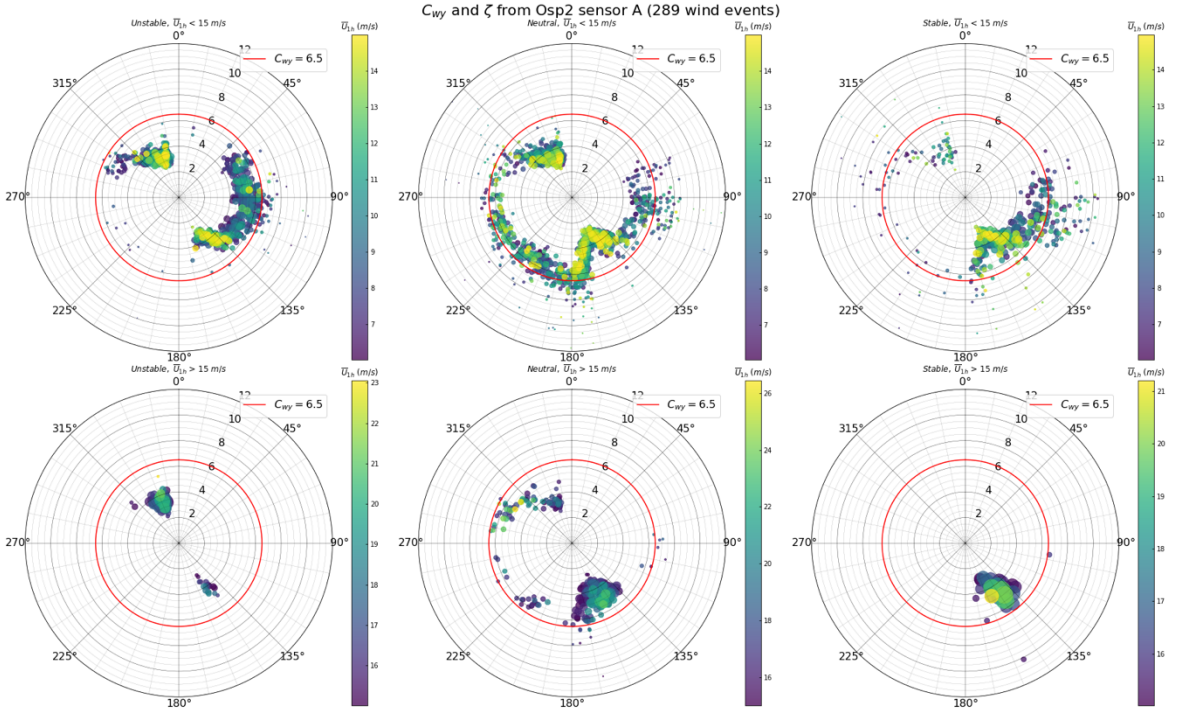

Figure B2.4: Distribution of the fitted C_{wy} with stability at Osp2.

