University of Porto

Faculty of Engineering

Perdigão Data Retriever

João Sebastião Brito Carrapa Ribeiro de Carvalho

Supervisor: Prof. José Manuel Laginha Mestre da Palma

Co-supervisor: Vasco Teófilo Preto Batista

Dissertation submited to the University of Porto
in partial fulfillment of the requirements for the
degree of Master in Mechanical Engineering.

September 2022

Contact information:
João Sebastião Brito Carrapa Ribeiro de Carvalho
email: up201303465@up.pt

Abstract

Researchers working with Perdigão-2017 data faced the problem of accessing the data in an easy and fast way, the data files are in a specific format, and it took a lot of time and knowledge to access the files manually and retrieve the data from them. The main requirement was to develop an app –Perdigão Data Retriever– to retrieve and process in a straightforward and user-friendly manner, the Perdigão-2017 campaign sonic anemometer data. Additionally, a separate tool was developed to allow for automated comparisons with computational simulations performed on the VENTOS®/M computational fluid dynamics code. The data was analysed for availability, spanning a preliminary measurement period and the entire intensive operational period of the campaign. To demonstrate the capabilities of both tools, a brief analysis was performed by comparing 5-min averaged sonic anemometer data with VENTOS®/M simulation results spanning a 24-hour period from May 14 2017, 18:00 UTC. It was concluded that the Perdigão Data Retriever is a time-saving, powerful tool to retrieve data from the Perdigão files (speedup of at least 18 times). It was also concluded that the Availability Map is a practical manner to check for data availability and Data Comparison tools are an effective way of comparing data from the Perdigão files against the data from simulations results from VENTOS®/M.

Resumo

O acesso aos dados da campanha de Perdigão-2017 é difícil, os ficheiros estão num formato específico, e requer tempo e conhecimento para a manipulação dos mesmos. O principal requisito deste trabalho foi desenvolver uma aplicação –Perdigão Data Retriever– para aceder aos dados (nomeadamente as medições dos anemómetros sónicos) de um modo rápido, simples e eficaz. Foi desenvolvido também um conjunto de ferramentas para comparações automáticas entre os dados das medições da campanha de Perdigão-2017 e resultados de simulações computacionais de escoamentos atmosféricos sobre o mesmo local, obtidas com o código VENTOS®/M. Foi verificada a disponibilidade dos dados, abrangendo um período de medições preliminares e o período de observações intensivas da campanha. Para demonstrar o desempenho das ferramentas desenvolvidas, uma breve análise foi feita comparando os dados de períodos de médias a 5-min dos anemómetros sónicos com resultados computacionais do VENTOS®/M, abrangendo um período de 24 h com início em 14 de Maio de 2017 às 18:00 UTC. Concluiu-se que a aplicação Perdigão Data Retriever é uma ferramenta potente e que permite uma poupança de tempo (de pelo menos 18 vezes) para aceder aos dados da campanha. O mapa de disponibilidade é práctico para verificar a disponibilidade dos dados e as ferramentas de comparação permitem uma rápida avaliação dos dados e uma comparação eficiente de medições com resultados computacionais do VENTOS®/M.

Acknowledgements

I would like to thank Prof. José Laginha Palma for his guidance and valuable observations.
I am deeply grateful to Vasco Batista for all the help and mentoring with the tools development and for providing scripts that were essential for function building.
I would like to thank Isadora Limas Coimbra and Jesus Monteiro for all the helpful advice.
I would like to thank my parents for all the support and patience throughout the entire course.
I would like to thank Nina Grillo for always being there.
I would like thank Bela Santana for being the best dog.
This accomplishment would not have been possible without any of them.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 The Problem

Researchers working with Perdigão-2017 (Fernando et al., 2019) data faced the problem of accessing and processing the data in a fast and easy way: the files containing the data are in a specific file format (netCDF, see Section 1.3) and it took a considerable amount of time and knowledge to access the files manually and retrieve the data from them. This problem presents a challenge for anyone that aims to process the dataset from the Perdigão-2017 campaign, leading to an inevitable loss of time until the successful extraction of the data.

Additionally, researchers working with these measurements mostly limited their analysis to a small subset of the data. For example, Batista (2019) used the 100 m mast data (towers – `tse04`, `tse09` and `tse13`) to validate simulation results (Palma et al., 2019) using the VENTOS®/M CFD code (Rodrigues et al., 2016). Palma et al. (2020) used data from the same masts in their work, now focusing on VENTOS®/2 (Castro et al., 2003) results. The post-processing and comparison or validation process was mostly manual and user-case dependent, while requiring knowledge on the existing toolset to process netCDF files. This, in turn, meant that analysing the full dataset would be a time-consuming effort.

1.2 Perdigão-2017 Campaign

The Perdigão-2017 field experiment (Fernando et al., 2019; Mann et al., 2017) had the purpose of collecting atmospheric data over complex topography. Sonic anemometers, water vapor/carbon dioxide sensors, standard temperature relative humidity (TRH) sensors, barometers, radiometers, etc. are among the 344 instruments (Perdigão Layout, 2022) deployed during the campaign, which ranged from November 29, 2016 to July 27, 2017.

1.3 Contextualizing netCDF

NetCDF (Network Common Data Form, UCAR (2022)) is a set of software libraries and machine-independent data formats that support the creation, access, and sharing of array-oriented scientific data. It is also a community standard for sharing scientific data. It is commonly used as a file format for storing multidimensional scientific data (variables) such as temperature, humidity, pressure, wind speed, and direction. Each of these variables can be displayed through a dimension (such as time).

Although it is commonly used for the storage of this type of data, the process of extracting and processing data from netCDF files is not easy and swift for users not accustomed to work with such files, and even for users with previous experience with netCDF, it is necessary to know how each specific file is organized in terms of dimensions and variables, to browse the file.

1.4 The Proposed Solution/ Requirements

It was proposed to develop a program or set of tools in Matlab or Python that would allow to automatically retrieve sonic anemometer data, process and filter the data and export it to file types that allow an easy management of the data (ASCII, such as `.txt` or `.csv`).

There were some main requirements that will be analysed with further detail in the Methodology Chapter 2. Briefly, those requirements concern:

- Data Gathering - a module should handle the tasks for data gathering. The data is to be obtained from Perdigão FEUP ((Gomes et al., 2020)) and UCAR Perdigão Catalog ((NCAR-EOL, 2022)) websites.
- Basic Filtering - filter the extracted data by, time period of interest, mast(s) or height(s).
- Export - the data to be exported can be determined by the user by using flag variables and should obey specific formats.
- Data Analysis and Comparison with Simulation Results - comparison operations between retrieved data and simulations from VENTOS®/M should be performed.

1.5 Thesis Outline

This chapter presents the problem, contextualizing Perdigão Campaign and netCDF files, proposed solution and requirements. Chapter 2 concerns the development and explanation of all tools. Results and outputs of the tools are displayed in Chapter 3 and finally, the main conclusions are consolidated in Chapter 4.

Chapter 2

Methodology

In this chapter, the methodology followed for the development of the proposed tools is presented. The Perdigão-2017 dataset is explored in Section 2.1. The development environment and programming language are unfolded in Section 2.2. App design, modules and functions for the Perdigão Data Retriever are described in Section 2.3. Section 2.4 concerns production, functions description and readability of the Availability Map. The development, modules and functions description, and output types of Data Comparison Tools are in Section 2.5.

2.1 Perdigão-2017 Dataset

The Perdigão-2017 dataset is composed of a collection of netCDF files (Section 1.3) that contain measurements from a variety of equipment (see Section 1.2), for varying time periods of the measurement campaign, which ranged from November 29, 2016 to July 27, 2017. The full dataset can be obtained from:

- The Perdigão campaign website: `https://perdigao.fe.up.pt`
- NCAR catalogue: `http://catalog.eol.ucar.edu/perdigao`

The subset (`.nc` files) explored in this work pertains to "NCAR-EOL Quality Controlled 5-minute ISFS surface flux data, geographic coordinate, tilt corrected", i.e., 5-min averaged measurements, corrected for boom orientation and tilt, and thus rotated into a geographic coordinate system (for more information see `https://data.eol.ucar.edu/dataset/536.011`). The equipment of interest for this work are uniquely the 3-D sonic anemometers (see NCAR-EOL (2022) for details on the processing applied). For this set, data exists only for the time range between November 29, 2016 to July 1, 2017.

Processing the original 20 Hz dataset was not considered for this work. However, care was taken in the development of the tools in this work, so that minimal changes might enable the processing of the high-frequency dataset. This adaptation is reserved for future work.

In the Perdigão campaign website, each of the files has an URL that is a sequence of website location, directories path inside the website and filename. All filenames are equal, except for a string that represents the file date (in the format YYYYMMDD),

i.e., each 5-min average file contains data for precisely one day, from 00:00 UTC. This characteristic will be used to download the files by date (Section 2.3.2).

The data in the `.nc` files is stored for each variable as an array containing the averaged data for 5-min periods, identified by a timestamp that matches the time (in seconds) to the middle of the period. The variables of interest in this work concern the 3-D sonic anemometers, and are listed below:

- `u_<YY>m_<XX>` - u wind speed component
- `v_<YY>m_<XX>` - v wind speed component
- `w_<YY>m_<XX>` - w wind speed component
- `u_u__<YY>m_<XX>` - u wind speed variance
- `v_v__<YY>m_<XX>` - v wind speed variance
- `w_w__<YY>m_<XX>` - w wind speed variance
- `u_v__<YY>m_<XX>` - u, v wind speed covariance
- `u_w__<YY>m_<XX>` - u, w wind speed covariance
- `v_w__<YY>m_<XX>` - v, w wind speed covariance
- `dir_<YY>m_<XX>` - wind direction
- `spd_<YY>m_<XX>` - wind speed (magnitude)
- `base_time` - the base time variable contains one value, the time of the start of the file, as a number of POSIX (non-leap) seconds since 1970 Jan 1, 00:00 UTC
- `time` - time variable with a time dimension whose values represent the number of seconds of the middle time of the 5-min period, since the base time, which provides a human-readable representation of the time in the file

Where <YY> and <XX> stand for approximate sonic anemometer height and tower code name (e.g. 10, tnw01). For more information on the convention used for the variable names, see NCAR-EOL (2022).

The .nc files contain data for latitude and longitude of each tower. The units are in Decimal Degrees (WGS84 Lat/Long). Those variables will not be used to export the coordinates of the sonic anemometers since latitude and longitude values (`.nc` files) are of low precision (low digit count). Instead, the coordinates available in Perdigão Layout (2022), using the PT-TM06/ETRS89 system, were used to create a file (dataset) with all sonic anemometers coordinates.

2.2 Development Environment

The programming language used to develop the tools was Python 3.9. The preference was primarily due to the existence of modules that assist with netCDF file operation (`netCDF4`) and simplify data manipulation (`pandas`, `numpy`) and file exporting.

Development tests and debugging were carried out in the Integrated Development Environment (IDE) provided by SPYDER, included the Anaconda 3 Python distribution (Anaconda). The hardware used was an Asus VivoBook 15 laptop, with specifications listed below:

- Operating System - Windows 10 Home
- Processor - Intel(R) Core(TM) i7-1065G7 CPU @ 1.30GHz 1.50 GHz
- Installed RAM - 12,0 GB
- System Type - 64 bits operating system, x64 based processor

The aforementioned Python modules or packages were some of the used to develop the tools in this work. Some of them are included with the Python standard library and others are external, i.e., must be installed separately or by using the Anaconda distribution. The used Python standard library modules:

- `os` - this module implements some useful functions on pathnames, it is used in the developed tools for calling the `os.path.join(path, *paths)` function.
- `requests` - allows to send HTTP requests using Python, returning a Response Object with all the response data (content, encoding, status, etc). Used to download the netCDF files from the Perdigão website.
- `time` - the Python time module provides multiple ways of representing time in code, such as objects, numbers, and strings. Used in code for time representation (timestamp) manipulation.

The external modules used were:

- `pandas` - an open source Python package that is most widely used for data science/data analysis and machine learning tasks. It is built on top of the NumPy library, which provides support for multi-dimensional arrays. It is used widely in the tools to manipulate and process data. For more information see Pandas (2022).
- `numpy` - NumPy is a Python library used for working with arrays. It also has functions for working in domain of linear algebra, Fourier transform, and matrices. It is used widely in the tools to manipulate and process data. For more information see Numpy (2022).
- `netCDF4` - Netcdf4-python is a Python interface to the netCDF C library. It allows to retrieve data from netCDF files into a Python environment. Used in the tools to retrieve data from the netCDF files. For more information see Whitaker.
- `Matplotlib` - Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python, and create publication quality plots. Used for graph plotting. For more information see Matplotlib (2022). The following submodules were used:
  - `Pyplot` - a collection of functions/methods used for plotting simple 2D graphs using Python. Used for graph plotting.
  - `ticker` - contains classes for a refined configuration of tick location and for-

matting. Generic tick locators and formatters are provided, as well as domain-specific custom ones. Used for graph plotting.

The source code for the developed tools and the availability map creation are available at (Carrapa, 2022). The Perdigão Data Retriever is to be supplied along with the dataset at (Gomes et al., 2020).

2.3 Perdigão Data Retriever

For a better understanding of the Perdigão Data Retriever, the app architecture, representing how the different modules and main functions are connected, is presented in the flowchart of Figure 2.1. The app can be conceptually divided in three phases:

1. Phase 1 - queries the user for:
    - the time period (beginning and ending dates) for which the data is to be retrieved;
    - option to re-sample data (period conversion factor);
    - option on how/if to section time (time sectioning mode);
    - option on height selection (height selection mode);
    - which sonic anemometers the data is to be retrieved. This can be supplied through the individual sonic anemometer identifiers or mast identifiers (if all sonic anemometers from the mast are to be exported).
2. Phase 2- processes the inputs, downloading the required `.nc` files, calculating additional variables and performing time averaging operations depending on the target time-step supplied by the user.
3. Phase 3 - Separate ASCII files are exported for each sonic anemometer, spanning the full selected period. A file containing the coordinates (ETRS89/PT-TM 06) of the processed sonic anemometers is also exported.

The exported files are in ASCII format, with file type options as .csv (comma separated) or .txt (space separated). After all exports are made the user is queried whether to terminate or restart the execution of the program (Yes/No) (e.g. It can be necessary to restart the app if the user wants a new set of dates).

The app runs on variables defined by the user. There are two main group of variables, the ideological difference between them resides in the inner working of the app:

- Time related variables – define the time period or periods of all exported files.
- Sonic anemometer variables – define which sonic anemometers data will be exported after processing.

The sonic anemometer variables consist of a pair of variables that define and identify which sonic anemometer is referred. Those variables are:

- `tower_code_name` - used as tower identifier (e.g. 'tnw01', using the tower designation from NCAR, see Perdigão Layout (2022).)

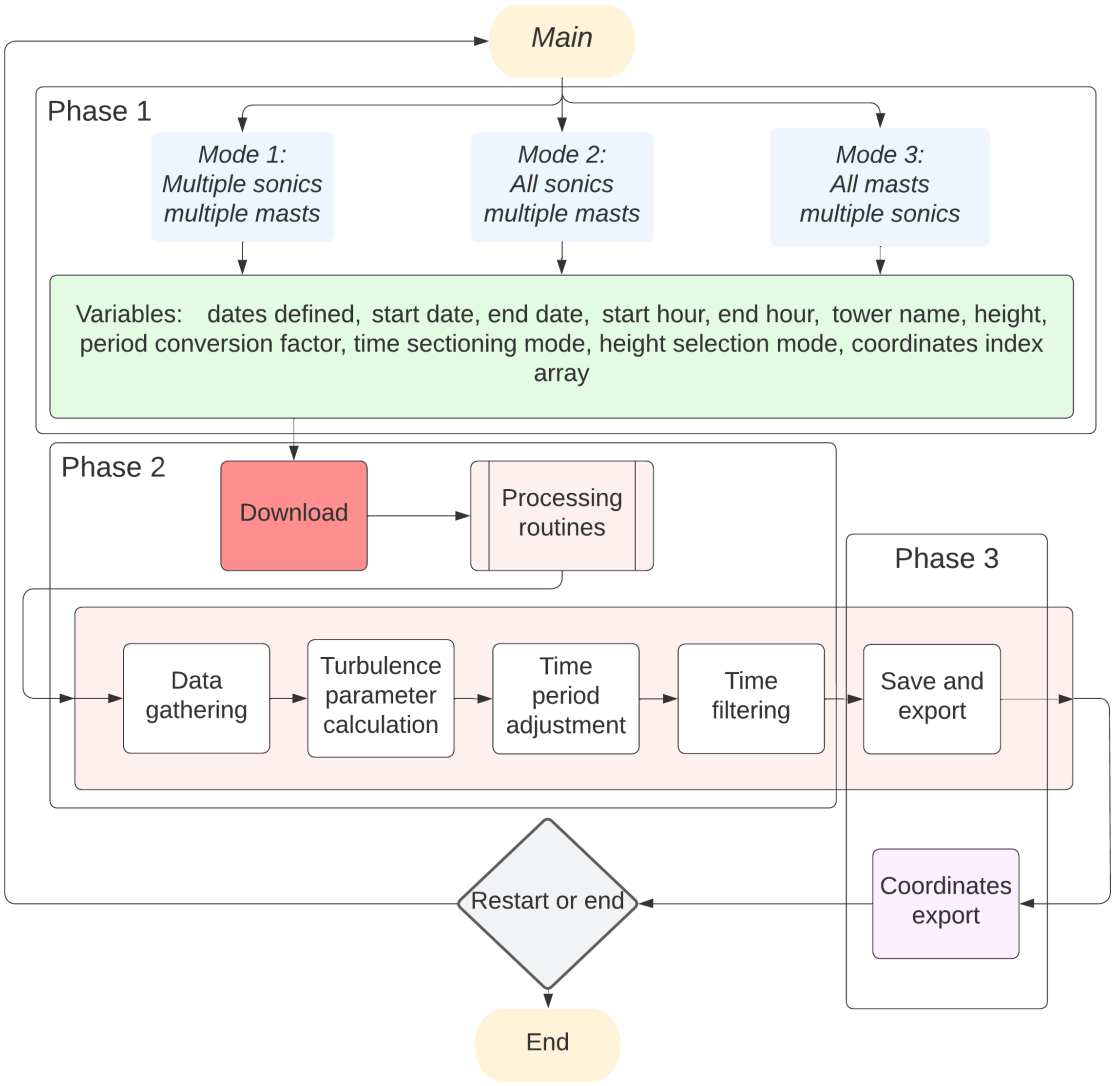

Figure 2.1: Flow chart for the Perdigão data retriever app.

- `sonic_height` - used as height identifier (e.g. '10').

After all user defined variables are set (Phase 1, Section 2.3.1), the app will loop through every selected sonic anemometer and download, calculate new parameters (turbulence), perform time resampling (if needed) (Phase 2, Section 2.3.2) and export data (Phase 3, Section 2.3.3) for each one of them, producing files containing the target data for each selected sonic anemometer. The app also produces one file containing all the selected sonic anemometers coordinates (will be used to obtain VENTOS®/M simulation files, Section 2.5.1).

2.3.1 Phase 1

The app is executed through the `main_perdigao_app.py` module (Appendix B.1.1). Upon execution, the user is presented with three options, to choose which sonic anemometers the data is to be retrieved from. Those options are:

- `Mode 1 - Multiple sonics, multiple masts`: The user selects any number of masts (through their identifier, e.g. `tse04`), and from those masts the user can select which sonic anemometers will be chosen for data retrieve.
- `Mode 2 - All sonics, multiple masts`: The user selects any number of masts, the data is retrieved from all available sonic anemometers in each of the chosen masts.
- `Mode 3 - All masts, multiple sonics heights`: The user selects any number of sonic anemometers height (e.g. 10, 20 and 30), the data is retrieved from all sonic anemometers at the selected heights.

These modes are used only to define the sonic anemometers to process. The time period of interest to the user is queried beforehand and the settings are applied to all selected anemometers.

The different execution modes were devised as a time-saving measure for the user, since `Mode 1` allows to define any combination and set of sonic anemometers. Some use-cases are now exemplified for the two latter modes: should the user want to select all sonic anemometers for five towers, `Mode 2` provides a simpler interface when compared to `Mode 1`, as the former requires only tower identifiers to be input, whereas the latter also queries for sonic anemometer heights. Should the user want to select all sonic anemometers at a specific height from the ground (for every tower), `Mode 3` is the more appropriate.

Regarding time variables the user is queried on `start_date, end_date, start_hour, end_hour`. The user is also queried one of the following options (represented internally by a flag variable, `sm`) that define the meaning of the start and end hour:

- Full days – data is presented with all time periods from 0:00 to 24:00 in every selected dates (`start_hour = 0, end_hour = 24,`);
- Start and end hour for every day – data is presented in a discontinuous way only for the time periods between a chosen start hour and end hour for every selected dates (e.g. from 12:00 to 18:00 every day);

- Start hour for the first day and end hour for the last day – data is presented in a continuous way from a chosen start hour for the first day until a chosen end hour for the last selected day (e.g. from 3:00 of the first day until 20:00 of the last day).