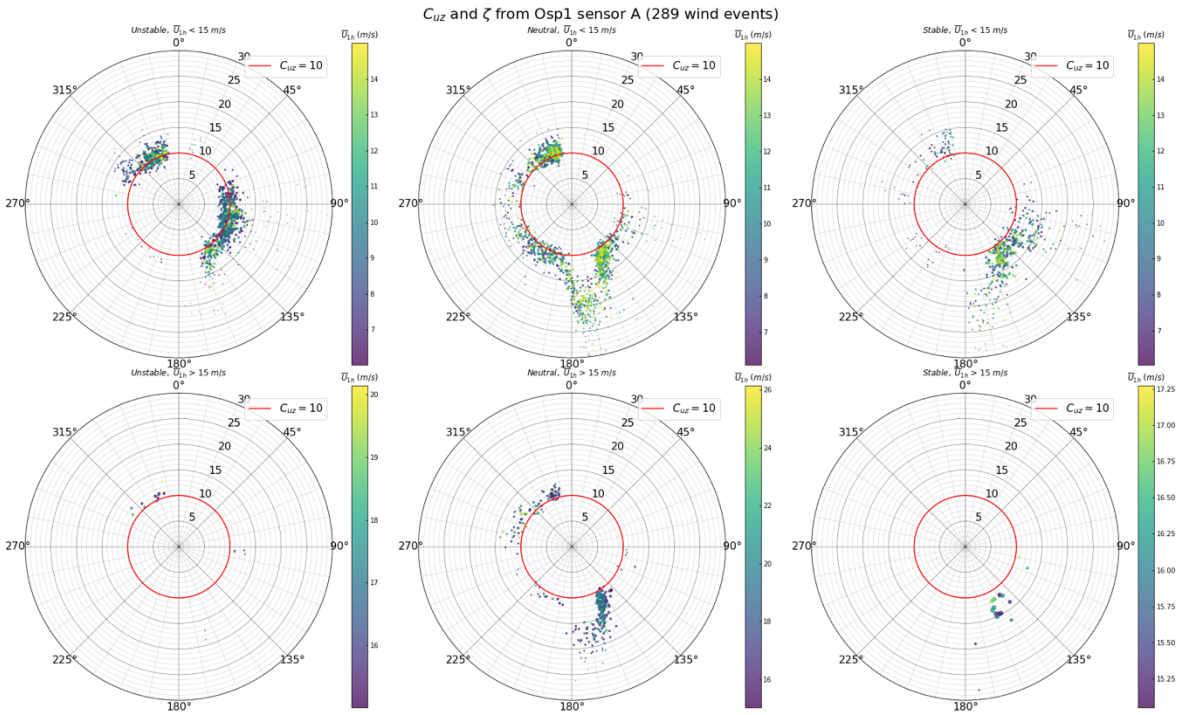

Figure B2.5: Distribution of the fitted C_{uz} with stability at Osp1.

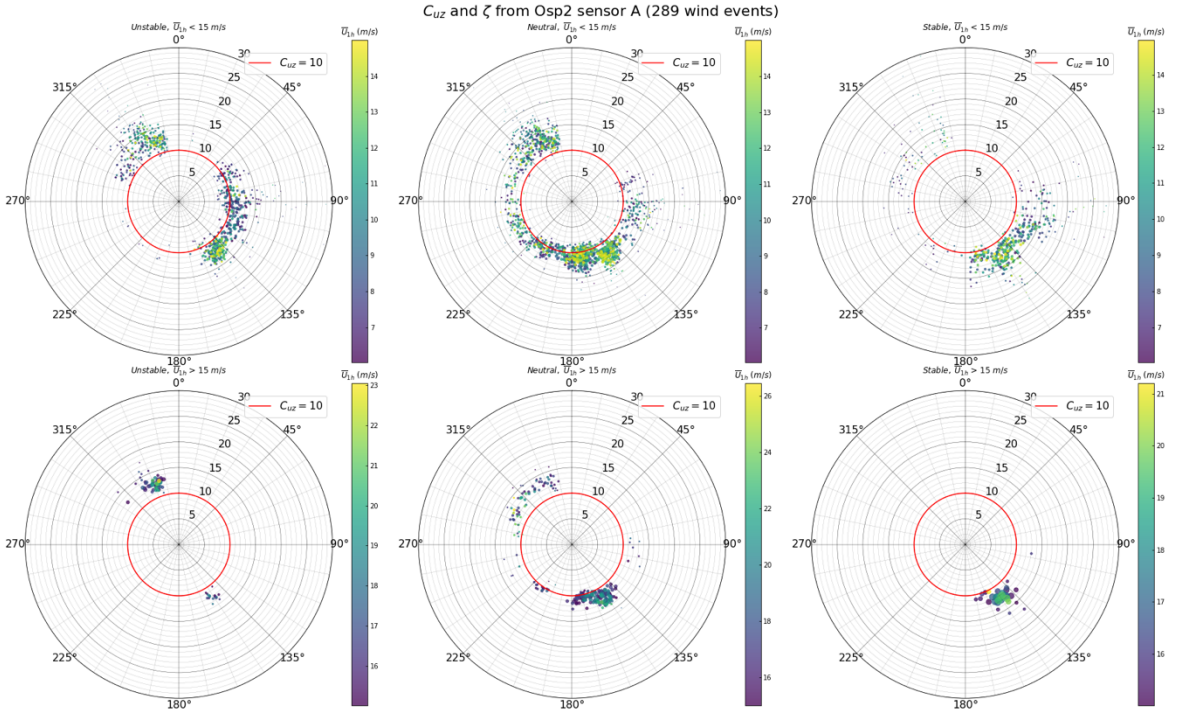

Figure B2.6: Distribution of the fitted C_{uz} with stability at Osp2.