After Phase 1, and the definition of the time period, time-step, height selection option (Mode 1) and the selection of the sonic anemometers of interest, the download and processing are executed. The routines that handle these functionalities are explained in Section 2.3.2 and documented in Appendix B. The queried variables are in Appendix A.1

It was not the objective of this work to implement a graphical user interface (GUI) for the developed tools. The main reason is that the tool is to be used mainly in high performance computing (HPC) systems that often require execution and interaction using a command line interface (CLI). The implementation of a configuration file instead of the existing interactive interface could be of interest for further automation when executing in high-performance systems.

2.3.2 Phase 2

As described in Section 2.1, each 5-min averaged netCDF file contains data for 24 hours of measurements, starting at 00:00 (UTC). The corresponding date is specified in the filename which appears in the URL for the file. Therefore, to download the files, the app needs only to identify all dates contained between the start and end dates supplied by the user. Files are downloaded to the directory where the app is executed and remain there unless manually removed, as such, the download function (Appendix B.1.8) first verifies if the files already exist and, in an affirmative case, the download is skipped.

Once all required files are in the local directory, the data processing routines are executed. These are contained in a single module file (`processing_functions.py`), and the functionalities include: data extraction from the netCDF files, time-averaging and filtering, turbulence parameter calculation (turbulence kinetic energy, turbulence intensity), and data export to ASCII filetypes.

The routines or functions are executed independently of the selected mode, i.e., the modes affect the user input (selection of sonic anemometers to process) and the processing and export is applied for each of the selected anemometers by looping through them.

Data Gathering

This sub-module contains functions to create a Pandas dataframe (2D array) with the 5-min averaged data for a single sonic, spanning all dates defined by the user (see Figure 2.2 for an example). The functions also allow to define the variables of interest which were considered to be all of the available in the netCDF files (in Section 2.1 and Appendix A.1), i.e., the functions concatenate the data from all files, with a timestep in rows and variables data for each 5-min period displayed in columns.

index	basetime	time	u	v	w	vh	dir	uu	vv	ww	uv	uw	vw
2017-06-01 00:02:30	1496275200	150	-0.401577	0.139434	-0.034372	0.461961	109.148	0.0108188	0.0383377	0.00986139	0.00234165	-0.000517247	0.00934236
2017-06-01 00:07:30	1496275200	450	-0.707295	0.11903	-0.107365	0.739618	99.5527	0.0284182	0.0283245	0.0111803	0.0120894	0.0078463	0.00591427
2017-06-01 00:12:30	1496275200	750	-0.429807	0.264233	-0.0271241	0.536085	121.582	0.0423122	0.0240137	0.0102108	0.00523133	0.00702757	0.00279718
2017-06-01 00:17:30	1496275200	1050	-0.52571	0.336632	-0.0697195	0.649877	122.633	0.0165215	0.0338669	0.00984704	0.00239533	0.00031979	0.00165458
2017-06-01 00:22:30	1496275200	1350	-0.430578	0.0427964	-0.100227	0.468108	95.6761	0.0133712	0.0390396	0.00691302	-0.00508677	-0.000506032	0.00380979
2017-06-01 00:27:30	1496275200	1650	-0.15993	-0.0901408	-0.0382429	0.207429	60.5932	0.0161834	0.0177805	0.00620437	0.00741224	0.00336457	0.00579415
2017-06-01 00:32:30	1496275200	1950	-0.0914043	-0.219948	-0.0784813	0.257025	22.5664	0.0109136	0.0138744	0.00360582	0.00272621	9.89465e-05	0.00374366
2017-06-01 00:37:30	1496275200	2250	-0.0375346	-0.200756	-0.0622852	0.222339	10.5901	0.00729453	0.0140681	0.00328618	-0.000924632	-0.000115008	0.00383846
2017-06-01 00:42:30	1496275200	2550	-0.0645259	-0.0527107	-0.0273327	0.128024	50.7549	0.00699486	0.0133227	0.00284458	-0.00034061	-0.000470801	0.0013714
2017-06-01 00:47:30	1496275200	2850	-0.101996	-0.036533	0.00904546	0.147202	70.2936	0.00774967	0.0171696	0.00605535	0.00323497	-0.00217783	0.00203616
2017-06-01 00:52:30	1496275200	3150	-0.172018	-0.0355854	-0.0290855	0.197973	78.3121	0.00765863	0.0133686	0.00324364	0.0046995	-0.0013967	0.00165937
2017-06-01 00:57:30	1496275200	3450	-0.147399	-0.0324594	-0.00839061	0.176467	77.5808	0.0105661	0.0179473	0.00503753	0.00529585	-0.0029606	0.0033498
2017-06-01 01:02:30	1496275200	3750	-0.359202	-0.0898052	-0.0878842	0.379039	75.963	0.0192328	0.0123668	0.00569761	0.00817307	-0.00150024	-7.75745e-05
2017-06-01 01:07:30	1496275200	4050	-0.317044	0.0383109	-0.0349767	0.351566	96.8901	0.0114468	0.0242634	0.00855123	0.00201394	0.000210353	-0.00221664
2017-06-01 01:12:30	1496275200	4350	-0.225153	0.106083	-0.0375135	0.342147	115.228	0.0216183	0.0545704	0.00913667	0.009766	0.00186876	0.00697332

Figure 2.2: Example of a data frame returned by the `data_gathering` module.

A Pandas data frame is a data structure that also contains labeled axes (rows and columns). It is the primary Pandas data structure, used in this app as the preferred way of storing data due to it's easy an practical methods for data manipulation, calculus and the labeled axes for supporting all operations. Pandas dataframes were used in detriment of NumPy arrays, as the former allow to have labelled axes (which were a great help to visual inspection during development), data heterogeneity (integers and floats) and a practical export of labelled data.

The data frame that this sub-module provides will be manipulated and used for calculation of new parameters until it meets the user choices for data exporting.

Time period adjustment

This sub-module handles the temporal resampling of the 5-min averaged data. If, when queried, the user has chosen a time period other than the standard for the data (5-min), and divisible by the same value (e.g. 20-min), the data is arithmetically averaged for the wind speed (magnitude and components) and direction. The process is not as straightforward for the average variances and covariances of the wind speed: these would be calculated using the "instantaneous" measurements (20 Hz data) and the average for the new target period. However, to avoid using the 20 Hz data, a re-sampling formula (Equations 2.1 and 2.2, Rodrigues et al. (2016)) was applied. Nevertheless the author notes that for a correct calculation of the variances and parameters that depend on these values, the 20 Hz data should be used. A new data frame is created with the resulting values for all variables.

$$\overline{u'u'} = \left(\frac{\overline{u_1^2} + \overline{u_2^2} + \cdots + \overline{u_n^2}}{n}\right) + \left(\frac{u'u'_1 \cdot u'u'_2 \cdot \cdots \cdot u'u'_n}{n}\right) + \left(\frac{u_1 + u_2 + \cdots + u_n}{n}\right)^2 \quad (2.1)$$

$$U_j = \left(\frac{u_1 + u_2 + \cdots + u_n}{n}\right) \quad (2.2)$$

Variance for new averaged period u wind component ($\overline{u'u'}$), using 5-min averaged period data ($\overline{u_i^2}$, $u'u'_i$ and u_i, with i as period index). Mean wind speed component for new averaged period (U_j, with j as wind component index). Extended equations in Appendix C.

Turbulence Parameters

This sub-module and its functions are interconnected to the former: if a time adjustment (temporal resampling) is needed, the turbulence parameter calculation is executed before the time adjustment, since those calculations can only be performed from the 5-min data.

Here, the data frame provided by the `data_gathering` sub-module is used (through wind components – `u,v,w` – and variances – `uu,vv,ww`) to calculate three new turbulence parameters – turbulence kinetic energy (TKE, Equation 2.3), turbulence intensity (TI, Equation 2.4) and turbulence intensity using horizontal wind speed components (TIH, Equation 2.5). The `time_period_adjustment` module is then called and are appended to the resulting data frame, at each timestep, those turbulence parameters (Figure 2.3).

$$\mathrm{TKE} = \frac{1}{2}\left(\overline{u'u'} + \overline{v'v'} + \overline{w'w'}\right) \tag{2.3}$$

$$\mathrm{TI} = \frac{\sqrt{\frac{1}{3}\left(\overline{u'u'} + \overline{v'v'} + \overline{w'w'}\right)}}{\sqrt{U^2 + V^2 + W^2}} \tag{2.4}$$

$$\mathrm{TIH} = \frac{\sqrt{\frac{1}{2}\left(\overline{u'u'} + \overline{v'v'}\right)}}{\sqrt{U^2 + V^2}} \tag{2.5}$$

Index	basetime	time	u	v	w	vh	dir	uu	vv	ww	uv	uw	vw	tke	ti	tih
2017-06-01 00:10:00	1496275200	600	-0.516097	0.214832	-0.0596451	-0.0596451	113.229	0.0245177	0.0311357	0.0102749	0.00551444	0.0036691	0.0049271	0.0116781	0.156946	0.154356
2017-06-01 00:30:00	1496275200	1800	-0.179862	-0.117012	-0.069809	-0.069809	47.3565	0.0119407	0.0211907	0.00500235	0.00103176	0.000710618	0.00429652	0.0171576	0.473978	0.494689
2017-06-01 00:50:00	1496275200	3000	-0.121485	-0.0393221	-0.0139408	-0.0139408	69.2353	0.00824232	0.015452	0.00429528	0.00322243	-0.00175148	0.00210418	0.00100816	0.201832	0.190459
2017-06-01 01:10:00	1496275200	4200	-0.262003	0.0171616	-0.0493538	-0.0493538	95.8895	0.0166852	0.0309056	0.00784893	0.00730066	0.00123006	0.00295298	0.00611743	0.239036	0.238243
2017-06-01 01:30:00	1496275200	5400	-0.119936	0.0766311	0.00481599	0.00481599	156.753	0.0126346	0.0221491	0.00589239	-0.000687546	-0.000641784	0.00220598	0.00894451	0.542248	0.541715
2017-06-01 01:50:00	1496275200	6600	-0.0963364	-0.0539967	-0.0275616	-0.0275616	128.156	0.0132647	0.0242351	0.00659081	0.00386376	-0.000398001	0.00727329	0.012506	0.802193	0.804378
2017-06-01 02:10:00	1496275200	7800	-0.0957049	-0.0780039	0.0103413	0.0103413	134.822	0.00943456	0.0337301	0.0100935	-0.00103293	-0.00302217	0.0129039	0.0031967	0.372595	0.36069
2017-06-01 02:30:00	1496275200	9000	0.00508184	0.0292633	0.0049753	0.0049753	234.814	0.0111253	0.020193	0.00539343	-0.00142432	-0.00156887	0.00580777	0.00809251	2.439	2.44177
2017-06-01 02:50:00	1496275200	10200	-0.156214	0.129073	-0.012523	-0.012523	127.732	0.0175527	0.036644	0.0116769	0.00350601	-0.0052257	0.00870082	0.00836574	0.367836	0.360709
2017-06-01 03:10:00	1496275200	11400	-0.343988	0.145932	-0.0466294	-0.0466294	111.262	0.0264134	0.0607985	0.0157198	0.00313918	-0.00168923	0.0155206	0.0239578	0.335616	0.330952
2017-06-01 03:30:00	1496275200	12600	-0.328886	0.0506012	-0.0657221	-0.0657221	101.784	0.0335502	0.090549	0.0259821	0.00106509	-0.00332603	0.0298023	0.0076735	0.210871	0.211151
2017-06-01 03:50:00	1496275200	13800	-0.366578	0.187741	-0.0512926	-0.0512926	118.887	0.0214943	0.0722018	0.0187655	0.00100659	-0.00310861	0.0245736	0.0174788	0.260088	0.256566
2017-06-01 04:10:00	1496275200	15000	-0.25698	0.049264	-0.0441775	-0.0441775	98.4816	0.0209399	0.0666896	0.0177173	-0.000391392	-0.00207998	0.0250898	0.00550614	0.228317	0.228749
2017-06-01 04:30:00	1496275200	16200	-0.378976	0.0633729	-0.0706872	-0.0706872	161.967	0.0277576	0.0806703	0.0214076	-0.010479	-0.00102147	0.0279104	0.0474246	0.455122	0.459811
2017-06-01 04:50:00	1496275200	17400	-0.218699	0.15107	-0.0580896	-0.0580896	135.586	0.0139885	0.136115	0.0355405	-0.00665714	-0.00969428	0.0588957	0.0116568	0.324007	0.320127
2017-06-01 05:10:00	1496275200	18600	-0.156043	-0.021293	-0.0529388	-0.0529388	81.4232	0.0123592	0.139656	0.0362264	-0.00837324	-0.00922747	0.0626348	0.00322822	0.279216	0.291829

Figure 2.3: Example of data frame returned by the `turbulence_parameters` module, highlighting the time period conversion from 5-min to 20-min averaged data and the three new turbulence parameters appended to the data frame.

Time filtering/sectioning

This sub-module trims the existing data frame, if needed, according to one of the options chosen by the user (2.3.1) for time sectioning (`sm, start_hour, end_hour`) This constituted the last processing step and the data frame returned by this sub-module is to be saved and exported.

2.3.3 Phase 3

Save and export

This sub-module takes the data frame provided by the last sub-module and saves it into three file formats:

- `.csv` – comma separated variables file;
- `.xls` – excel file;
- `.txt` – text file.

These file types are easy to access for post data processing, which is the main objective of the app.

The files are exported to the data folder where the app is located and the files name include a description on start and end date(if the file has data for more than one day), start and end hour for each day and the chosen time period:

`<LL>m\_<XX>\_<YYYYMMDD>\_<YYYYMMDD>\_<HH>-<HH>h\_P-<MM>min.csv,`

with `LL` – sonic height; `XX` – tower code name; `YYYYMMDD` – date in the format Year-Month-Day; `HH` – hour of the day; `MM` – minutes of the time period.

Should a file with the same name already exist in the export folder that file is overwritten.

When the export of the file containing the Perdigão measurements data is concluded, the sonic anemometer coordinates data is returned to the mode where the `process_routines` module was called. The mode (1, 2 or 3) will continue to call the `process_routines` module (for different sonic anemometers) until all selected sonic anemometer files are exported. After all the Perdigão files are exported, a single file containing the coordinates and height of all selected sonic anemometers is also exported (format: `.dat`)

2.4 Availability Map

Following the development and first executions of the Perdigão Data Retriever app, users observed that there were anemometers with extended periods of no, or partial, data availability, due to, for example, late deployment in the campaign, or outages due to malfunctioning equipment. As such, a data availability map was sought to provide information on the availability of sonic anemometer data for the entire campaign.

2.4.1 Production of the data availability map

The production of the data availability map required the download of the entire 5-min averaged dataset for the campaign, and involved, for each file:

1. Data retrieval of 5-min averaged data (representing a full day of measurements).
2. For each sonic anemometer, and variable, a function checks for the existence and validity of data, returning an integer value depending on the result: 0 if no data exists for that sonic anemometer, for every variable, for the full day; 1 if the data is complete; 2 if there are some periods with valid data and others with invalid or missing data (e.g. `tnw05_2m` in '14/05/2017' just has data from 6:00 to 13:00); and 3 if there was any problem with the `availability_function`. The data filters for each variable can be seen in Appendix B.2.
3. The returned integers are stored in a Pandas DataFrame with a single column (day) and the number of rows equating to the number of sonic anemometers deployed during the campaign.

All dates in the Perdigão campaign are then looped through and the results are appended to the main DataFrame. The output is a 2D array with the dates for column index (e.g. format: 20170514) and the sonic anemometers identifier (`<tower>_<height>m`) in rows that is exported to a `.csv` file. The values stored, being integers, are to be processed to produce a graphic availability map.

Although conceptually simple to produce, the time required to process all files and obtain the availability array was large (7 h), using the hardware described in Section 2.2. This occurred due to memory limitations in the system, requiring the processing of a maximum of 20 consecutive days of data.

The filters used to determine the validity of the data were set with boundaries that would just exclude aberrant (e.g. a wind speed value of 100 $\mathrm{m\,s^{-1}}$) or impossible (e.g. direction value of 370°) values. There are some data errors that the `availability_function` does not cover, if a variable has the same value for some sequential periods (frozen data log) the filter does not exclude them. It could be a future work to re-do the availability map with a new set of filters. For that purpose, instructions come in Appendix B.2, as well as the filters values.

2.4.2 Availability map post-processing

The 2D availability array was imported to an Excel file and the values were assigned a colour. Functions were implemented in the Excel file to provide information on availability per day and availability per sonic anemometer.

The Excel file is located in a folder of the Perdigão Data Retriever folders, which is available at (Carrapa, 2022). It is required the Excel file for an appropriate visualisation of the full data, due to its extension.

2.5 Data Comparison Tools

One of the objectives of this work was to provide a framework to enable straightforward comparisons between sonic anemometer measurements and computational simulation data. To this end, the Perdigão Data Retriever (Section 2.3) contributed by converting the netCDF files, which require special libraries to be read and processed, into ASCII files. The files obtained from that app can then be easily used for direct comparisons with post-processed results (time-series) of atmospheric flow simulations, by importing them using any programming language or data visualisation program (e.g., Tecplot®, `https://www.tecplot.com/`). However, a separate tool was created in the effort to expeditiously provide both qualitative and quantitative comparisons between measurements and simulations. The first would be provided by time-series plots of variables of interest, and the latter by error estimates encompassing varying time-periods.

At this stage, the interest was in enabling the comparison framework for use with VENTOS®/M (Section 2.5.1) simulations over Perdigão. VENTOS®/M time-series files are also in ASCII format, and are described in detail in Section 2.5.1. For future works, however, the tools can be expanded to provide compatibility with results from other codes (e.g. WRF).

2.5.1 VENTOS®/M Files

VENTOS®/M produces 3-D time-dependent results (X-min averages) in netCDF format, that are post-processed using a set of provided tools that allow to obtain:

1. Time-series (ASCII, `.dat` filetype) at specific locations of the computational domain;
2. Three-dimensional subsets converted to ASCII or Tecplot® binary formats.
3. Two-dimensional horizontal slices in ASCII or Tecplot® binary formats.

For comparison with measurements files obtained from the Perdigão Data Retriever, the execution of post-processing tool (1) is required. This tool requires that two files be specified:

- A configuration file specifying: (a) start and end times for the time-series, (b) variables to output.
- An ASCII (`.dat`) file with a list of coordinates (x, y, z_{agl} – above ground level), space-separated, and in the coordinate system used in the simulation (typically WGS84/UTM29n for Perdigão)

A user that intends to directly compare simulation results with sonic anemometer measurements from Perdigão-2017 must ensure the provision of matching start and end times to the Perdigão Data Retriever and the VENTOS®/M post-processing tool (in step (a)), and that the average period is the same between simulation results and measurements (typically the default – 5-min). The variables selected for output (step (b)) must be the default, which include the averaged wind speed components (u, v, w), magnitude (vh), and direction (`dir`), turbulence intensities (ti, tih – already calculated) and variances ($u'u'$, $v'v'$, $w'w'$). The turbulence kinetic energy (TKE) for the simulation results

must be calculated using Equation 2.3.

The list of points to probe in the simulation results correspond to the locations of the sonic anemometers selected for processing on execution of the Perdigão Data Retriever: these are output to a `.dat` file (see Section 2.3.3) that matches the format requirements of the VENTOS®/M post-processing tool.

The sonic coordinates file, however, is exported using x, y coordinates in the original system: ETRS89/PT-TM06. Ideally, options should be provided to allow the user to specify a target coordinate system, to allow for a more seamless integration of the developed toolset with the existing VENTOS®/M toolset. However, this feature was not implemented prior to the writing of this thesis, requiring an external coordinate transformation. The feature will be considered for future works.

One difference between VENTOS®/M results and measurements that must be accounted for is the timestamp for each measurement (or 5-min average): whereas in the measurements, a 5-min period is represented by a timestamp in the midpoint of the period (e.g. from 00:00:00 to 00:05:00, the value is attributed to 00:02:30), in VENTOS®, the period is represented by its end time (00:05:00 following the previous example). This is corrected by shifting the measurements by half of the selected time period.

Simulation results are presented in this work with the aim of exemplifying the capabilities of the developed toolset. The results pertain to existing simulations, performed and analysed in previous works (Palma et al., 2019; Batista, 2019). As it is not the focus of this work, the reader should refer to those publications for additional information on the simulation's setup and execution.

2.5.2 Plotting and error estimation

The Graph Creation Environment is the module that handles the time-series plotting for qualitative analyses as well as the more quantitative error estimation between simulations and measurements. Here, by altering a starting portion of the source code, the user can define which sonics or towers will be compared and which set of variables will be compared, with the added option to calculate and export Root Mean Squared Error (RMSE) and Bias values for any of the available variables.

The variables that the user can set are:

- `plot_type`: variable to know which set of variables to plot. Possible values are:
  1. 'vh' , 'dir' , 'tke';
  2. 'u' , 'v' , 'w';
  3. 'uu' , 'vv' , 'ww';
  4. plots hourly RMSE series, encompassing the maximum and minimum values for 'vh' , 'dir' , 'tke' along all sonics of a selected tower;
  5. calculate RMSE and Bias values for 'vh' , 'dir' , 'tke' over any specified periods (24h divisible periods: 1, 2, 3, 4, 6, 8, 12, 24)(tables).
- `table_rmse`: flag variable to know if hourly RMSE table is to be created;

- `rmse_var`: variable that defines which field variable will be used for error calculation;
- `table_bias`: flag variable to know if hourly Bias table is to be created;
- `bias_var`: variable that defines which field variable will be used for error calculation;
- `towers`: array containing names of the towers from where the sonic anemometer values will be compared;
- `heights`: array containing sonic anemometer heights if the user wants to specify it (optional).

Options for `plot_type = 1`, 2 and 3 allow to obtain vertically stacked graphs of the variables corresponding to each option, as exemplified by Figures 2.4, 2.5 and 2.6, respectively. These include the measurements and simulation time-series, and allow for the definition of a shaded area, representing a period of interest. The x axis represents the time in hours (UTC), with labels allowing to identify the date. Option `plot_type = 4` calculates and returns the hourly RMSE range (Figure 2.7, blue shaded area, between minimum and maximum) for the sonics of a selected tower.

The same RMSE data and Bias calculations can be obtained and exported in `.csv` format using options `table_rmse` and `rmse_var` (to define the target variable), for the sonic anemometers in the selected towers examples for hourly periods are in Table 2.1.The used formulas for RMSE and Bias are:

$$\text{RMSE} = \sqrt{\left(\frac{1}{N}\sum_{i=1}^{N}(sim_i - exp_i)^2\right)} \tag{2.6}$$

$$\text{Bias} = \frac{1}{N}\sum_{i=1}^{N}(sim_i - exp_i)^2 \tag{2.7}$$

RMSE and Bias are calculated for time periods (1h - Table 2.1, 6h - Table 2.2, 24h - Table 2.3) using the 5-min averaged data (N = number of 5-min periods, i - period index) from Perdigão files (exp) and simulations (sim).

The 6-hour 2.2 and 24-hour 2.3 RMSE and Bias Tables for the last 24h contain error indicators for all sonic anemometers whereas the hourly RMSE Table 2.1 concerns the sonic anemometers in one tower. With that difference the 6h and 24h tables are produced in a separate module (`tables_rmse_bias`) and the hourly tables are produced for every selected tower in the Graph Creation Environment B.3.1.

Period selection is set to 1 hour at Graph Creation Environment and can be set to either 6 or 24 hours in module `tables_rmse_bias` B.3.3.

The processing is handled by functions from the `graph_functions` module, that are

expanded upon in Appendix B.3.2.

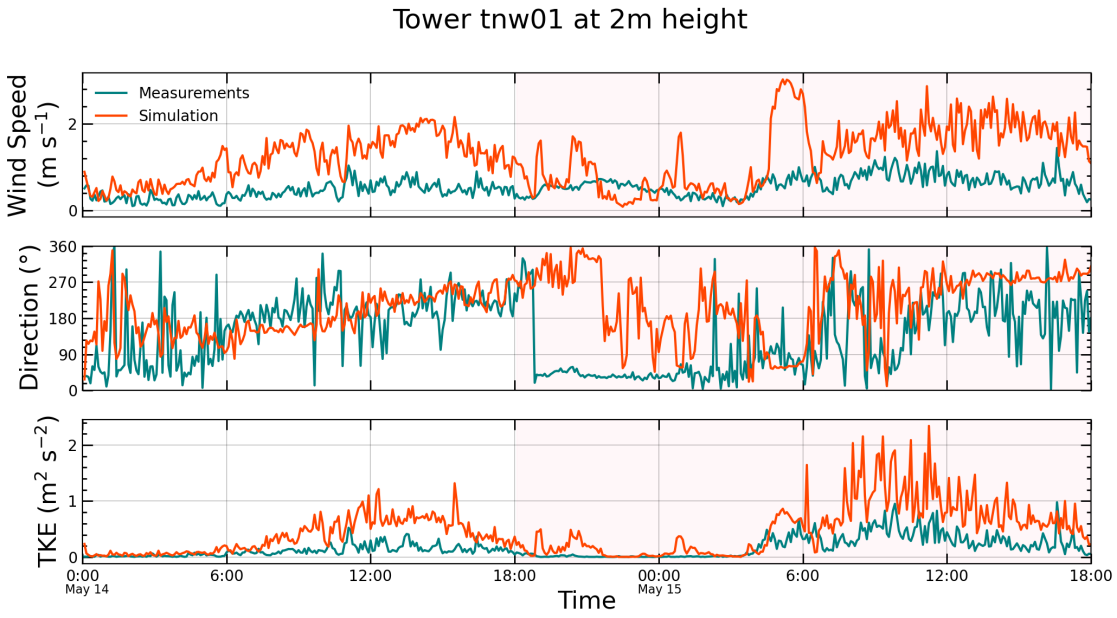

Figure 2.4: Example of time-series comparison graph obtained using `plot_type = 1`: tnw01_2m graph for wind speed, direction and turbulence kinetic energy.

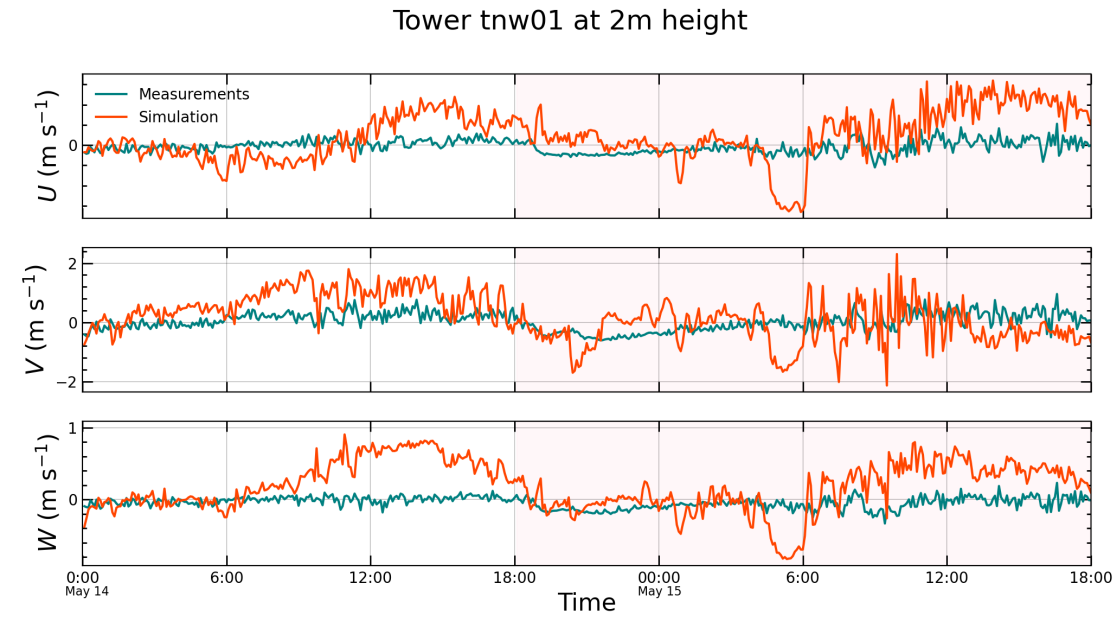

Figure 2.5: Example of time-series comparison graph obtained using `plot_type = 2`: tnw01_2m graph for wind components (u,v,w).

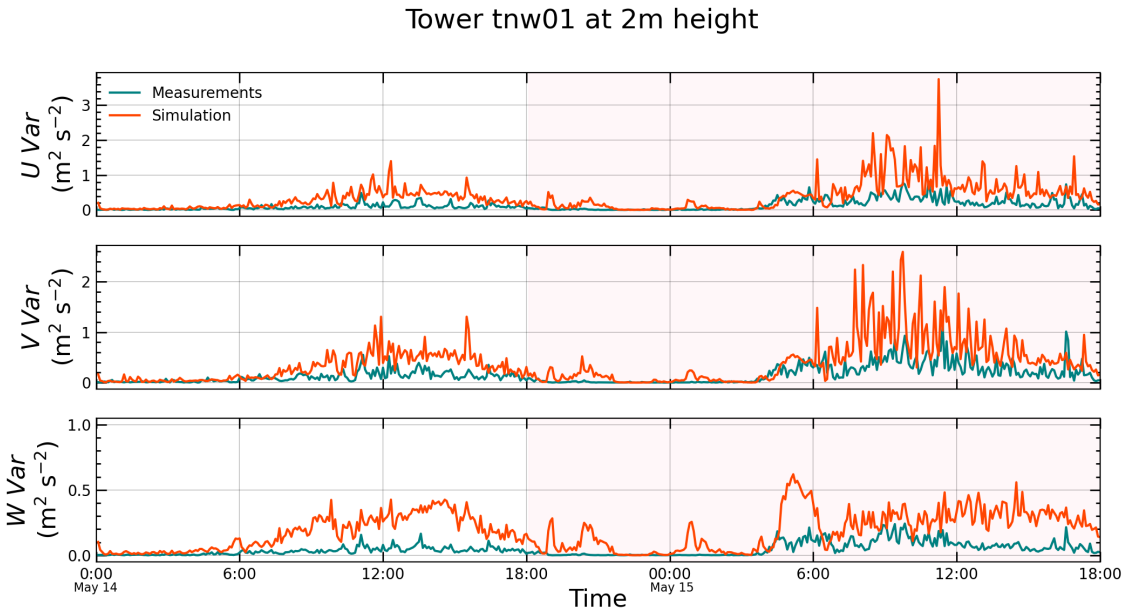

Figure 2.6: Example of time-series comparison graph obtained using `plot_type = 3`: tnw01_2m graph for wind speed variances (uu, vv, ww)

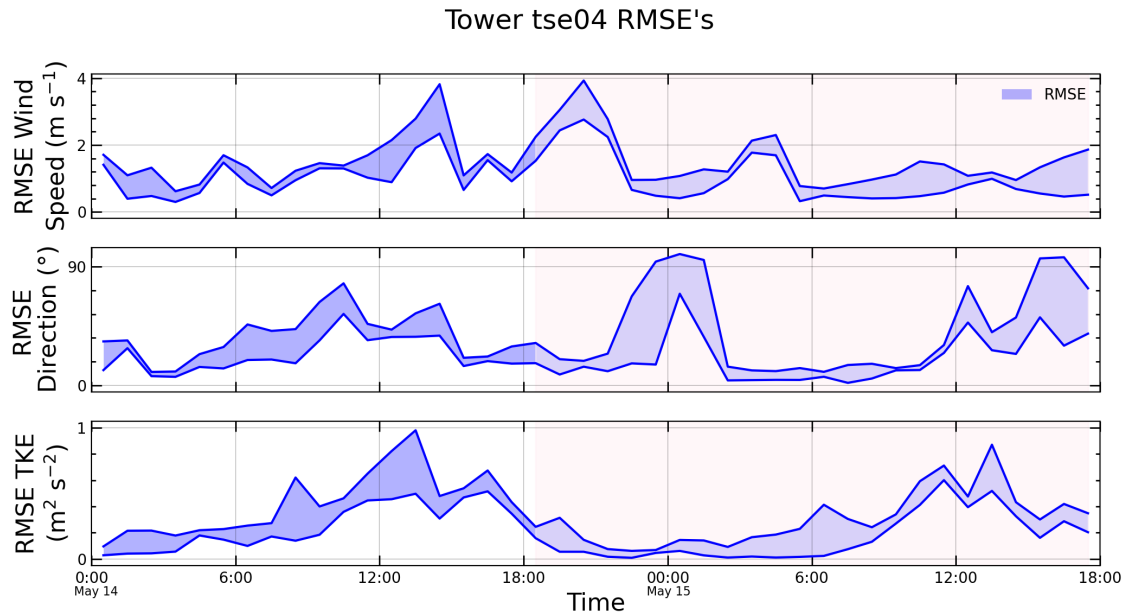

Figure 2.7: Example graph obtained using `plot_type = 4`: tse04 time-series of hourly RMSE of wind speed, direction and turbulence kinetic energy

Table 2.1: Hourly RMSE for Wind Speed ($m\,s^{-1}$), all sonic anemometers for tnw07, for a 24-hour period.

sonic	18	19	20	21	22	23	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
4	1.4741	2.4073	1.6841	0.2847	0.3127	0.7661	0.767	0.6627	0.5921	0.9232	0.9164	1.7662	2.4027	1.4374	0.931	0.9434	0.8484	0.8023	0.7357	1.2743	1.3531	1.5479	1.3453	1.5333
6	1.6947	2.7738	1.8255	0.3129	0.206	0.7694	0.8556	0.5713	0.417	0.8332	0.8452	1.816	2.6551	1.2883	0.8802	0.9872	0.8736	0.801	0.7199	1.3731	1.4053	1.7333	1.5082	1.7305
8	1.7581	2.9473	1.8711	0.325	0.0813	0.6553	0.7311	0.4046	0.3446	0.5623	0.6562	1.7081	2.5894	1.0023	0.7305	0.9534	0.7594	0.8174	0.6671	1.2943	1.3054	1.7762	1.5118	1.762
10	1.7984	3.0947	1.8963	0.4142	0.2468	0.5194	0.5713	0.3093	0.5034	0.2806	0.4713	1.5591	2.4796	0.7526	0.6383	0.9378	0.6658	1.0044	0.7225	1.2792	1.2454	1.8295	1.513	1.7806
12	1.6665	3.2006	1.9303	0.4526	0.3665	0.499	0.4885	0.3288	0.6157	0.1807	0.4201	1.4871	2.4532	0.6504	0.6023	0.811	0.6551	1.0462	0.7866	1.3024	1.2677	1.8676	1.5618	1.8153
20	1.7039	3.2619	1.7874	0.8046	1.0547	0.5677	0.393	0.646	1.1209	0.8473	0.8842	1.1752	2.0538	0.6544	0.7503	0.9177	0.5926	1.2346	0.9789	1.2546	1.2403	1.7491	1.5197	1.7133
30	1.5502	3.4944	1.8453	0.9204	1.2393	0.5863	0.3401	0.8275	1.2495	1.1688	0.6875	0.993	1.5717	0.9766	0.8671	0.789	0.5945	1.2777	0.9899	1.1173	1.2709	1.6366	1.4817	1.5748
40	1.3473	3.7748	2.1249	0.8699	1.1533	0.4265	0.3492	0.7561	1.1131	1.0289	0.6326	0.8221	1.2078	1.1741	0.7066	0.6971	0.6871	1.3709	1.1068	0.9927	1.262	1.5471	1.4897	1.3237
60	1.2014	4.2796	3.0303	1.1446	0.8265	0.5006	0.3549	0.3925	0.6716	0.761	0.7348	0.5925	1.1296	1.362	0.804	0.7861	0.7426	1.4864	1.1895	0.9117	1.3145	1.4606	1.4739	1.0912

Table 2.2: First lines of 6h RMSE and Bias table for Wind Speed ($m\,s^{-1}$), all sonic anemometers, for a 24-hour period

	0-6h		6-12h		12-18h		18-24h	
sonic	RMSE	Bias	RMSE	Bias	RMSE	Bias	RMSE	Bias
tnw01.2m	0.5939	-0.5524	0.8298	-0.6832	0.6002	-0.5081	0.3271	-0.2584
tnw01.10m	0.6838	-0.5671	1.736	-1.2749	1.831	-1.6949	1.1108	-1.0255
tnw01.20m	1.0675	-0.79	2.5006	-1.7422	2.7483	-2.56	1.6055	-1.487
tnw02.2m	0.718	-0.5453	1.3023	-1.2902	0.3783	-0.065	0.4199	0.3386
tnw02.10m	1.0099	-0.874	2.4845	-2.4306	0.585	-0.4906	0.4074	0.0408
tnw02.20m	1.3654	-1.1542	3.0573	-2.9701	1.1574	-1.06	0.724	-0.4943
tnw03.2m	0.3295	-0.2461	2.8806	-2.5705	2.9402	-2.7508	0.8318	-0.774
tnw03.10m	1.6218	-1.5344	4.6899	-4.1826	5.0283	-4.8798	1.847	-1.7113
...	...	...	...	...	...	...	...	...

Table 2.3: First lines of 24h RMSE and Bias table, all sonic anemometers, for a 24-hour period

	Wind speed ($m\,s^{-1}$)		Direction (°)		TKE ($m^2\,s^{-2}$)	
sonic	RMSE	Bias	RMSE	Bias	RMSE	Bias
tnw01.2m	0.9802	0.7062	99.3042	16.5745	0.5006	0.329
tnw01.10m	1.4157	1.0135	90.326	23.2835	0.5891	0.0934
tnw01.20m	1.5014	0.8886	84.9842	8.2622	0.6505	-0.0738
tnw02.2m	1.3094	1.1276	77.7695	8.9289	0.5355	0.3618
tnw02.10m	1.9105	1.5965	75.5996	25.8817	0.427	0.0596
tnw02.20m	1.8758	1.3346	79.5561	26.14	0.5664	-0.2412
tnw03.2m	1.6174	1.3105	43.5371	9.2377	0.5835	0.4611
tnw03.10m	1.7162	0.8583	33.6137	6.0806	0.2621	-0.0203
...	...	...	...	...	...	...

Chapter 3

Results

In this chapter, results obtained from using the different tools are presented. Features and output file examples from the Perdigão Data Retriever are in Section 3.1, together with an estimation of the time savings of this tool when compared to previous procedures. Section 3.2 covers the availability of the sonic anemometer data for the Perdigão-2017 campaign, based on the analysis of the obtained availability map. The Data Comparison Tools features and output files are presented in Section 3.3, the time savings are estimated similarly to Section 3.1. An analysis enabled by the use of this tool is exemplified, and a brief comparison is conducted between measurements and an atmospheric flow simulation over Perdigão.

3.1 Perdigão Data Retriever

Prior to this work, researchers at FEUP working with measurements from the Perdigão campaign mostly limited their analysis to the data from three 100 m towers – `tse04`, `tse09` and `tse13` (Batista, 2019; Palma et al., 2020). The vast amounts of data and the mostly manual and use-case dependent processing would result in large and time consuming efforts to analyse the full dataset. This app was provided with the aim of mitigating or removing those difficulties, providing a more direct access to the data of the other 47 towers. It also removes the overhead required for new researchers to access the data on the netCDF files (familiarisation with the format and libraries).

The main objective of the app was therefore to convert the present data of the complex and raw Perdigão files (.nc) into user friendly and easily accessible files (.csv, .txt, .xls). With small changes to the code, it should be straighforward to make the app export the data to similar file types, if needed.

Following is a summary of the features provided by the Perdigão Data Retriever, as presented in Section 2.3:

1. Automatically obtain sonic anemometer data (5-min averaged) from `https://perdigao.fe.up.pt/`;
2. Filter erroneous data;
3. Obtain field data for specified time periods, mast(s), sonic anemometer(s);

4. Export to easily accessible ASCII .dat or similar files (.txt, .csv, .xls);
5. Calculation of turbulence parameters (TKE, TI, TIH) from extracted data;
6. Time period conversion or re-sampling (from 5-min to 10, 15, 20, 30, or 60-min averaged data);
7. Time sectioning/filtering of the data;
8. Modularity of the app allows that changes to parts of the code are easy to make;
9. Export a file with list of coordinates of processed sonic anemometers (for use in simulation post-processing, to obtain results at the sonic's location).

One of the highlights is simple user interface for the manipulation of the data: the user can easily select the desired averaging period (if different from 5 min), which time period from the campaign to export, and select any configuration or group of sonic anemometers by tower or height, instead of selecting each one manually.

3.1.1 Time savings

To have some tangible information on the amount of time saved by the use of this tool, a simple experiment was undertaken: a python script was written from scratch to download the file (.nc) for May, 14 2017, extract the target variables for the `tnw01.10m` sonic anemometer, and place them into a data frame (pandas DataFrame) similar to the ones that the app provides. The resulting data frame is in the raw state from the `extract_function` module and does not have calculated turbulence parameters. It is important to note that this program written from scratch by the author of this work, who at this point had some knowledge of this specific filetype (.nc) and its libraries, foregoing the need for familiarisation with extraction functions and variable naming conventions adopted for the Perdigão-2017 dataset (see Section 2.1).

The time that the experiment of retrieving data from one file, of one sonic anemometer, without calculating additional turbulence parameters, was of 24 min and 44s. The Perdigão Data Retriever was then executed to retrieve data from sonic anemometers with heights of 2, 10 and 20m from towers tnw01, tnw02 and tnw03, for 0h, May 14 until 18h, May 15, 2017, with calculated turbulence parameters. The operation took 1 min and 21 s.

One must note that the time of writing the script form scratch should be much longer if the programmer does not have knowledge of netCDF format and available libraries. From this simple exercise, and assuming the minimum possible time to write a script from the scratch (24min 44s), Perdigão Data retriever shows a time reduction of at least 18x.

This experience explores the extreme case of retrieving data for one anemometer in one day, another calculation was undertaken to estimate the time consumption in the other extreme case, all anemometers in all days.

In order to obtain files for the full dataset the manual process would use the existing scripts for data retrieval and turbulence calculation and change the files names and/or date for different anemometers on different dates. Assuming that the process would

take 2 min to change the files names and dates for the data retrieval script and another 2 min to change the files names for the turbulence calculation script, it would take 4 min to obtain a file with the retrieved data for each anemometer in each day by manually changing the necessary parameters.