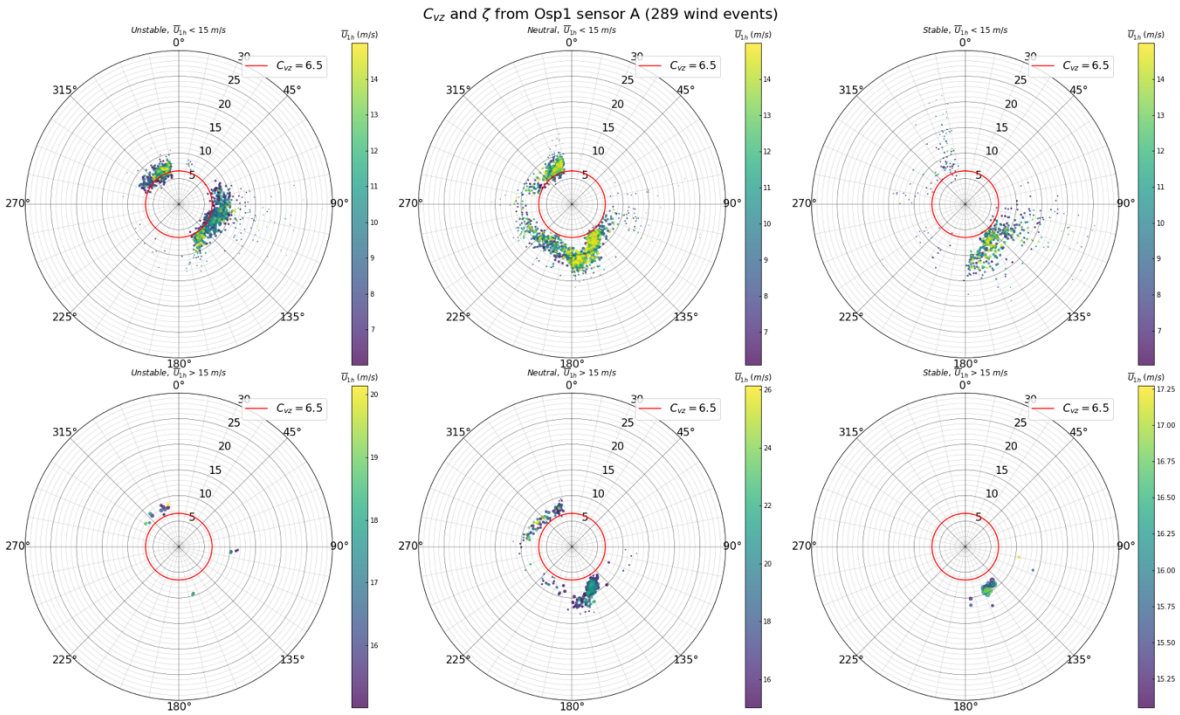

Figure B2.7: Distribution of the fitted C_{vz} with stability at Osp1.