Considering the size of the dataset (175 days $\times$ 185 anemometers), the time necessary to obtain files for the whole dataset can be seen in the following Equation 3.1.

$$4\,\text{min} \times 175\,\text{d} \times 185\ \textit{anemometers} = 129\,500\,\text{min} \approx 2158\,\text{h} \approx 90\,\text{d} \tag{3.1}$$

90 days of uninterrupted user work to obtain the files for the full dataset, which means 3 months of work. Taking into consideration that the user would work only 8h per day, that time would increase to 9 months. And also taking in consideration that the user probably will not work on weekends and holidays, the time required would add up to approximately 1 year, Equation 3.2. In this extreme case the use of the app becomes unavoidable.

$$3\,\text{month}(\textit{uninterrupted}) \rightarrow 9\,\text{month}(8\textit{hperday}) \rightarrow 1\,\text{year} \tag{3.2}$$

3.1.2 Output file example

Figures 3.1 and 3.2 are sectioned examples of the files output by the Perdigão Data Retriever.

```
1494720000.0000 150.0000 -0.2930 -0.4171 -0.0948 0.5134 35.0844 0.0047 0.0096 0.0030 0.0031 0.0016 0.0027 0.0086 0.1464 0.1354
1494720000.0000 450.0000 -0.3414 -0.4691 -0.1076 0.5834 36.0500 0.0034 0.0081 0.0030 0.0015 0.0011 0.0021 0.0073 0.1179 0.1069
1494720000.0000 750.0000 -0.3165 -0.4761 -0.1115 0.5752 33.6088 0.0091 0.0160 0.0038 0.0091 0.0032 0.0049 0.0144 0.1684 0.1600
1494720000.0000 1050.0000 -0.0763 -0.2355 -0.0124 0.2725 17.9472 0.0109 0.0232 0.0040 0.0028 0.0010 0.0034 0.0190 0.4547 0.4305
1494720000.0000 1350.0000 -0.1686 -0.0964 -0.0498 0.2299 60.2479 0.0028 0.0178 0.0013 0.0025 0.0010 0.0029 0.0110 0.4272 0.4274
1494720000.0000 1650.0000 -0.2848 -0.2292 -0.0996 0.3731 51.1772 0.0062 0.0077 0.0020 0.0015 0.0025 0.0013 0.0079 0.1919 0.1860
1494720000.0000 1950.0000 -0.0519 0.0217 -0.0023 0.2387 112.7259 0.0303 0.0378 0.0031 0.0316 0.0066 0.0074 0.0356 2.7354 2.6777
1494720000.0000 2250.0000 -0.1922 -0.1140 -0.0683 0.3298 59.3287 0.0205 0.0534 0.0075 0.0248 0.0094 0.0127 0.0407 0.7049 0.7024
1494720000.0000 2550.0000 -0.3174 -0.2967 -0.1291 0.4415 46.9305 0.0057 0.0097 0.0026 0.0014 0.0019 0.0033 0.0090 0.1713 0.1651
1494720000.0000 2850.0000 -0.2982 -0.3289 -0.1084 0.4661 42.1921 0.0152 0.0233 0.0071 0.0017 0.0038 0.0077 0.0228 0.2698 0.2552
1494720000.0000 3150.0000 -0.1144 -0.1138 -0.0436 0.2104 45.1562 0.0123 0.0225 0.0048 0.0104 0.0054 0.0064 0.0198 0.6875 0.6679
1494720000.0000 3450.0000 -0.0327 -0.1882 -0.0357 0.2214 9.8656 0.0058 0.0178 0.0035 0.0070 0.0021 0.0048 0.0135 0.4887 0.4641
1494720000.0000 3750.0000 -0.1924 -0.2458 -0.0624 0.3237 38.0434 0.0025 0.0112 0.0018 -0.0005 0.0005 0.0020 0.0078 0.2258 0.2164
1494720000.0000 4050.0000 -0.0668 0.0986 0.0102 0.1529 145.8800 0.0065 0.0064 0.0011 0.0049 0.0013 0.0013 0.0070 0.5730 0.5523
```

Figure 3.1: Example of output file in format `.txt`.

	basetime	time	u	v	w	vh	dir	uu	vv	ww	uv	uw	vw	tke	ti	tih
2017-05-14 00:02:30	1,49E+09	150	-0,29299	-0,41713	-0,09479	0,513359	35,08441	0,004749	0,009551	0,002991	0,003109	0,001627	0,002734	0,008646	0,146426	0,135443
2017-05-14 00:07:30	1,49E+09	450	-0,34143	-0,46908	-0,10756	0,583407	36,05001	0,003413	0,008127	0,002992	0,001547	0,001137	0,002061	0,007266	0,117947	0,106898
2017-05-14 00:12:30	1,49E+09	750	-0,31646	-0,47615	-0,11155	0,575237	33,6088	0,009056	0,016042	0,00378	0,009058	0,003203	0,004872	0,014439	0,168432	0,159983
2017-05-14 00:17:30	1,49E+09	1050	-0,07627	-0,23546	-0,0124	0,2725	17,9472	0,010899	0,023168	0,004027	0,002773	0,001026	0,003356	0,019046	0,454709	0,430545
2017-05-14 00:22:30	1,49E+09	1350	-0,16859	-0,09637	-0,0498	0,229905	60,24788	0,00283	0,017839	0,001334	0,002525	0,000966	0,002933	0,011002	0,427194	0,427439
2017-05-14 00:27:30	1,49E+09	1650	-0,28483	-0,2292	-0,09961	0,373059	51,17723	0,006223	0,007652	0,001979	0,001528	0,002453	0,001293	0,007927	0,191854	0,186021
2017-05-14 00:32:30	1,49E+09	1950	-0,0519	0,02174	-0,00229	0,23865	112,7259	0,030347	0,037772	0,003083	0,031568	0,006602	0,007394	0,035601	2,735395	2,677728
2017-05-14 00:37:30	1,49E+09	2250	-0,19216	-0,11396	-0,06832	0,3298	59,3287	0,020499	0,053376	0,007474	0,024762	0,009363	0,012713	0,040674	0,704857	0,702402
2017-05-14 00:42:30	1,49E+09	2550	-0,31743	-0,29673	-0,12913	0,441502	46,93045	0,005746	0,00969	0,002646	0,001413	0,001887	0,003259	0,009041	0,171264	0,165078
2017-05-14 00:47:30	1,49E+09	2850	-0,29817	-0,32892	-0,10839	0,466143	42,19214	0,015191	0,023324	0,00708	0,001669	0,003771	0,007677	0,022798	0,269768	0,255222
2017-05-14 00:52:30	1,49E+09	3150	-0,11438	-0,11376	-0,0436	0,210449	45,15622	0,012324	0,022501	0,004769	0,010439	0,005353	0,006352	0,019797	0,687492	0,667889
2017-05-14 00:57:30	1,49E+09	3450	-0,03273	-0,18823	-0,03571	0,22137	9,865599	0,005769	0,017817	0,003482	0,006988	0,002089	0,004803	0,013534	0,488721	0,464107
2017-05-14 01:02:30	1,49E+09	3750	-0,19236	-0,24582	-0,06244	0,323666	38,04335	0,00245	0,011235	0,001816	-0,00046	0,000451	0,001974	0,007751	0,225817	0,216381
2017-05-14 01:07:30	1,49E+09	4050	-0,06679	0,098568	0,010177	0,152869	145,88	0,006536	0,006436	0,001091	0,004929	0,001341	0,001262	0,007032	0,572956	0,552295
2017-05-14 01:12:30	1,49E+09	4350	0,015379	-0,03651	-0,00958	0,17967	337,1561	0,025209	0,013387	0,003861	0,003678	0,004678	0,002989	0,021229	2,918966	2,863347

Figure 3.2: Example of output file in format `.xls`.

3.2 Availability Map

The Availability Map provides a powerful tool to find if there is data available for any sonic anemometer in any given date in a condensed way and filter the files that have erroneous data. Figure 3.3 represents the Availability Map for all sonic anemometers and the entire measurement period, using the following color code for easy data readability:

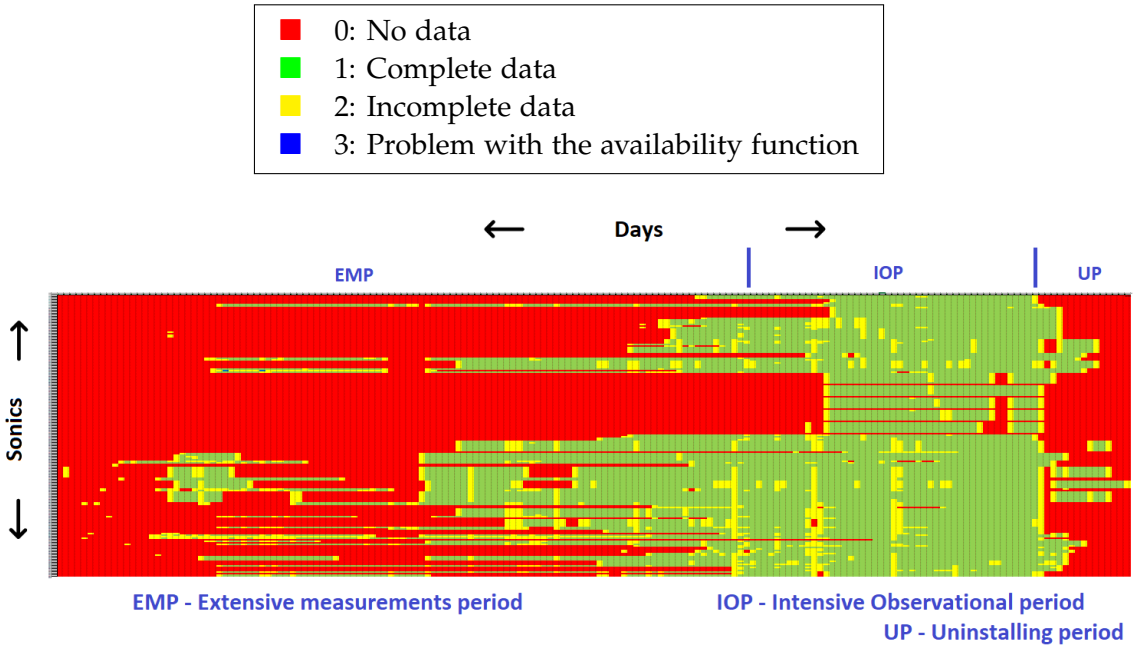

Figure 3.3: Availability map, colored in excel and zoomed to fit

The map can be divided in three main areas:

1. Extensive Measurements Period (EMP), from December 15, 2016 to April 30 2017.
2. Intensive Observational Period (IOP), from May 1 to June 15, 2017.
3. Uninstalling Period (UP), from June 16 to July 1, 2017

The area that has data for almost every sonic anemometer is the Intensive Observational Period (IOP). The area left to the IOP is the Extensive Measurements Period (EMP). The area right to the IOP is the Uninstalling Period.

Some findings are now presented: `rnw07_2m` had no data for the entire campaign, whereas `tnw011_2m` was the sonic anemometer with more days of full data (73,14%). The dates with higher number of sonic anemometers with full data availability were 21, May and from 13 to 15 July 2017. Considering all sonic anemometers, the data availability by period was of 21,72% for the EMP, 80,04% for the IOP and 21,79% for the UP.

3.3 Data Comparison Tools

The Data Comparison Tools provide multiple features:

1. data analysis and comparison between measurements VENTOS®/M simulation results;
2. automated generation of tables with error indicators (RMSE, Bias) for a single, or an ensemble of sonics;
3. automated generation of plots: time-series, multiple variables, RMSE ranges;
4. error indicators for different time periods (1, 6 or 24h, full time period).

For a demonstration of the capabilities of this tool, measurements from 00:00 (UTC), May 14, 2017 to 18:00, May 15 were selected, together with post-processed results of a VENTOS®/M simulation over Perdigão during the same period. The same simulation has been the focus of previous publications (Batista, 2019; Palma et al., 2019), which the reader can consult for additional information.

3.3.1 Time savings

A similar experiment to that of Section 3.1.1 was performed. Before the Data Comparison Tools, graph creation was done manually for each sonic anemometer. The following steps were necessary until a graph could be created:

1. retrieve time series for the required sonics from Perdigão-2017 `.nc` files;
2. execute additional processing over the data from 1 (turbulence parameter calculation, time period filtering and correction - see Section 2.5.1);
3. retrieve a file with the coordinates of the processed sonic anemometers, post-process VENTOS®/M results file to obtain time-series at those locations;
4. execute additional processing on the VENTOS®/M file (turbulence parameter calculation, time period filtering);
5. obtain or use an existing plot script or plotting software and make the required modifications to plot the desired graph;

Steps 1 and 2 concern the functionality of the Perdigão Data Retriever, whereas steps 3, 4 and 5 concern the functionality of the Data Comparison Tools.

The experiment now performed only accounts for the time necessary to perform step 5, meaning that all the data treatment necessary to plot such graph is not accounted for in this experiment. The time that takes to perform steps 3 and 4 should be taken into consideration when evaluating the results of the experiment.

A function built in the Comparison Tools for the calculation of turbulence parameters was used to process data in VENTOS®/M file, so that a graph containing wind speed, direction and turbulence kinetic energy could be created. This was obtained for 1 sonic anemometer, and a similar python script to that used in the Data Comparison Tools, was used here to obtain the plots. The plot formatting settings already defined (defaults) were used. Code was written from scratch to select the data to be plotted from both Perdigão-2017 and VENTOS®/M files. Changes were made in the existing script so that the variables of interest could be plotted.

The "manual" method took 41 min and 4s whereas the Data Comparison Tools app, to perform the same plot now for all sonics in the three 100 m towers, took 34s. Although the comparison is not direct, the time taken to perform step 5 using the Data Comparison tools results in a speedup of at least 72 times. One should note that the time previously required to obtain the plots would be much longer, involving 3 and 4. The former requires the generation of a list of coordinates which have to be manually obtained for each sonic anemometer from the Perdigão-2017 campaign website, whereas the tools developed in this work contain a database with the coordinates of all sonic anemometers (see Section 2.3).

3.3.2 Table analysis example

The tables containing RMSE values for the variables of interest can be used to identify periods and sonic anemometers of interest, i.e., those of maximum and minimum error, The full table can be found in Appendix D. which can be further analysed using time series graphs exported using the same tool.

RMSE and bias for the final 24-hours of the analysed period were obtained for all sonic anemometers. The highest and lowest RMSE values for each variable were signalled and used to identify the graphs to plot (Section 3.3.3).

Some sonic anemometers had missing data, as observed in the data availability map (Figure 3.3). The source of the missing data can be any number of reasons, from data loss in the creation of the files (.nc) to wrongfully logged data or equipment failure. Tables D.1, D.2, D.3 and D.4 show lines with missing values for those reasons.

The analyses of the 24-hour RMSE table can be used to find sonic anemometers of potential interest. Table 3.1, merely a section of the full Table D.1, highlights the sonic anemometers with highest and lowest RMSE for each of the analysed variables.

Sectioning the analysis in 6-hour periods can provide additional information on how errors vary throughout the period of interest, and provide the time of day of highest deviation between the simulation and measurements. Three tables were created concerning the three variables: Wind Speed, Direction, Turbulence Kinetic Energy. The full 6h tables can be found in Appendix D, Tables D.2, D.3 and D.4.

Table 3.2 contains sections of the full 6-hour RMSE and Bias table for Wind Speed, from 18h May 14 to 18h May 15, 2017, for the sonic anemometers presented in Table 3.1 (with the highest and lowest value of 24h RMSE for each variable).

Tables can be automatically created for every tower, concerning the three variables: wind speed, direction, turbulence kinetic energy, with the hourly RMSE variation. An example is in Tables 3.3 and 3.4, concerning the tower where the sonic anemometer with lowest 24h RMSE value for wind speed from 2.3 is located.

The RMSE values for the 2m sonic anemometer have a low fluctuation throughout the entire time period. The RMSE values for the 10m sonic anemometer fluctuate a bit more than the other. A clear trend for growth or reduction of the RMSE's throughout the entire time period was not found. The hourly RMSE's are also plotted in Figure 3.11.

Table 3.1: Sections of 24h RMSE and Bias table, all sonic anemometers, from 18h May 14 to 18h May 15, 2017. Highest (red) and lowest (green) RMSE values signalled for each variable.

	Wind speed ($m\,s^{-1}$)		Direction (°)		TKE ($m^2\,s^{-2}$)	
sonic	RMSE	Bias	RMSE	Bias	RMSE	Bias
tnw01.2m	0.9802	0.7062	99.3042	16.5745	0.5006	0.329
tnw01.10m	1.4157	1.0135	90.326	23.2835	0.5891	0.0934
tnw01.20m	1.5014	0.8886	84.9842	8.2622	0.6505	-0.0738
⋮	⋮	⋮	⋮	⋮	⋮	⋮
tnw09.2m	0.5318	0.1824	94.2611	20.8304	0.3543	0.1755
⋮	⋮	⋮	⋮	⋮	⋮	⋮
tnw13.2m	0.5064	0.2424	82.1445	-18.6724	0.1778	0.1181
⋮	⋮	⋮	⋮	⋮	⋮	⋮
rsw01.20m	1.4113	0.7231	31.4073	5.5773	0.2669	-0.0395
rsw02.10m	2.4016	2.0963	33.0303	-0.9516	0.4725	-0.0747
⋮	⋮	⋮	⋮	⋮	⋮	⋮
v03.2m	0.7499	0.4848	109.5798	18.7027	0.3323	0.2062
⋮	⋮	⋮	⋮	⋮	⋮	⋮

Table 3.2: Sections of 6h RMSE and Bias table for Wind Speed ($m\,s^{-1}$), all sonic anemometers, from 18h May 14 to 18h May 15, 2017, showing the sonic anemometers with highest and lowest 24 RMSE for each variable.

	0-6h		6-12h		12-18h		18-24h	
sonic	RMSE	Bias	RMSE	Bias	RMSE	Bias	RMSE	Bias
tnw01.2m	0.493	0.1295	1.0529	0.6556	0.9286	0.8205	1.2762	1.2192
tnw01.10m	1.0788	0.7084	1.7446	1.2773	0.9398	0.4695	1.7108	1.5987
tnw01.20m	1.2144	0.7345	1.7702	1.4048	1.1505	-0.1612	1.7565	1.5764
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
tnw09.2m	0.5218	0.227	0.6436	-0.0843	0.4188	0.1972	0.5188	0.3895
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
tnw13.2m	0.2844	-0.1816	0.3902	-0.0483	0.6568	0.6353	0.6009	0.5644
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
rsw01.20m	1.8917	1.5892	1.2227	-0.5208	0.7818	0.4919	1.5108	1.3322
rsw02.10m	1.891	1.6373	1.796	1.5155	3.5508	3.4858	1.9133	1.7464
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
v03.2m	0.9744	0.6573	0.4752	0.1619	0.8507	0.63	0.5921	0.49
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Table 3.3: Hourly RMSE for Wind Speed ($m\,s^{-1}$), all sonic anemometers for tnw09, from 18h May 14 to 06h May 15, 2017

sonic	18	19	20	21	22	23	0	1	2	3	4	5
2	0.6321	0.4709	0.318	0.5995	0.3936	0.6301	0.5911	0.697	1.0682	0.5854	0.2261	0.34
10	1.615	2.3582	2.6502	1.1596	1.3708	1.1211	1.1482	1.5947	1.6279	0.614	0.6918	0.6544

Table 3.4: Hourly RMSE for Wind Speed ($m\,s^{-1}$), all sonic anemometers for tnw09, from 06h May 15 to 18h May 15, 2017

sonic	6	7	8	9	10	11	12	13	14	15	16	17
2	0.3654	0.3273	0.3273	0.5624	0.4501	0.4308	0.4594	0.2479	0.5178	0.6285	0.6345	0.526
10	0.3391	0.8678	0.7264	0.7297	0.6136	0.6257	0.6779	0.5952	0.9353	1.4492	1.5237	1.2112

3.3.3 Graph analysis example

Time series plots are generated for chosen sets of variables, allowing for qualitative comparisons between experimental and simulation data. All graphs have an option to highlight a period of higher importance, in this case the last 24 hours of data, which were also the focus of other publications Batista (2019); Palma et al. (2019).

From the 24 h RMSE Table 3.1, the two sonic anemometers with the highest and lowest RMSE for wind speed where chosen to demonstrate the plotting capabilities of the tool. Since it is not the focus of this work, a merely superficial discussion is presented on the observations for each graph (Figures 3.4 and 3.5) to exemplify the kind of analyses that can be performed.

Wind speed, direction, turbulence parameter analyses

This graph type plots in a time series, the values for Wind Speed, Direction, Turbulence Parameter, from both experimental and simulation data. The third parameter can be selected by the users, with the available options being: turbulence kinetic energy (`tke`), turbulence intensity (`ti`) or turbulence intensity from horizontal components (`tih`).

Figures 3.4 and 3.5 represent the time series comparison of wind speed, direction and TKE for the aforementioned sonic anemometers of interest. In Figure 3.4, for `rsw02` at 10 m (anemometer with the highest 24 h RMSE for wind speed), the computational results are higher than the measurements, which could be caused by the use of an inappropriate (low) surface roughness in the computational model. Around 00:00 May 15, direction for measurements data shifts very quickly, very low values of wind speed mean chaotic flow, which could be the explanation.

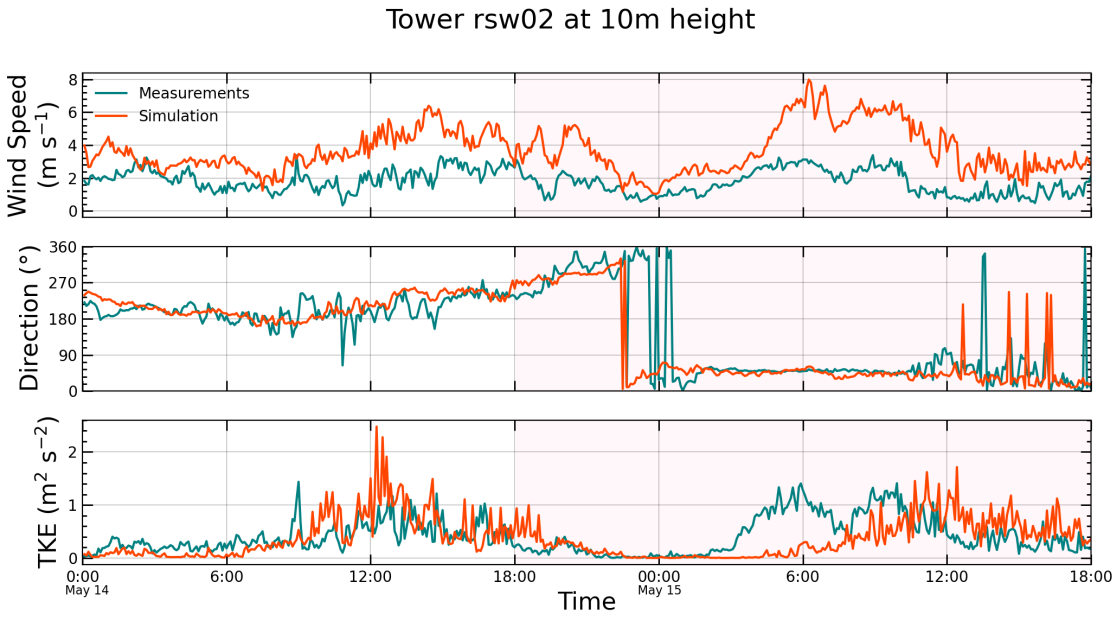

Figure 3.4: rsw02.10m sonic anemometer graph for Wind Speed, Direction, Turbulence Kinetic Energy

Figure 3.5 represents the 2 m sonic anemometer at `tnw09` with the lowest 24 h RMSE for wind speed, i.e., a good approximation between computational results and measurements. This is in constrast with the observations at a slightly higher elevation in `rsw02`, indicating that roughness might not be the only factor, but a deeper analysis considering the location and additional topographic and flow features is necessary. Direction varies largely for the simulation results and very low values of wind speed could be the cause.

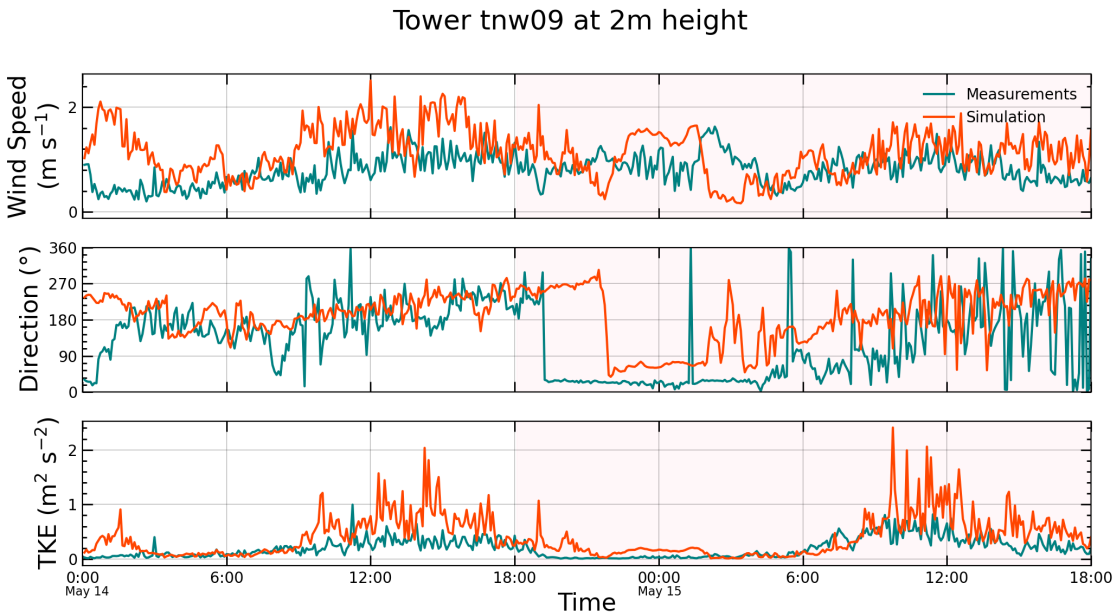

Figure 3.5: tnw09.2m sonic anemometer graph for wind speed, direction, turbulence kinetic energy

Wind speed components

This graph type plots in a time series the values for the wind components (u, v, w) from both experimental and simulations data, using a similar layout to the one in Section 3.3.3.

Two examples are presented in Figures 3.6 and 3.7.

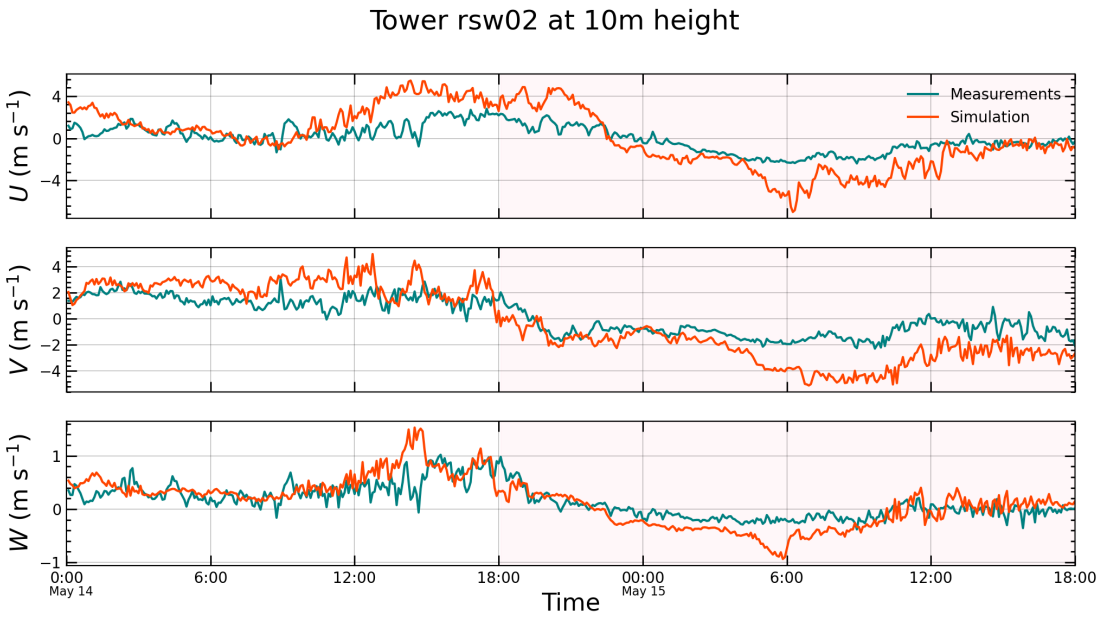

Figure 3.6: rsw02.10m sonic anemometer graph for wind components (u,v,w)

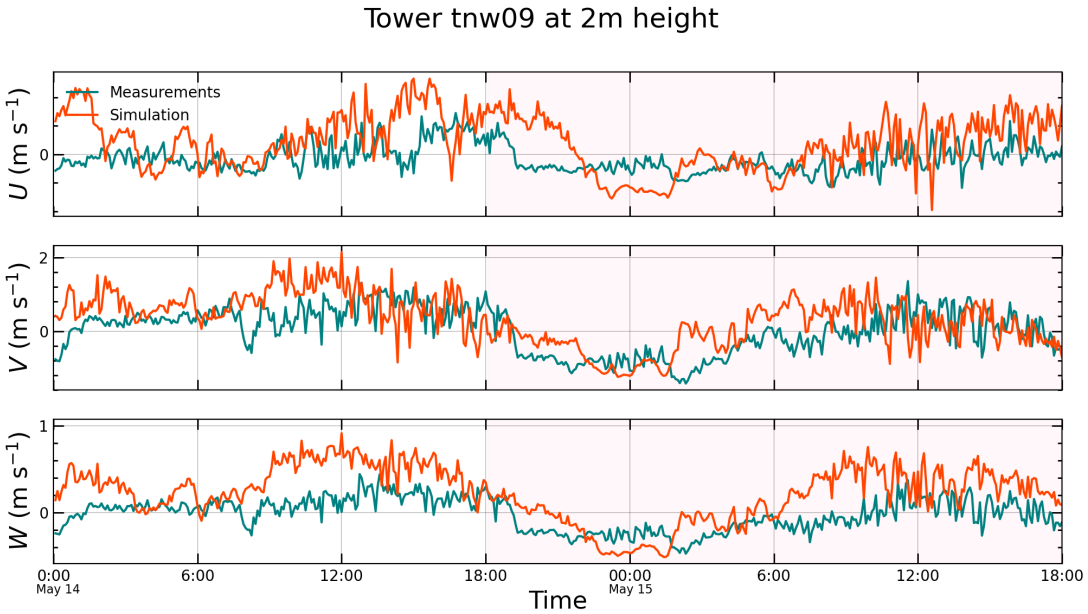

Figure 3.7: tnw09.2m sonic anemometer graph for wind components (u,v,w)

Wind variances

This graph type also uses a similar layout to the one in Section 3.3.3, now considering the wind velocity variances ($u'u'$, $v'v'$, $w'w'$) from both experimental and simulations data. Two examples are in Figures 3.8 and 3.9:

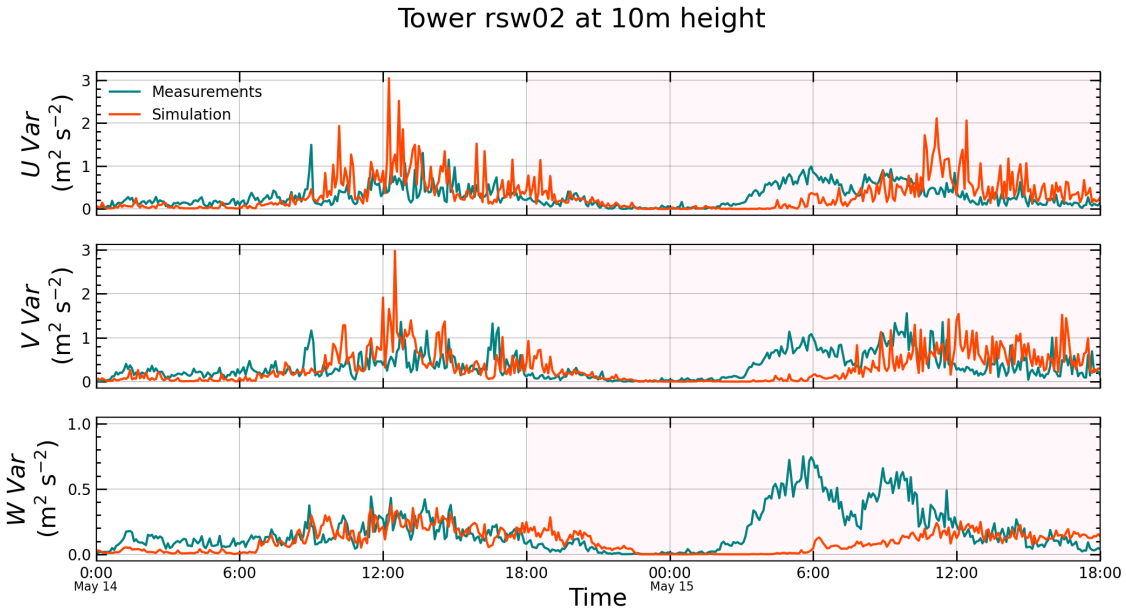

Figure 3.8: rsw02.10m sonic anemometer graph for wind variances (uu,vv,ww)

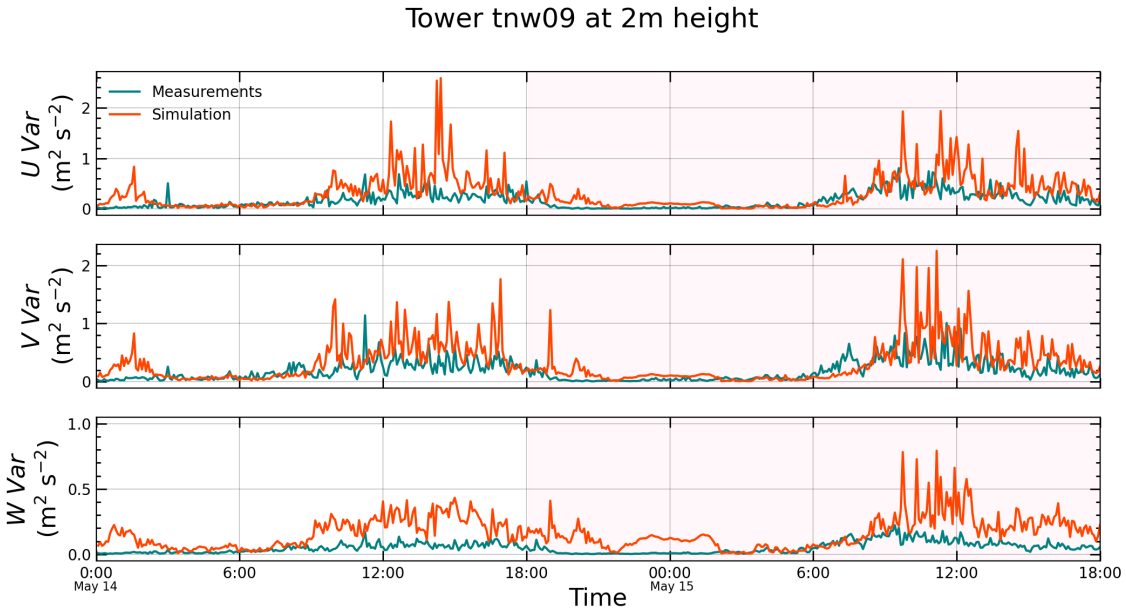

Figure 3.9: tnw09.2m sonic anemometer graph for wind variances (uu,vv,ww)

Hourly RMSE

The graph type for RMSE variation concerns a set of sonic anemometers (e.g., all in a specified tower) for a selected variable. The values are calculated hourly and it can provide useful information on how the error varies over, for example, a full day. The two lines of the graphs 3.10 and 3.11 represent the maximum and minimum RMSE values, considering, in this case, every sonic anemometer in the designated tower.

Two towers were selected, and the graphs obtained considering all anemometers in each tower. Figures 3.10 and 3.11 represent the hourly RMSE variation for towers `rsw02` and `tnw09`.

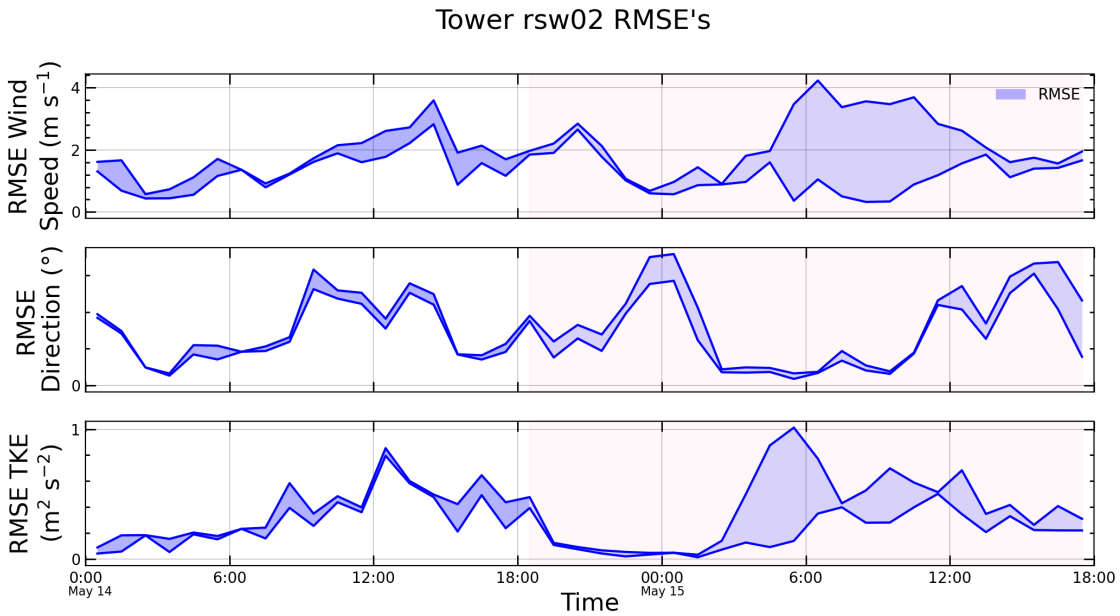

Figure 3.10: rsw02 tower graph for hourly RMSE variation for wind speed, direction, turbulence kinetic energy, for every sonic anemometer in tower

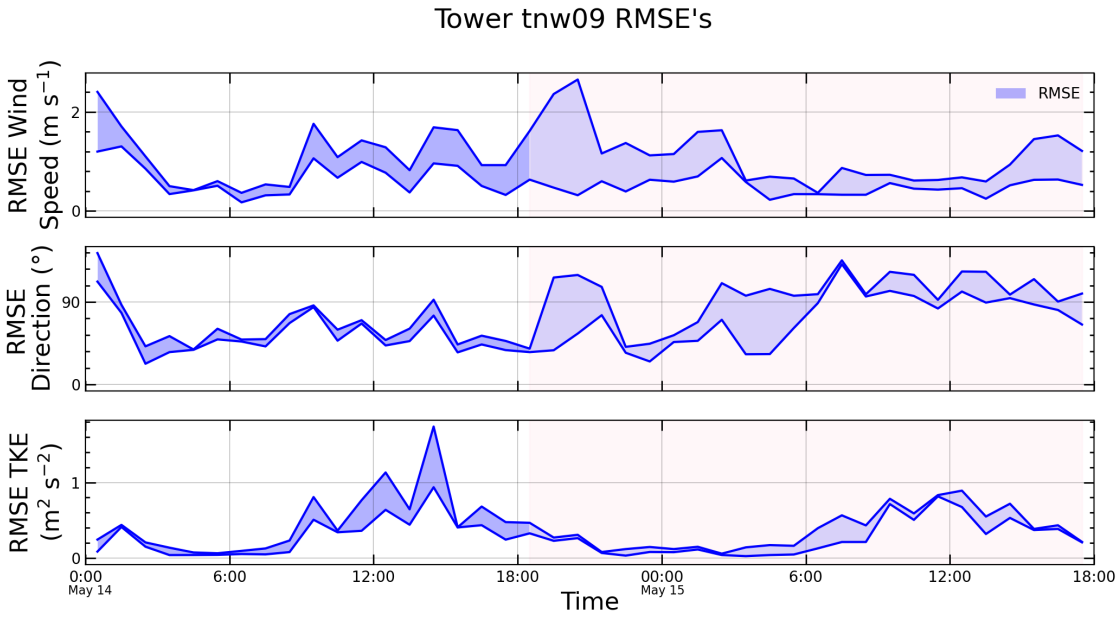

Figure 3.11: tnw09 tower graph for hourly RMSE variation for wind speed, direction, turbulence kinetic energy, for every sonic anemometer in tower

Chapter 4

Conclusions

The Perdigão Data Retriever provides direct and quick access to Perdigão-2017 sonic anemometer measurements, allowing to filter, average, calculate additional parameters (turbulence kinetic energy and turbulence intensity), and export to easily accessible file formats (e.g. `.csv`). Using the app instead of the previous (manual) process resulted in a speedup of at least 18 times (a process that could take more than one hour is now performed in a few minutes). Still, the time reduction can be exponentially higher as the size of the dataset to be retrieved increases.

An availability map was obtained, providing the means to visualize the Perdigão-2017 data set availability quickly and practically. Considering all sonic anemometers, data was available for 21,72 % of the extensive measurement and 80,04% of the intensive measurement periods.

Comparison Tools were developed to provide users with an expedited way of comparing data from the sonic anemometer measurements obtained using the Perdigão Data Retriever against simulations results (graphs and tables). Compared to the previous (manual) process, the use of the tools provides a minimum speedup of 72 times (i.e., a process that could take more than 40 min can now be performed in less than a minute).

4.1 Future Works

- The measurement files that are explored in this work concern 5-min averaged data; it would be useful to adapt the Perdigão Data Retriever to the 20 Hz data set. This would also provide the ability for temporal averaging to any target period and perform improved calculations of variances and turbulence parameters for periods other than 5-min.
- The use of an external configuration file could be implemented as an alternative to the existing interactive user interface. This would allow for additional automation capabilities when defining all input parameters (e.g., period of interest, sonic anemometers to process, etc.).
- The sonic anemometer coordinates file provided by the app is exported using coordinates in the original system: ETRS89/PT-TM06. Options could be implemented

to allow the user to specify a target coordinate system, expanding integration of the developed toolset with the existing VENTOS®/M, or other codes' toolset.