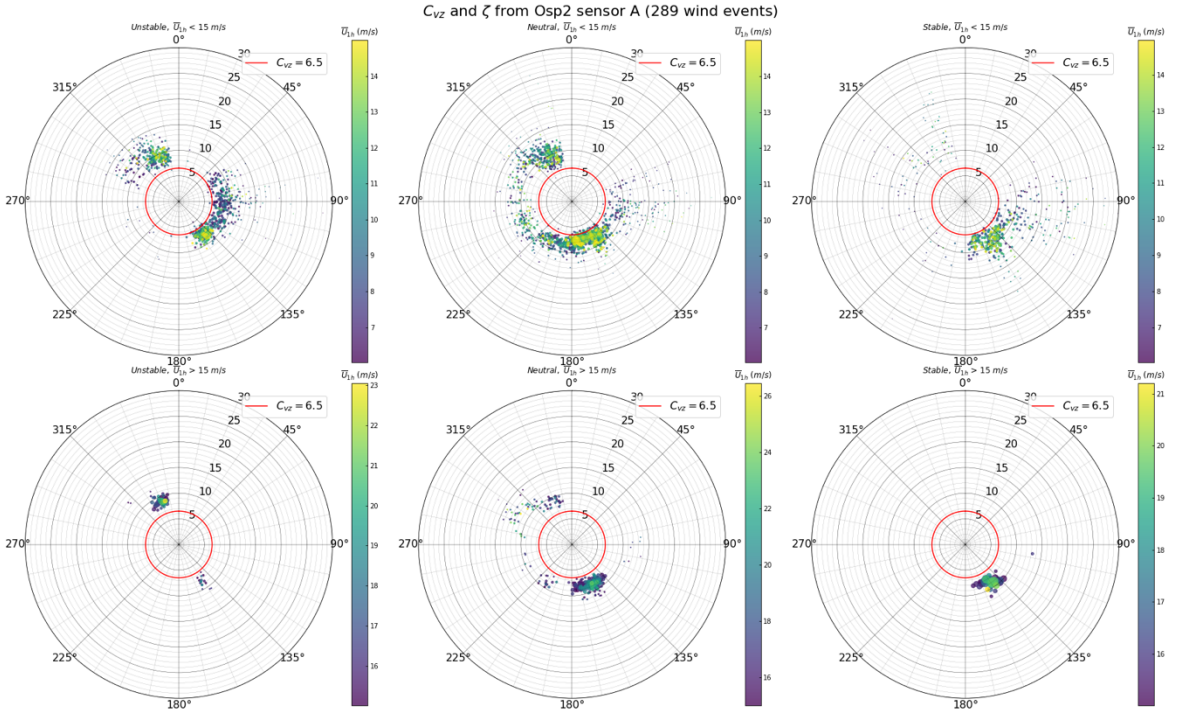

Figure B2.8: Distribution of the fitted C_{vz} with stability at Osp2.