- The filters used to determine the validity of the data were not very restrictive. A review of those filters, and the addition of, for example, the detection of repeated data values in consecutive periods could be applied when obtaining the availability map.
- The Comparison Tools enable the comparison framework for use with VENTOS®/M simulations over Perdigão measurements. For future works, however, the tools can be expanded to provide compatibility with results from other codes (e.g. WRF, VENTOS®/2).
- Further development of features to the comparison tools (different types of data presentation like scatter plots, etc.; comparing measurements with results from multiple simulations) could be valuable future work.
- An option to export files in netCDF format that could gather multiple sonic anemometers data in one file or other data specifications could be of use to future data post-processing.
- An option in the comparison tools to plot in time series different anemometers alongside could be valuable for further studies and understanding of the flow in the Perdigão site.

Bibliography

Anaconda. Anaconda | The World's Most Popular Data Science Platform. URL https://www.anaconda.com/.

V. T. P. Batista. Flow patterns over Serra do Perdigão: analysis of a 24 hour period from May 14 at 18:00 UTC. Master's thesis, Faculdade de Engenharia da Universidade do Porto, July 2019. URL https://hdl.handle.net/10216/122289.

J. Carrapa. Perdigao_python project github repository, September 2022. URL https://github.com/j-carrapa/Perdigao_python. original-date: 2022-05-11T16:56:09Z.

F.A. Castro, J.M.L.M. Palma, and A. Silva Lopes. Simulation of the Askervein flow. Part 1: Reynolds Averaged Navier-Stokes equations ($k-\varepsilon$ turbulence model). *Boundary-Layer Meteorology*, 107(3):501–530, June 2003. ISSN 0006-8314, 1573-1472. doi: 10.1023/A:1022818327584. URL http://link.springer.com/article/10.1023/A%3A1022818327584.

UCAR Catalog EOL. Perdigao Data Access, 2022. URL https://data.eol.ucar.edu/master_lists/generated/perdigao/.

H.J.S. Fernando, J. Mann, J.M.L.M. Palma, J.K. Lundquist, R.J. Barthelmie, M. Belo-Pereira, W.O.J. Brown, F. K. Chow, T. Gerz, C.M. Hocut, P.M. Klein, L.S. Leo, J.C. Matos, S.P. Oncley, S.C. Pryor, L. Bariteau, T.M. Bell, N. Bodini, M.B. Carney, M.S. Courtney, E.D. Creegan, R. Dimitrova, S. Gomes, M. Hagen, J.O. Hyde, S. Kigle, R. Krishnamurthy, J.C. Lopes, L. Mazzaro, J.M.T. Neher, R. Menke, P. Murphy, L. Oswald, S. Otarola-Bustos, A.K. Pattantyus, C. Veiga Rodrigues, A. Schady, N. Sirin, S. Spuler, E. Svensson, J. Tomaszewski, D.D. Turner, L. van Veen, N. Vasiljević, D. Vassallo, S. Voss, N. Wildmann, and Y. Wang. The Perdigão: Peering into microscale details of mountain winds. *Bulletin of the American Meteorological Society*, 100(5):799–819, May 2019. ISSN 0003-0007, 1520-0477. doi: 10.1175/BAMS-D-17-0227.1. URL http://journals.ametsoc.org/doi/10.1175/BAMS-D-17-0227.1.

D. F. Gomes, J. C. Lopes, José L. Palma, G. Lopes, I. Fernandes, R. Abreu, and J. Fernandes. Perdigão Field Experiment, 2020. URL https://perdigao.fe.up.pt/.

J. Mann, N. Angelou, J. Arnqvist, D. Callies, E. Cantero, R. Chávez Arroyo, M. Courtney, J. Cuxart, E. Dellwik, J. Gottschall, S. Ivanell, P. Kuhn, G. Lea, J. Matos, C. Rodrigues, J. Palma, L. Pauscher, A. Peña, J. Rodrigo, S. Söderberg, and N. Vasiljević. Complex terrain experiments in the new european wind atlas. *Philosophical Transactions of the Royal Society A: Mathematical Physical and Engineering Sciences*,

375:(2091):20160101, April 2017. doi: 10.1098/rsta.2016.0101. URL http://rsta.royalsocietypublishing.org/content/375/2091/20160101.

Matplotlib. Matplotlib — Visualization with Python, 2022. URL https://matplotlib.org/.

NCAR-EOL. Perdigão-ISFS Data Report | Earth Observing Laboratory, 2022. URL https://www.eol.ucar.edu/content/perdig%C3%A3o-isfs-data-report.

NCAR-EOL. ISFS Variable Names | Earth Observing Laboratory, 2022. URL https://www.eol.ucar.edu/content/isfs-variable-names.

Numpy. NumPy, 2022. URL https://numpy.org/.

J. M. L. M. Palma, C. A. M. Silva, V. C. Gomes, A. Silva Lopes, T. Simões, P. Costa, and V. T. P. Batista. The digital terrain model in the computational modelling of the flow over the Perdigão site: the appropriate grid size. *Wind Energy Science*, 5(4):1469–1485, November 2020. ISSN 2366-7443. doi: https://doi.org/10.5194/wes-5-1469-2020. URL https://wes.copernicus.org/articles/5/1469/2020/. Publisher: Copernicus GmbH.

J.M.L.M. Palma, A. Silva Lopes, V.M. Costa Gomes, C. Veiga Rodrigues, R. Menke, N. Vasiljević, and J. Mann. Unravelling the wind flow over highly complex regions through computational modeling and two-dimensional lidar scanning. *Journal of Physics: Conference Series*, 1222:012006, May 2019. ISSN 1742-6588, 1742-6596. doi: 10.1088/1742-6596/1222/1/012006. URL https://iopscience.iop.org/article/10.1088/1742-6596/1222/1/012006.

Pandas. Pandas - Python Data Analysis Library, 2022. URL https://pandas.pydata.org/.

Feup website Perdigão Layout. Perdigão 2017: experiment Layout, 2022. URL https://perdigao.fe.up.pt/api/documents/versions/307?download.

C. Veiga Rodrigues, J. M. L. M. Palma, and Á. H. Rodrigues. Atmospheric Flow over a Mountainous Region by a One-Way Coupled Approach Based on Reynolds-Averaged Turbulence Modelling. *Boundary-Layer Meteorology*, 159(2):407–437, May 2016. ISSN 0006-8314, 1573-1472. doi: 10.1007/s10546-015-0116-7. URL http://link.springer.com/10.1007/s10546-015-0116-7.

UCAR. Unidata | NetCDF, 2022. URL https://www.unidata.ucar.edu/software/netcdf/.

J. Whitaker. netCDF4: Provides an object-oriented python interface to the netCDF version 4 library. URL http://github.com/Unidata/netcdf4-python.

Appendices

Appendix A

Variables Used in Code List

List of variables used in code.

- Perdigão Data Retriever A.1
- Availability Map A.2
- Data Comparison Tools A.3

A.1 Perdigão Data Retriever

User defined variables:

Table A.1: User defined variables

Variable	Description
`dates_def`	Dates defined by the user – an array containing in sequence all the days dates that the user as chosen to extract data from
`z1, z2`	Start and end date – those can be different dates or the same, if the user only wants data from one day
`z3, z4`	Start and end hour – the user can specify a start hour and an end hour for each day or a start hour for the first day and an end hour for the last day
`k`	Time period conversion factor – if the user wants to retrieve the data in a time period different than 5min, a conversion will be made using this factor
`sm`	Time sectioning flag variable: the user might choose an alternative to having the full day data, it can be specific parts of every day or, a start hour for the first day and an end hour for the last day, this flag variable will be used to process that request
`hs`	Height selection mode flag variable - the user specifies how the height selection process will occur, same height(s) for all selected towers or select individually sonic height(s) for each selected tower
`x_arr`	Array with selected towers name – to extract the data from any sonic anemometer, a tower name is necessary, this array provides that name
`y_arr`	Array with selected sonic height – to extract the data from any sonic anemometer, a sonic height is necessary, this array provides that name
`sdf`	2D array with selected sonic height for each selected tower – to extract the data from any sonic anemometer, a sonic height and tower name is necessary, this array provides those two variables, used in mode 1 present in B.1.2

Data variables extracted:

Table A.2: Data variables extracted

Variable	Description
`basetime`	The base time variable contains one value, the time of the start of the file, as a number of POSIX (non-leap) seconds since 1970 Jan 1, 00:00 UTC
`time`	Time variable with a time dimension whose values represent a number of seconds since the base time, which provides a human-readable representation of the time in the file
`u, v, w`	These three variables represent the wind components in geographic coordinate system, they're units are $\mathrm{m\,s^{-1}}$
`vh`	Wind speed variable $\mathrm{m\,s^{-1}}$
`dir`	Wind direction in geographic coordinate system, units in $^\circ$
`uu, vv, ww`	These three variables represent the wind components variances in geographic coordinate system, they're units are $\mathrm{m^2\,s^{-2}}$
`uv, uw, vw`	These three variables represent the wind components covariances in geographic coordinate system, they're units are $\mathrm{m^2\,s^{-2}}$

Turbulence calculated data variables:

Table A.3: Turbulence calculated data variables

Variable	Description
`tke`	Turbulence kinetic energy, units in $\mathrm{m^2\,s^{-2}}$
`ti`	Turbulence intensity
`tih`	Turbulence intensity from horizontal components

Data manipulation variables in processing:

Table A.4: Data manipulation variables

Variable	Description
`dfc`	Data frame - 2D array that contains all the data for the target extracted variables as columns and row indexed by time. This variable will change throughout the processing, being updated at each step until reaches the export phase
`tke_arr`	Turbulence kinetic energy time adjusted array - array that stores the calculated turbulence parameter for a different than 5-min period
`ti_arr`	Turbulence intensity time adjusted array - array that stores the calculated turbulence parameter for a different than 5-min period
`tih_arr`	Turbulence intensity from horizontal components time adjusted array - array that stores the calculated turbulence parameter for a different than 5-min period

Coordinates file variables:

Table A.5: Coordinates variables

Variable	Description
`coord_index_array`	Coordinates index array - an array that contains the coordinates of each retrieved sonic in Easthing, Northing, height
`df_coord`	All sonic anemometers coordinates data frame - a data frame that contains the coordinates of all sonic anemometers in Easthing, Northing, height
`df_sonics`	Selected sonic anemometers coordinates data frame - a data frame that contains the coordinates of all selected sonic anemometers in Easthing, Northing, height, that will be exported
`direc`	Directory file path - a file path to the folder where the selected sonic anemometers coordinates data frame is to be stored

Supporting operations variables:

Table A.6: Supporting operations variables

Variable	Description
`x`	Tower name - variable used as argument when calling processing functions that contains tower name code
`y`	Sonic height - variable used as argument when calling processing functions that contains sonic height
`height, m, i, j`	Sonic anemometer supporting variables - variables used to loop through the selected sonic anemometers in mode 1 B.1.2
`hei, i, j`	Sonic anemometer supporting variables - variables used to loop through the selected sonic anemometers in mode 2 B.1.3
`tow, i`	Sonic anemometer supporting variables - variables used to loop through the selected sonic anemometers in mode 3 B.1.4
`g1 , g2`	Loop internal functioning variables in main B.1.1
`f_ans ,g_ans`	Input variables in main B.1.1
...	Local variables for small functions are not mentioned

A.2 Availability Map

Variables List:

Table A.7: Availability map variables

Variable	Description
`dates_tot`	All dates array – an array containing in sequence all the dates in the Perdigão campaign
`dates_def`	Dates defined by the user – an array containing sequencial dates that match one time interval of the `dates_tot` array
`df`	Data frame - 2D array that contains all the data for the availability map created so far, used for store data for every step of the iterative process

A.3 Data Comparison Tools

A.3.1 Graph creation environment

User defined variables:

Table A.8: Graph defined variables

Variable	Description
`plot_type`	Type of plot - flag variable to determine which type of graph is to be made (e.g. `'vh, dir, tke', 'u,v,w', etc.`)
`table_rmse`	Hourly RMSE table - flag variable to determine if an hourly RMSE table is to be made or not
`rmse_var`	RMSE variable - flag variable to determine which variable will the hourly RMSE table concern
`towers`	Towers array - array containing the names of the selected towers for graph plotting and table construction
`highlight`	Highlight flag variable - to determine if the last 24h will appear highlighted in the graphs or not

Supporting operations variables:

Table A.9: Graph supporting variables

Variable	Description
`tower_p`	Tower name - variable used as argument to retrieve data from the correct files
`height`	Sonic height - variable used as argument to retrieve data from the correct files
`heights`	Heights available for tower array - array containing heights available for tower specified by `j`
`j`	Tower index position - used in `tl.sonics_available_name(j)`
`hour_period`	Hour period = 1h
`direc_pf`	Directory path to folder containing Perdigão retrieved files
`path_ventos_sonics_coord`	Path to file containing all conic anemometers coordinates
`direc_sim`	Directory path to folder containing VENTOS®/M simulation files
`a_name, b_name, c_name`	Names of the variables for each graph type
`nvars`	Array containing all selected variable names
`sta, he, va`	RMSE data supporting variables
`df, pf_2d_array, pf_num_rows`	Arrays containing data from the Perdigão files
`df_ventos, data`	Arrays containing data from the VENTOS®/M simulation files
`v_name`	Variable name from `nvars`
`nhour, var, h`	Variables used for constructing error indicator arrays
`err, mse, rmse, bias, r, ss`	Error indicator arrays
`mean_e, mean_s`	Mean values arrays
`rmse_min, rmse_max`	Array containing min and max hourly RMSE for selected tower
`rm`	Variable used for building of `rmse_min, rmse_max` arrays
`df_rmse`	Array containing calculated RMSE data

A.3.2 Tables RMSE and Bias

User defined variables:

Table A.10: Tables defined variables

Variable	Description
`towers`	Towers array - array containing the names of the selected towers for table construction
`period`	Selected period (6 or 24h)

Supporting operations variables:

Table A.11: Tables supporting variables

Variable	Description
`tower_p`	Tower name - variable used as argument to retrieve data from the correct files
`height`	Sonic height - variable used as argument to retrieve data from the correct files
`heights`	Heights available for tower array - array containing heights available for tower specified by `j`
`j`	Tower index position - used in `tl.sonics_available_name(j)`
`df_24`	Array containing data for the 24h RMSE and BIAS
`df_6_vh`	Array containing data for the 6h RMSE and BIAS of Wind Speed
`df_6_dir`	Array containing data for the 6h RMSE and BIAS of Direction
`df_6_tke`	Array containing data for the 6h RMSE and BIAS of Turbulence Kinetic Energy
`direc_pf`	Directory path to folder containing Perdigão retrieved files
`path_ventos_sonics_coord`	Path to file containing all conic anemometers coordinates
`direc_sim`	Directory path to folder containing VENTOS®/M simulation files
`a_name, b_name, c_name`	Names of the variables to be used
`nvars`	Array containing all selected variable names
`sta, he, va`	RMSE data supporting variables
`df, pf_2d_array, pf_num_rows`	Arrays containing data from the Perdigão files
`df_ventos, data`	Arrays containing data from the VENTOS®/M simulation files
`v_name`	Variable name from `nvars`
`nh, var, h`	Variables used for constructing error indicator arrays
`rmse_s, bias_s, rmse_24, bias_24, r, ss`	Error indicator arrays
`mean_e, mean_s`	Mean values arrays

Appendix B

Full Modules Code and Description

Full Modules Code

- Perdigão Data Retriever B.1
- Availability Map B.2
- Data Comparison Tools B.3

B.1 Perdigão Data Retriever

B.1.1 main_perdigao_app

Table B.1: Main module variables

Variable	Description
`g1 ,g2`	Loop internal functioning variables
`f_ans`	Input variable - mode selection
`g_ans`	Input variable - restart option

This is the main module of the app and where the app starts.

When this module is executed, it starts the interface with the user and queries which mode of data selection is to be chosen.

When any of the modes is completed, the app returns to this module and asks the user if the program should restart for a new process of data retrieval or if it is the end (Yes/No).

Module code:

```
# Main - Top level module, user interface to run sub-modules

# Allows to choose one mode of data gathering, process and export and repeat the
    operation

g1 = 0
```

```
6
7  while g1 != 1:
8
9      g2 = 0
10
11     f_ans = input("Choose one option:\n[1] Multiple sonics, multiple masts\n[2]
           All sonics, multiple masts\n[3] All masts, multiple sonic
           heights\nPlease choose the number of the desired mode:")
12     f_ans = int(f_ans)
13
14     if f_ans == 1:
15         import mode_01_multiple_sonics_multiple_masts
16
17     if f_ans == 2:
18         import mode_02_all_sonics_multiple_masts
19
20     if f_ans == 3:
21         import mode_03_all_masts_multiple_sonics
22
23     while g2 != 1:
24         g_ans = input("End of data gathering?\n[yes/no]:")
25
26         if g_ans == 'no':
27             g2 = 1
28             continue
29         if g_ans == 'yes':
30             g2 = 1
31             g1 = 1
32             continue
33         else:
34             print("Type correct answer.\n[yes/no]:")
35             continue
36     continue
```

B.1.2 mode_01_multiple_sonics_multiple_masts

Table B.2: Mode 1 defined variables

Variable	Description
`dates_def`	Dates defined by the user – an array containing in sequence all the days dates that the user as chosen to extract data from
`z1, z2`	Start and end date – those can be different dates or the same, if the user only wants data from one day
`z3, z4`	Start and end hour – the user can specify a start hour and an end hour for each day or a start hour for the first day and an end hour for the last day
`k`	Time period conversion factor – if the user wants to retrieve the data in a time period different than 5min, a conversion will be made using this factor
`sm`	Time sectioning flag variable: the user might choose an alternative to having the full day data, it can be specific parts of every day or, a start hour for the first day and an end hour for the last day, this flag variable will be used to process that request
`hs`	Height selection mode flag variable - the user specifies how the height selection process will occur, same height(s) for all selected towers or select individually sonic height(s) for each selected tower
`x_arr`	Array with selected towers name – to extract the data from any sonic anemometer, a tower name is necessary, this array provides that name
`j_arr`	Array with selected towers name index position – used in supporting operations in mode 1
`y_arr`	Array with selected sonic height – to extract the data from any sonic anemometer, a sonic height is necessary, this array provides that name
`sdf`	2D array with selected sonic height for each selected tower – to extract the data from any sonic anemometer, a sonic height and tower name is necessary, this array provides those two variables, used just in mode 1
`i, j`	Sonic anemometer supporting variables - variables used to loop through the selected sonic anemometers in mode 1
`m`	Sonic anemometer availability - array containing sonic anemometer availabilty for each tower
`height`	Heights array
`x, y`	sonic anemometer variables used to call `processing_functions` module

In `mode_01_multiple_sonics_multiple_masts`, as in the other modes, the user starts to select the time variables for data retrieval. Then, any number of masts can be selected, and from those masts the user can select which sonic anemometers will be chosen for data retrieval.

The code is divided in "Parts" that will be explained bellow:

- Part 1: The user selects the time period of the sample, that can be the standard

5-min data, or 10, 20, 30 or 60 min. `k` (conversion factor) is stored as a variable that will be used for time conversion of data if needed.

- Part 2: The user selects a start and end date to retrieve data from. Three variables are stored: `dates_def` (array containing all the dates defined, can have multiple or only one date), `z1` (start date), `z2` (end date). The `dates_def` array will have sequential dates from the start date until the end date.
- Part 3: The user might choose if the data is to be delivered for all 24h per day, just for a time interval in each day (e.g. from 12:00 to 18:00 every day) or continuous data from a selected hour in the first day until a selected hour in the last day (which may be the same as the first day). This information is stored in `sm` (time sectioning flag variable), and if the option chosen was not the full 24h per day, `z3` (start hour variable) and `z4` (end hour variable) will be stored as well.
- Part 4: the NetCDF files concerning the dates defined by the user are downloaded.

  These first four parts are common to all the modes. The differentiation of the modes appears in the next parts to be presented. As explained before, in terms of data retrieval, each sonic anemometer is identified by a pair of variables: tower name and sonic height.
- Part 5: The user types any number of tower names where the target sonic anemometers are located. That information is stored in a pair of variables, `x_arr` (array containing the names of the selected towers) and `j_arr` (array containing the matching index position of the selected towers name). Afterwards, the user might choose how to select the sonic heights to conclude the process of sonic anemometer selection. The two available options are: select the same heights for every chosen towers or loop through towers to learn sonic anemometer heights for each one. This information is stored in `hs` (height selection flag variable).
- Parts 6-12: Using the defined variables in the first 5 parts, the app will loop through the sonic anemometer variables to call the `processing_functions` module, which will export a data file for each of the selected sonic anemometers.
- Part 13: Compiles and exports a file containing the coordinates of all selected sonic anemometers in Easthing, Northing, height.

Module code:

```
# Mode 1 - Chosen sonics per tower and height

'''This module extracts, processes and exports data for selected sonics in
    selected towers in the dates defined by the user'''

import tower_location as tl
import processing_functions as pr
import input_functions as ip
import download as dl
import pandas as pd
import numpy as np
import os.path

```

```
'''Initialization of variables'''

coord_index_array = np.array(-1)
tl.coordinates_file_creation()

'''----- PART 1 ----------'''

# Ask for the period time of the sample, multiples of 5 min: 5, 10, 15, 20,
    30min or 1h, doesn't accept other time periods

k = ip.input_time_period()

'''----- PART 2 ----------'''

# define an array that contains the dates defined by the user - Ask for input of
    time period

dates_def = ip.input_dates_defined()

if dates_def.size > 1:
    z1 = int(dates_def[0])
    z2 = int(dates_def[-1])

else:
    z1 = int(dates_def)
    z2 = int(dates_def)

# z1 - Start date
# z2 - End date

# the array dates_def contains all the dates defined by the user for extracting
    data, every position of the array contains the date of 1 day with the format
    YYYYMMDD

'''----- PART 3 ----------'''

# Ask for input of which hours of the day or days are of interest
# z3 = start hour
# z4 = ending hour

# Sectioning mode - User selects if the selected days have data for the whole
    day (sm = 1), just a time interval per day (sm = 2) or continuous data from
    the start hour to the end hour (sm = 3)

sm = ip.input_sectioning_mode()

if sm == 1:
    z3 = 0
    z4 = 24

if sm == 2:

    sta_end = ip.input_hours_sm2()
```

```
60
61      z3 = int(sta_end[0])
62      z4 = int(sta_end[1])
63
64  if sm == 3:
65
66      sta_end = ip.input_hours_sm3(z1, z2)
67
68      z3 = int(sta_end[0])
69      z4 = int(sta_end[1])
70
71  '''----- PART 4 ----------'''
72
73  # Download files matching the dates defined by the user
74
75  dl.download(dates_def)
76
77  '''----- PART 5 ----------'''
78
79  # Input section on wich mast and height the sonic data will be extracted
80
81  x_j = ip.input_multiple_towers()
82
83  # x_arr - Contains the towers names defined by the user, j_arr - Contains the
        index position of the towers defined by the user
84
85  x_arr = x_j[0]
86  j_arr = x_j[1]
87
88  # Ask if the heights selection will be done individually or for all towers
89
90  hs = ip.height_selection_mode()
91
92  #  Ask for input of sonic height, the code will not continue until a valid
        height for the choosen tower is given
93
94  if hs == 1:
95      # input section for all heigts needed
96      y_arr = ip.height_selection_hs1()
97
98  if hs == 2:
99      # loop through towers to learn sonics heights for each one
100     sdf = ip.height_selection_hs2(x_arr, j_arr)
101
102
103 '''----- PARTS 6-12 ----------'''
104
105 print("Gathering data and exporting")
106
107 # Just to explain user what is happening
108
109 # Loop through the selected sonic heights and towers, in order to retrieve the
        data
```

```
110
111 if hs == 1:
112
113     height = tl.sonics_height_array()
114     m = tl.sonics_towers_map()
115
116     if y_arr.size > 1:
117
118         for y in y_arr:
119
120             y = int(y)
121             i = tl.height_index(y)
122
123             # Loop through selected towers, check wich ones match the selected
                    heights and call the process_routines function to process and
                    export the data in each one of them
124             a = 0
125
126             if x_arr.size > 1:
127
128                 for x in x_arr:
129
130                     j = int(j_arr[a])
131                     a +=1
132
133                     if m[i,j] == 1:
134
135                         # this function compiles every process of data gathering,
                                calculate turbulence parameters, adjust the time of
                                the samples, adjust the time period through the day
                                and saving and exporting the data for 1 sonic in the
                                designated time period.
136
137                         coord_index_array = pr.process_routines(dates_def, z1, z2,
                                z3, z4, x, y, k, sm, coord_index_array)
138
139             else:
140
141                 x = str(x_arr)
142                 j = int(j_arr)
143
144                 if m[i,j] == 1:
145
146                     coord_index_array = pr.process_routines(dates_def, z1, z2, z3,
                            z4, x, y, k, sm, coord_index_array)
147
148
149     else:
150         y = int(y_arr)
151         i = tl.height_index(y)
152
153         a = 0
154
```

```
155         if x_arr.size > 1:
156 
157             for x in x_arr:
158 
159                 j = int(j_arr[a])
160                 a +=1
161 
162                 if m[i,j] == 1:
163 
164                     coord_index_array = pr.process_routines(dates_def, z1, z2, z3,
                            z4, x, y, k, sm, coord_index_array)
165 
166         else:
167 
168             x = str(x_arr)
169             j = int(j_arr)
170 
171             if m[i,j] == 1:
172 
173                 coord_index_array = pr.process_routines(dates_def, z1, z2, z3,
                        z4, x, y, k, sm, coord_index_array)
174 
175 
176 if hs == 2:
177 
178     #Loop through selected towers
179 
180     if x_arr.size > 1:
181         i = 0
182         j = 0
183         for x in x_arr:
184 
185             i += 1
186 
187             y_arr = sdf[:,j]
188             y_arr = np.unique(y_arr)
189             y_arr = y_arr[1:]
190 
191             j += 1
192 
193             # Loop through all the sonic heights selected for that tower and call
                    the process_routines function to process and export the data for
                    each sonic in that tower
194             if y_arr.size > 1:
195 
196                 for y in y_arr:
197 
198                     y = int(y)
199 
200                     coord_index_array = pr.process_routines(dates_def, z1, z2, z3,
                            z4, x, y, k, sm, coord_index_array)
201 
202             else:
```

```
203
204                 y = int(y_arr)
205
206                 coord_index_array = pr.process_routines(dates_def, z1, z2, z3,
                        z4, x, y, k, sm, coord_index_array)
207
208     else:
209         x = str(x_arr)
210
211         y_arr = sdf
212
213         if y_arr.size > 1:
214
215             for y in y_arr:
216
217                 #y = int(y)
218
219                 coord_index_array = pr.process_routines(dates_def, z1, z2, z3,
                        z4, x, y, k, sm, coord_index_array)
220
221         else:
222
223             y = int(y_arr)
224
225             coord_index_array = pr.process_routines(dates_def, z1, z2, z3, z4, x,
                    y, k, sm, coord_index_array)
226
227
228 '''----- PART 13 ----------'''
229
230 df_coord = pd.read_csv ('sonics_coord_est_nor_z.csv', usecols=
        ['Easting','Northing','Z','sonic'])
231
232 coord_index_array = np.unique(coord_index_array)
233 coord_index_array = coord_index_array[1:]
234
235 df_sonics = df_coord.loc[coord_index_array]
236
237 direc = r'./data_pf'
238
239 df_sonics.to_csv(os.path.join(r'{}'.format(direc),'selected_sonics_coordinates.csv'))
240
241 df_sonics.to_excel(os.path.join(r'{}'.format(direc),'selected_sonics_coordinates.xls'))
242
243 df_sonics.to_csv(os.path.join(r'{}'.format(direc),'selected_sonics_coordinates.dat'),
        sep = " ", index=False, header=False)
```

B.1.3 mode_02_all_sonics_multiple_masts

Table B.3: Mode 2 defined variables

Variable	Description
`dates_def`	Dates defined by the user – an array containing in sequence all the days dates that the user as chosen to extract data from
`z1, z2`	Start and end date – those can be different dates or the same, if the user only wants data from one day
`z3, z4`	Start and end hour – the user can specify a start hour and an end hour for each day or a start hour for the first day and an end hour for the last day
`k`	Time period conversion factor – if the user wants to retrieve the data in a time period different than 5min, a conversion will be made using this factor
`sm`	Time sectioning flag variable: the user might choose an alternative to having the full day data, it can be specific parts of every day or, a start hour for the first day and an end hour for the last day, this flag variable will be used to process that request
`hs`	Height selection mode flag variable - the user specifies how the height selection process will occur, same height(s) for all selected towers or select individually sonic height(s) for each selected tower
`x_arr`	Array with selected towers name – to extract the data from any sonic anemometer, a tower name is necessary, this array provides that name
`j_arr`	Array with selected towers name index position – used in supporting operations in mode 1
`hei, i, j`	Sonic anemometer supporting variables - variables used to loop through the selected sonic anemometers in mode 2 B.1.3
`m`	Sonic anemometer availability - array containing sonic anemometer availabilty for each tower
`height`	Heights array
`x, y`	sonic anemometer variables used to call `processing_functions` module

In `mode_02_all_sonics_multiple_masts`, as in the other modes, the user starts to select the time variables for data retrieval. Then, any number of masts can be selected, and from those masts all sonic anemometers for each mast are automatically selected for data retrieval.

The code is divided in "Parts" that will be explained bellow:

- Part 1: The user selects the time period of the sample, that can be the standard 5-min data, or 10, 20, 30 or 60 min. `k` (conversion factor) is stored as a variable that will be used for time conversion of data if needed.
- Part 2: The user selects a start and end date to retrieve data from. Three variables are stored: `dates_def` (array containing all the dates defined, can have multiple or only one date), `z1` (start date), `z2` (end date). The `dates_def` array will have

sequential dates from the start date until the end date.

- Part 3: The user might choose if the data is to be delivered for all 24h per day, just for a time interval in each day (e.g. from 12:00 to 18:00 every day) or continuous data from a selected hour in the first day until a selected hour in the last day (which may be the same as the first day). This information is stored in `sm` (time sectioning flag variable), and if the option chosen was not the full 24h per day, `z3` (start hour variable) and `z4` (end hour variable) will be stored as well.
- Part 4: the NetCDF files concerning the dates defined by the user are downloaded.

  These first four parts are common to all the modes. The differentiation of the modes appears in the next parts to be presented. As explained before, in terms of data retrieval, each sonic anemometer is located by a pair of variables: tower name and sonic height.
- Part 5: The user types any number of tower names. That information is stored in a pair of variables, `x_arr` (array containing the names of the selected towers) and `j_arr` (array containing the matching index position of the selected towers name). In this mode, all sonic anemometers for each tower are selected automatically, so the user only selects the towers.
- Parts 6-12: Using the defined variables in the first 5 parts, the app will loop through the sonic anemometer variables to call the process_routines sub-module, which will export a data file for each of the selected sonic anemometers.
- Part 13: Compiles and exports a file containing the coordinates of all selected sonics in Easthing, Northing, height.

Module code:

```
# Mode 2 - All sonics multiple towers

'''This module extracts, processes and exports data for all sonics in selected
    towers in the dates defined by the user'''

import tower_location as tl
import processing_functions as pr
import input_functions as ip
import download as dl
import pandas as pd
import numpy as np
import os.path

'''Initialization of variables'''

coord_index_array = np.array(-1)
tl.coordinates_file_creation()

'''----- PART 1 ----------'''

# - Ask for the period time of the sample, multiples of 5 min: 5, 10, 15, 20,
    30min or 1h, doesn't accept other time periods
```

```
21
22 k = ip.input_time_period()
23
24 '''----- PART 2 ----------'''
25
26 # define an array that contains the dates defined by the user - Ask for input of
       time period
27
28 dates_def = ip.input_dates_defined()
29
30 if dates_def.size > 1:
31     z1 = int(dates_def[0])
32     z2 = int(dates_def[-1])
33
34 else:
35     z1 = int(dates_def)
36     z2 = int(dates_def)
37
38 # z1 - Start date
39 # z2 - End date
40
41 # the array dates_def contains all the dates defined by the user for extracting
       data, every position of the array contains the date of 1 day with the format
       YYYYMMDD
42
43 '''----- PART 3 ----------'''
44
45 # Ask for input of which hours of the day or days are of interest
46 # z3 = start hour
47 # z4 = ending hour
48
49 # Sectioning mode - User selects if the selected days have data for the whole
       day (sm = 1), just a time interval per day (sm = 2) or continuous data from
       the start hour to the end hour (sm = 3)
50
51 sm = ip.input_sectioning_mode()
52
53 if sm == 1:
54     z3 = 0
55     z4 = 24
56
57 if sm == 2:
58
59     sta_end = ip.input_hours_sm2()
60
61     z3 = int(sta_end[0])
62     z4 = int(sta_end[1])
63
64 if sm == 3:
65
66     sta_end = ip.input_hours_sm3(z1, z2)
67
68     z3 = int(sta_end[0])
```

```
69      z4 = int(sta_end[1])
70
71  '''----- PART 4 ----------'''
72
73  # Download files matching the dates defined by the user
74
75  dl.download(dates_def)
76
77  '''----- PART 5 ----------'''
78
79  # Input section on wich mast and height the sonic data will be extracted
80
81  x_j = ip.input_multiple_towers()
82
83  # x_arr - Contains the towers names defined by the user, j_arr - Contains the
        index position of the towers defined by the user
84
85  x_arr = x_j[0]
86  j_arr = x_j[1]
87
88
89  '''----- PARTS 6-12 ----------'''
90
91  print("Gathering data and exporting")
92
93  # Just to explain user what is happening
94
95  #Loop through towers (if there was a incorrect use of the mode and only one
        tower was selected, it will work nevertheless)
96
97  if x_arr.size > 1:
98      i = 0
99      for x in x_arr:
100
101         j = j_arr.item(i)
102         j = int(j)
103
104         # Create an array with the sonics heights available for that tower
105
106         hei = tl.sonics_available_name(j)
107
108         i += 1
109
110         # Loop through all the sonics and call the extraction module and
              concatenate the data in order, using time as index, in each one of
              them
111         for y in hei:
112
113             # this function compiles every process of data gathering, calculate
                  turbulence parameters, adjust the time of the samples, adjust the
                  time period through the day and saving and exporting the data for
                  1 sonic in the designated time period.
114
```

```
115             coord_index_array = pr.process_routines(dates_def, z1, z2, z3, z4, x,
                    y, k, sm, coord_index_array)
116
117 else:
118     x = str(x_arr)
119     j = int(j_arr)
120
121     # Create an array with the sonics heights available for that tower
122
123     hei = tl.sonics_available_name(j)
124
125     for y in hei:
126
127         coord_index_array = pr.process_routines(dates_def, z1, z2, z3, z4, x, y,
                k, sm, coord_index_array)
128
129 '''----- PART 13 ----------'''
130
131 df_coord = pd.read_csv ('sonics_coord_est_nor_z.csv', usecols=
    ['Easting','Northing','Z','sonic'])
132
133 coord_index_array = np.unique(coord_index_array)
134 coord_index_array = coord_index_array[1:]
135
136 df_sonics = df_coord.loc[coord_index_array]
137
138 direc = r'C:\Users\Baba\Desktop\João\Tese\Python\Teste_1\Perdigao_python\App
    2.0\data_pf'
139
140 df_sonics.to_csv(os.path.join(r'{}'.format(direc),'selected_sonics_coordinates.csv'))
141
142 df_sonics.to_excel(os.path.join(r'{}'.format(direc),'selected_sonics_coordinates.xls'))
143
144 df_sonics.to_csv(os.path.join(r'{}'.format(direc),'selected_sonics_coordinates.dat'),
    sep = " ", index=False, header=False)
```

B.1.4 mode_03_all_masts_multiple_sonics

Table B.4: Mode 3 defined variables

Variable	Description
`dates_def`	Dates defined by the user – an array containing in sequence all the days dates that the user as chosen to extract data from
`z1, z2`	Start and end date – those can be different dates or the same, if the user only wants data from one day
`z3, z4`	Start and end hour – the user can specify a start hour and an end hour for each day or a start hour for the first day and an end hour for the last day
`k`	Time period conversion factor – if the user wants to retrieve the data in a time period different than 5min, a conversion will be made using this factor
`sm`	Time sectioning flag variable: the user might choose an alternative to having the full day data, it can be specific parts of every day or, a start hour for the first day and an end hour for the last day, this flag variable will be used to process that request
`y_arr`	Array with selected sonic height – to extract the data from any sonic anemometer, a sonic height is necessary, this array provides that name
`tow, i`	Sonic anemometer supporting variables - variables used to loop through the selected sonic anemometers in mode 3 B.1.4
`m`	Sonic anemometer availability - array containing sonic anemometer availabilty for each tower
`x, y`	sonic anemometer variables used to call `processing_functions` module

In `mode_03_all_masts_multiple_sonics`, as in the other modes, the user starts to select the time variables for data retrieval. Then, any number of sonic heights can be selected, all sonic anemometers for the selected heights in all towers are automatically selected for data retrieval.

The code is divided in "Parts" that will be explained bellow:

- Part 1: The user selects the time period of the sample, that can be the standard 5-min data, or 10, 20, 30 or 60 min. `k` (conversion factor) is stored as a variable that will be used for time conversion of data if needed.
- Part 2: The user selects a start and end date to retrieve data from. Three variables are stored: `dates_def` (array containing all the dates defined, can have multiple or only one date), `z1` (start date), `z2` (end date). The `dates_def` array will have sequential dates from the start date until the end date.
- Part 3: The user might choose if the data is to be delivered for all 24h per day, just for a time interval in each day (e.g. from 12:00 to 18:00 every day) or continuous data from a selected hour in the first day until a selected hour in the last day (which may be the same as the first day). This information is stored in `sm` (time

sectioning flag variable), and if the option chosen was not the full 24h per day, z3 (start hour variable) and z4 (end hour variable) will be stored as well.

- Part 4: the NetCDF files concerning the dates defined by the user are downloaded.

  These first four parts are common to all the modes. The differentiation of the modes appears in the next parts to be presented. As explained before, in terms of data retrieval, each sonic anemometer is located by a pair of variables: tower name and sonic height.

- Part 5: The user types any number of sonic heights. That information is stored in a variable, y_arr (array containing the heights selected). In this mode, all sonic anemometers in all towers for the chosen heights are selected automatically, so the user only selects the sonic heights.

- Parts 6-12: Using the defined variables in the first 5 parts, the app will loop through the sonic anemometer variables to call the process_routines sub-module, which will export a data file for each of the selected sonic anemometers.

- Part 13: Compiles and exports a file containing the coordinates of all selected sonic anemometers in Easthing, Northing, height.

Module code:

```
# Mode 3 - All towers multiple sonics

'''This module extracts, processes and exports data for all towers in selected
    sonic heights in the dates defined by the user'''

import tower_location as tl
import processing_functions as pr
import input_functions as ip
import download as dl
import pandas as pd
import numpy as np
import os.path

'''Initialization of variables'''

coord_index_array = np.array(-1)
tl.coordinates_file_creation()

'''----- PART 1 ----------'''

# - Ask for the period time of the sample, multiples of 5 min: 5, 10, 15, 20,
    30min or 1h, doesn't accept other time periods

k = ip.input_time_period()

'''----- PART 2 ----------'''

# define an array that contains the dates defined by the user - Ask for input of
    time period

```

```
28 dates_def = ip.input_dates_defined()
29
30 if dates_def.size > 1:
31     z1 = int(dates_def[0])
32     z2 = int(dates_def[-1])
33
34 else:
35     z1 = int(dates_def)
36     z2 = int(dates_def)
37
38 # z1 - Start date
39 # z2 - End date
40
41 # the array dates_def contains all the dates defined by the user for extracting
       data, every position of the array contains the date of 1 day with the format
       YYYYMMDD
42
43 '''----- PART 3 ----------'''
44
45 # Ask for input of which hours of the day or days are of interest
46 # z3 = start hour
47 # z4 = ending hour
48
49 # Sectioning mode - User selects if the selected days have data for the whole
       day (sm = 1), just a time interval per day (sm = 2) or continuous data from
       the start hour to the end hour (sm = 3)
50
51 sm = ip.input_sectioning_mode()
52
53 if sm == 1:
54     z3 = 0
55     z4 = 24
56
57 if sm == 2:
58
59     sta_end = ip.input_hours_sm2()
60
61     z3 = int(sta_end[0])
62     z4 = int(sta_end[1])
63
64 if sm == 3:
65
66     sta_end = ip.input_hours_sm3(z1, z2)
67
68     z3 = int(sta_end[0])
69     z4 = int(sta_end[1])
70
71 '''----- PART 4 ----------'''
72
73 # Download files matching the dates defined by the user
74
75 dl.download(dates_def)
76
```

```
'''----- PART 5 ----------'''

# Create input section on wich sonics heights the sonics data will be extracted

# input section for all heigts needed
y_arr = ip.height_selection_hs1()

# At this point we have the dates defined by the user and the sonics heights

'''----- PARTS 6-12 ----------'''

print("Gathering data and exporting")

# Just to explain user what is happening

if y_arr.size > 1:

    for y in y_arr:

        i = tl.height_index(y)

        # Create an array with the towers available for that sonic height

        tow = tl.tower_avaliable_name(i)

        # Loop through all the available towers and call the process_routines
            function to process and export the data in each one of them, for the
            designated sonic height
        for x in tow:

            # this function compiles every process of data gathering, calculate
                turbulence parameters, adjust the time of the samples, adjust the
                time period through the day and saving and exporting the data for
                1 sonic in the designated time period.

            coord_index_array = pr.process_routines(dates_def, z1, z2, z3, z4, x,
                y, k, sm, coord_index_array)

else:

    y = int(y_arr)
    i = tl.height_index(y)

    # Create an array with the towers available for that sonic height

    tow = tl.tower_avaliable_name(i)

    for x in tow:

        coord_index_array = pr.process_routines(dates_def, z1, z2, z3, z4, x, y,
            k, sm, coord_index_array)

'''----- PART 13 ----------'''
```

```
123
124 df_coord = pd.read_csv ('sonics_coord_est_nor_z.csv', usecols=
        ['Easting','Northing','Z','sonic'])
125
126 coord_index_array = np.unique(coord_index_array)
127 coord_index_array = coord_index_array[1:]
128
129 df_sonics = df_coord.loc[coord_index_array]
130
131 direc = r'C:\Users\Baba\Desktop\João\Tese\Python\Teste_1\Perdigao_python\App
        2.0\data_pf'
132
133 df_sonics.to_csv(os.path.join(r'{}'.format(direc),'selected_sonics_coordinates.csv'))
134
135 df_sonics.to_excel(os.path.join(r'{}'.format(direc),'selected_sonics_coordinates.xls'))
136
137 df_sonics.to_csv(os.path.join(r'{}'.format(direc),'selected_sonics_coordinates.dat'),
        sep = " ", index=False, header=False)
```

B.1.5 Input functions

This module only contains functions that are used to obtain the sonic anemometer and time related variables from 2.3.1 Phase 1.