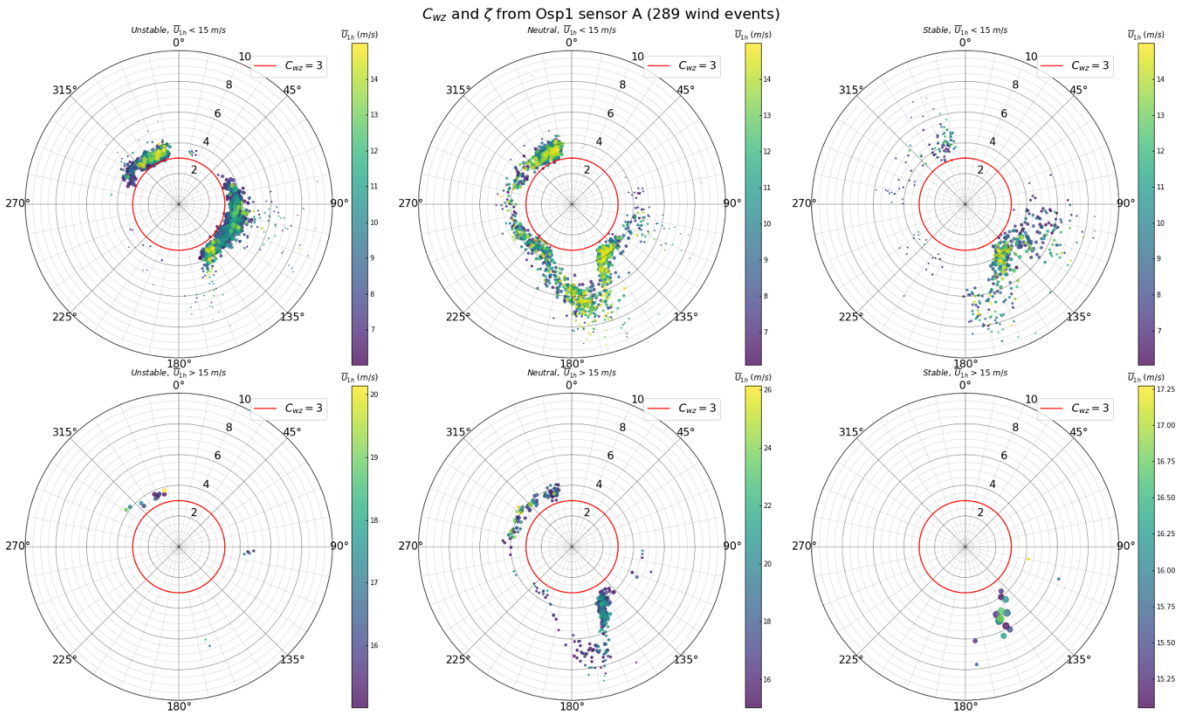

Figure B2.9: Distribution of the fitted C_{wz} with stability at Osp1.

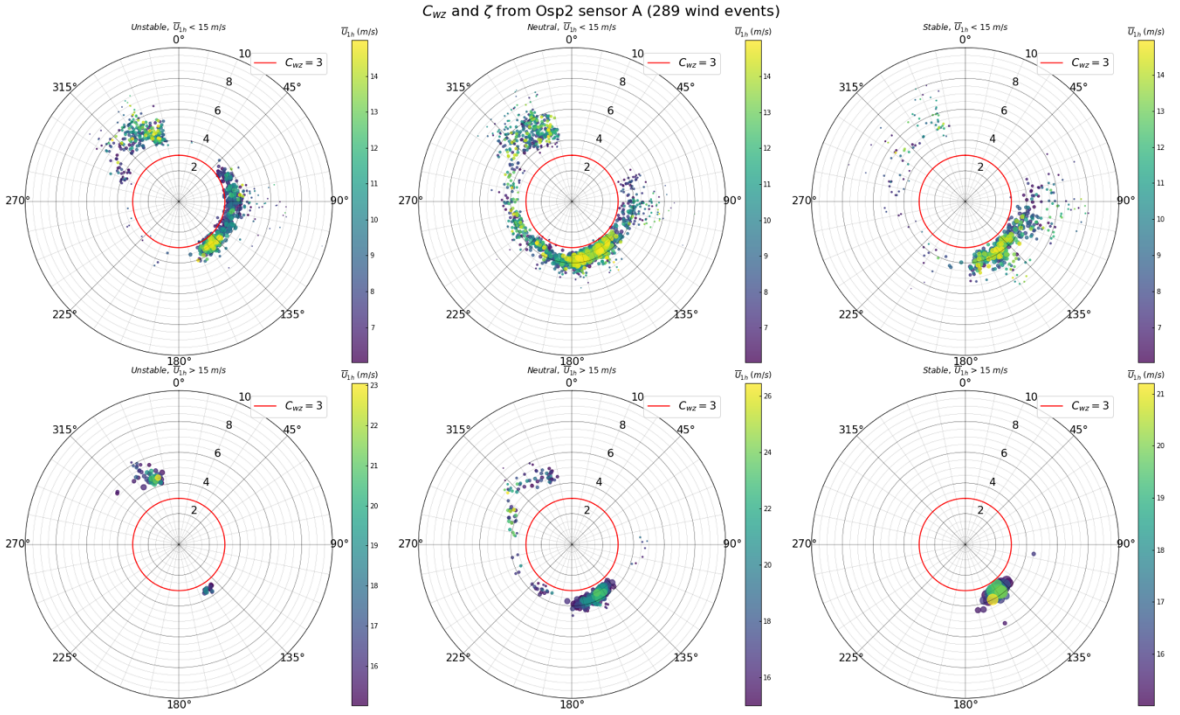

Figure B2.10: Distribution of the fitted C_{wz} with stability at Osp2.