All of the functions in this module use input functions, so that the user can define the variables of interest:

- `input_time_period ()`:

  This function asks the user to input the desired time period (5, 10, 15, 20, 30 or 60 min) and returns a period conversion ratio. doesn't accept a different time period.

- `input_sectioning_mode ()`:

  This function asks the user to input the sectioning mode (Full days, start and end hour for every day or start hour for the first day and end hour for the last day only) and returns a sectioning mode flag variable. doesn't accept a different sectioning mode.

- `input_dates_defined ()`:

  This function asks the user to input the start and end date for data retrieval and returns an array with the defined dates (np array of int). doesn't accept dates that are out of the range of the Perdigão campaign.

- `input_hours_sm2 ()`:

  This function asks the user to input the start and end hour, according to sectioning mode 2, for data retrieval and returns an array with the start and end hours (np array of int). Doesn't accept incompatible hours (e.g. start hour > end hour).

- `input_hours_sm3 ()`:

  This function asks the user to input the start and end hour, according to sectioning

mode 3, for data retrieval and returns an array with the start and end hours (np array of int). Doesn't accept incompatible hours (e.g. if the user defined data retrieval for just one day, start hour can´t surpass end hour).

- `input_multiple_towers ():`

  This function asks the user to input tower code names and returns an array with the towers names typed (np array of string). Doesn't accept no towers selected neither wrong tower code names.

- `input_multiple_sonic_heights ():`

  This function asks the user to input sonic heights and returns an array with the sonic heights names typed (np array of string). Doesn't accept no heights selected neither wrong sonic heights.

- `height_selection_mode ():`

  This function asks the user to input the height selection mode (Selected heights for all selected towers or select heights for each tower manually) and returns a height selection mode flag variable. Doesn't accept a different height selection mode.

- `height_selection_hs1():`

  This function asks the user to input sonic heights and returns an array with the sonic heights typed (np array of int). Doesn't accept no heights selected neither wrong sonic heights.

- `height_selection_hs2(x_arr, j_arr):`

  This function asks the user to input sonic heights for each of the previously selected towers and returns an 2D array with the sonic heights typed for each tower (np array of int). Doesn't accept no heights selected neither wrong sonic heights for each specific tower.

Module code:

```
1  # This module contains the input routines defined as functions
2
3  import numpy as np
4  import dates_array
5  import tower_location as tl
6
7  # Time period - Ask for the period time of the sample, multiples of 5 min: 5,
       10, 15, 20, 30min or 1h, doesn't accept other time periods
8
9  def input_time_period ():
10     '''Takes none, returns period conversion ratio (int)'''
11     p0 = 0
12
13     while p0 != 1:
14         period_val = input("Type one of the options for time period in min:\n5,
               10, 15, 20, 30 or 60:")
15
16         p = int(period_val)
```

```
17
18          if p == 5 or p == 10 or p == 15 or p == 20 or p == 30 or p == 60:
19              p0 = 1
20              continue
21          else:
22              print("Invalid time period")
23              continue
24
25      period_conv = int(period_val)/5
26      period_conv = int(period_conv)
27
28      return period_conv
29
30
31  # Sectioning mode - User selects if the selected days have data for the whole
        day, just a time interval per day or continuous data from the start hour to
        the end hour
32
33  def input_sectioning_mode ():
34      '''Takes none, Returns sectioning mode code(int)'''
35      b1 = 0
36      while b1 != 1:
37
38          m = input("Select day time sectioning:\n[1] Full days\n[2] Start and end
                hour for every day\n[3] Start hour for the first day and end hour for
                the last day only\nPlease select desired mode:")
39          m = int(m)
40
41          if m == 1:
42              b1 = 1
43              break
44          if m == 2:
45              b1 = 1
46              break
47          if m == 3:
48              b1 = 1
49              break
50          else:
51              print("Please select [1],[2] or [3]")
52              continue
53      return m
54
55
56  # define an array that contains the dates defined by the user - Ask for input of
        time period
57
58  def input_dates_defined ():
59      '''Takes none, Returns array with dates defined (np array of int)'''
60      dates_tot = dates_array.dates_tot_function()
61      dates_16 = dates_array.dates_16_function()
62
63      a4 = 0
64
```

```
65      while a4 != 1:
66          z1 = input("Select start date with the format, YYYYMMDD:")
67          z1 = int(z1)
68          for c in dates_tot:
69              if z1 == c:
70                  a4 = 1
71                  break
72          if a4 != 1:
73              print("Start date incorrect")
74              print("Possible start dates:")
75              print(dates_16)
76              print("And every date from 16 January 2017 to 1 July 2017")
77          continue
78
79      a5 = 0
80
81      while a5 != 1:
82          z2 = input("Select end date with the format, YYYYMMDD:")
83          z2 = int(z2)
84          for c in dates_tot:
85              if z2 == c:
86                  if z2 >= z1:
87                      a5 = 1
88                      break
89                  else:
90                      print("End date can not be a previous date from the starting
                            date")
91
92
93          if a5 != 1:
94              print("Start date incorrect")
95              print("Possible start dates:")
96          continue
97
98      # using defined sart and end dates (z1, z2) to create an array with the
            defined dates
99      # This for cycle is showing an error (dates_def appears first for append
            than for creation, the fact is that the if line for the append will not
            ever be executed before the if line for creation)
100
101     a6 = 0
102
103     for q in dates_tot:
104         if a6 == 1:
105             temp = np.array(q, dtype='i4')
106             dates_def = np.append(dates_def, [temp])
107         if z1 == q:
108             dates_def = np.array(z1, dtype = 'i4')
109             a6 = 1
110         if z2 == q:
111             a6 = 0
112
113     # the array dates_def contains all the dates defined by the user for
```

```
        extracting data, every position of the array contains the date of 1 day
        with the format YYYYMMDD

    return dates_def


# Ask for input of which hours of the day or days are of interest for every day
    defined by the user

def input_hours_sm2 ():
    '''Takes none, Returns array with start and end hour (np array of int)'''
    hour_arr = np.arange(0, 25)

    a7 = 0

    while a7 != 1:
        z3 = input("Select start hour from 0-23:")
        z3 = int(z3)
        for c in hour_arr:
            if z3 == c and z3 != 24:
                a7 = 1
                break
        if a7 != 1:
            print("Start hour incorrect")
            print("Possible start hours:")
            print(hour_arr[0:24])
        continue

    a8 = 0

    while a8 != 1:
        z4 = input("Select ending hour from {}-24:".format(z3+1))
        z4 = int(z4)
        for c in hour_arr:
            if z4 == c and z4 > z3:
                a8 = 1
                break
        if a8 != 1:
            print("Ending hour incorrect")
            print("Possible ending hours:")
            print(hour_arr[z3+1:25])
        continue

    hours = np.array([z3, z4])

    return hours


# Ask for input of the start hour and end hour for the time period defined by
    the user

def input_hours_sm3 (z1, z2):
    '''Takes none, Returns array with start and end hour (np array of int)'''
```

```
163     hour_arr = np.arange(0, 25)
164
165     if z1 == z2:
166
167         a7 = 0
168
169         while a7 != 1:
170             z3 = input("Select start hour from 0-23:")
171             z3 = int(z3)
172             for c in hour_arr:
173                 if z3 == c and z3 != 24:
174                     a7 = 1
175                     break
176             if a7 != 1:
177                 print("Start hour incorrect")
178                 print("Possible start hours:")
179                 print(hour_arr[0:24])
180             continue
181
182         a8 = 0
183
184         while a8 != 1:
185             z4 = input("Select ending hour from {}-24:".format(z3+1))
186             z4 = int(z4)
187             for c in hour_arr:
188                 if z4 == c and z4 > z3:
189                     a8 = 1
190                     break
191             if a8 != 1:
192                 print("Ending hour incorrect")
193                 print("Possible ending hours:")
194                 print(hour_arr[z3+1:25])
195             continue
196
197         hours = np.array([z3, z4])
198
199         return hours
200
201     else:
202
203         a7 = 0
204
205         while a7 != 1:
206             z3 = input("Select start hour from 0-23:")
207             z3 = int(z3)
208             for c in hour_arr:
209                 if z3 == c and z3 != 24:
210                     a7 = 1
211                     break
212             if a7 != 1:
213                 print("Start hour incorrect")
214                 print("Possible start hours:")
215                 print(hour_arr[0:24])
```

```
216             continue
217
218         a8 = 0
219
220         while a8 != 1:
221             z4 = input("Select ending hour from 1-24:")
222             z4 = int(z4)
223             for c in hour_arr:
224                 if z4 == c and z4 != 0:
225                     a8 = 1
226                     break
227             if a8 != 1:
228                 print("Ending hour incorrect")
229                 print("Possible ending hours:")
230                 print(hour_arr[1:25])
231             continue
232
233         hours = np.array([z3, z4])
234
235         return hours
236
237
238 # Input section to learn which towers are of interest. Can be used to get only
      one tower or multiple tow
239
240 def input_multiple_towers ():
241     '''Takes none, Returns array with the towers names typed (np array of
          string)'''
242     # Array with tower name code
243
244     t_name = tl.towers_name()
245
246     # Ask for input of tower code name, the code will not continue until a valid
          code name is given
247
248     a1 = 0
249     a2 = 0
250     a3 = 0
251
252     print("Type 1 tower code name and press enter.\nYou will be asked for a new
          tower code name until you type[end]")
253
254     while a3 != 1:
255         a1 = 0
256         j = 0
257         x = input("Tower name code:")
258         for a in t_name:
259
260             if x == 'end':
261                 if a2 == 0:
262                     print("Select at least 1 tower")
263                     a1 = 1
264                 if a2 == 1:
```

```
265                     a3 = 1
266                     a1 = 1
267 
268                 break
269 
270             if x == a:
271 
272                 if a2 == 1:
273                     x_arr = np.append(x_arr, x)
274                     j_arr = np.append(j_arr, j)
275 
276                 if a2 == 0:
277                     x_arr = np.array(x)
278                     j_arr = np.array(j)
279                     x0 = x
280                     j0 = j
281                     a2 = 1
282 
283                 a1 = 1
284                 break
285             j = j + 1
286         if a1 != 1:
287             print("Code name incorrect")
288 
289         continue
290 
291     # At this point we have the dates defined by the user and the towers
292 
293     x_j_arrays = np.array([x_arr, j_arr])
294     return x_j_arrays
295 
296 
297 # Input section to learn which sonic heights are of interest. Can be used to get
        only one sonic height or multiple sonic heights - for mode all towers
        multiple sonics
298 
299 def input_multiple_sonic_heights():
300     '''Takes none, returns array with the sonic heights names typed (np array of
            string)'''
301     # Ask for input of sonic height, the code will not continue until a valid
            code name is given
302 
303     height = tl.sonics_height_array()
304     m = tl.sonics_towers_map()
305 
306     a2 = 0
307     a3 = 0
308 
309     print("Type 1 sonic height and press enter.\nYou will be asked for a new
            sonic height until you type[end]")
310 
311     while a3 != 1:
312         i = 0
```

```
313         a1 = 0
314         y = input("Sonic height:")
315         for a in height:
316 
317 
318             if y == 'end':
319                 a3 = 1
320                 a1 = 1
321                 break
322 
323             if y == a:
324 
325                 if a2 == 1:
326                     y_arr = np.append(y_arr, y)
327                     i_arr = np.append(i_arr, i)
328 
329                 if a2 == 0:
330                     y_arr = np.array(y)
331                     i_arr = np.array(i)
332                     y0 = y
333                     i0 = i
334                     a2 = 1
335 
336                 a1 = 1
337                 break
338             i = i + 1
339         if a1 != 1:
340             print("Sonic height incorrect")
341 
342         continue
343 
344 
345 # Input function to learn height selection mode
346 
347 def height_selection_mode ():
348     '''Takes none, Returns code variable for height mode selection (int)'''
349     a = 0
350     while a != 1:
351         hs = input("Please type height selection mode:\n[1] Selected heights for
                all selected towers\n[2] Select heights for each tower
                manually\nPlease select desired mode:")
352         try:
353             hs = int(hs)
354         except:
355             pass
356         if hs == 1 or hs ==2:
357             a = 1
358             break
359         else:
360             print("Please type '1' or '2'")
361             continue
362     return hs
363 
```

```
364
365 # Input section to learn which sonic heights are of interest for every tower.
        Can be used to get only one sonic height or multiple sonic heights - for
        mode chosen towers choosen sonics and chosen sonic height for all towers
366
367 def height_selection_hs1():
368     '''Takes none, Returns np array with the selected heights for every tower(np
            array of int)'''
369     height = tl.sonics_height_array()
370     a2 = 0
371     a3 = 0
372
373     print("Type 1 sonic height and press enter.\nYou will be asked for a new
            sonic height until you type[end]")
374
375     while a3 != 1:
376         a1 = 0
377         y = input("Sonic height:")
378         for a in height:
379
380             if y == 'end':
381                 a3 = 1
382                 a1 = 1
383                 break
384
385
386             if y == a:
387
388                 try:
389                     y = int(y)
390
391                     if a2 == 1:
392                         y_arr = np.append(y_arr, y)
393                         a1 = 1
394
395                     if a2 == 0:
396                         y_arr = np.array(y)
397                         y0 = y
398
399                         a1 = 1
400                         a2 = 1
401
402                     a1 = 1
403
404                 except:
405                     pass
406
407             if a1 == 1:
408                 break
409
410
411         if a1 != 1:
412             print("Sonic height incorrect")
```

```
413             print("Available sonic heights:")
414             print(height)
415 
416         continue
417 
418     if y_arr.size > 1:
419 
420         y_arr = np.sort(y_arr)
421 
422     y_arr = np.unique(y_arr)
423 
424     return y_arr
425 
426 
427 # Input section to learn which sonic heights are of interest for each tower. Can
        be used to get only one sonic height or multiple sonic heights per tower -
        for mode chosen towers chosen sonics
428 
429 def height_selection_hs2(x_arr, j_arr):
430     '''Takes x_arr and j_arr (np arrays of string), Returns 2D np array with the
            selected heights for each tower(np 2D array of int)'''
431     if x_arr.size == 1:
432 
433         j = int(j_arr)
434         x = x_arr[0]
435         san = tl.sonics_available_name(j)
436         print("Available heights for {}:".format(x))
437         print(san)
438 
439         a2 = 0
440         a4 = 0
441         a5 = 0
442         while a2 != 1:
443 
444             y = input("Tower height:")
445 
446             if y == 'end' and a4 == 1:
447                 a2 = 1
448                 a5 = 1
449                 #a4 = 0
450                 break
451 
452             try:
453                 y = int (y)
454             except:
455                 pass
456 
457             for b in san:
458                 if y == b:
459 
460                     if a4 == 1:
461                         sn = np.append(sn, y)
462 
```

```
463                 if a4 == 0:
464                     sn = np.array(y)
465                     a4 = 1
466 
467 
468                 a5 = 1
469                 break
470 
471             if a5 != 1:
472                 print("Select an appropriate height")
473 
474             continue
475 
476         sdf = np.array([sn])
477         sdf = np.sort(sdf)
478         sdf = np.unique(sdf)
479 
480     else:
481 
482         sdf = np.zeros((11,50))
483 
484         j1 = 0
485         i1 = 0
486 
487         for a in x_arr:
488 
489             j = int(j_arr[j1])
490             san = tl.sonics_available_name(j)
491             print("Available heights for {}:".format(a))
492             print(san)
493 
494             a2 = 0
495             a4 = 0
496             while a2 != 1:
497 
498                 a5 = 0
499 
500                 y = input("Tower height:")
501 
502                 if y == 'end' and a4 == 1:
503                     a2 = 1
504                     a5 = 1
505                     a4 = 0
506                     break
507 
508                 try:
509                     y = int (y)
510                 except:
511                     pass
512 
513                 for b in san:
514                     if y == b:
515 
```

```
                    if a4 == 1:

                        sdf[i1,j1] = y
                        temp = sdf[:, j1]
                        temp = np.unique(temp)
                        temp = np.sort(temp)
                        temp = temp[1:]
                        a = temp.size
                        b = 11-a
                        while b>0:
                            temp = np.append(temp, 0)
                            b -=1

                        sdf[:, j1] = temp[:]

                    if a4 == 0:

                        sdf[i1,j1] = y
                        a = 1
                        a4 = 1

                    i1 = a
                    a5 = 1
                    break

            if a5 != 1:
                print("Select an appropriate height")

            continue

        j1 += 1
        i1 = 0

return sdf
```

B.1.6 Dates array

This module only contains functions that are used to build arrays that contain all the possible dates for data extracting, matching the dates of the Perdigão campaign.

Those dates arrays will be used as support to many functions of the app.

Functions:

- `dates_16_function ():`

  This function creates an array with the available dates for 2016 (np array of int) and returns it.

- `dates_tot_function ():`

  This function creates an array with all the available dates (np array of int) and returns it.

Module code:

```
# File containing arrays with the dates used for downloading and extracting data of the netCD fiiles

# The array 'dates' will contain all the dates of the days with available data

def dates_16_function ():
    '''Takes none, Returns array with the available dates for 2016 (np array of int)'''
    dates = np.array([20161129, 20161201, 20161202, 20161205], dtype='i4')

    i = 7

    while i<10:
        s = '2016120'+ str(i)
        temp = np.array(s, dtype='i4')
        dates = np.append(dates, [temp])
        i = i+1

    dates = np.append(dates, [20161210])

    # Array with the dates of the days with available data for the months of November and Dezember of 2016

    dates_16 = dates
    return dates_16


def dates_tot_function ():
    '''Takes none, Returns array with all the available dates (np array of int)'''
    dates = dates_16_function()

    # Filling the 'dates' array with the remaining dates of the year 2017

    j = 1
    i = 16

    # Array with the dates of the days with available data for the months from January until July of 2017

    # 1- January, 3- March, 5- May months with 31 days
    # 2- February - 29 days
    # 4- April, 6- June

    while j<7:
        if j == 1 or j == 3 or j == 5:
            while i < 10:
                s = '20170' + str(j)+'0'+ str(i)
                temp = np.array(s, dtype='i4')
                dates = np.append(dates, [temp])
                i = i+1
```

```
            else:
                while i <32:
                    s = '20170' + str(j)+ str(i)
                    temp = np.array(s, dtype='i4')
                    dates = np.append(dates, [temp])
                    i = i+1
                else:
                    i=1

        if j == 2:
            while i <10:
                s = '20170' + str(j)+ '0' + str(i)
                temp = np.array(s, dtype='i4')
                dates = np.append(dates, [temp])
                i = i+1
            else:
                while i <29:
                    s = '20170' + str(j)+ str(i)
                    temp = np.array(s, dtype='i4')
                    dates = np.append(dates, [temp])
                    i = i+1
                else:
                    i = 1
        if j == 4 or j == 6:
            while i <10:
                s = '20170' + str(j)+ '0' + str(i)
                temp = np.array(s, dtype='i4')
                dates = np.append(dates, [temp])
                i = i+1
            else:
                while i < 31:
                    s = '20170' + str(j)+ str(i)
                    temp = np.array(s, dtype='i4')
                    dates = np.append(dates, [temp])
                    i = i+1
                else:
                    i = 1
        j = j+1

    else:
        dates = np.append(dates, [20170701])

    dates_tot = dates

    return dates_tot
```

B.1.7 Tower location

This module only contains functions relative to tower and sonic location.

The functions from this module are used throughout the app, from the input functions to processing data and coordinates files processing:

- `towers_name ()`:

  This function creates an array with all the available tower names (np array of string) and returns it.

- `tower_index_pos(name)`:

  This function a tower name and returns it's index position in the towers name array created by the towers_name () function.

- `lat_lon_towers ()`:

  This function creates a 2D array that contains the latitude and longitude values, extracted from the '.nc' files, for every tower, matching the index position in the towers name array created by the `towers_name ()` function. The 'nc' files don't have data for all the towers, so, that missing data is manually inserted. The data is in Decimal Degrees (DD).

- `sonics_height_array ()`:

  This function creates an array with all the available heights (np array of string) and returns it.

- `sonics_towers_map ()`:

  This function creates a 2D array with the availability ('0' or '1') of sonic heights in each tower (np 2D array of int) and returns it.

- `sonics_height (t)`:

  This function creates an array with the availability ('0' or '1') of sonic heights for the selected tower (np array of int) and returns it.

- `tower_available (d)`:

  This function creates an array with the availability ('0' or '1') of towers for the selected sonic height (np array of int) and returns it.

- `sonics_available_name (j)`:

  This function takes the index position of the selected tower, creates an array with the available sonic heights for that tower (np array of int) and returns it.

- `tower_available_name (i)`:

  This function takes the index position of the selected sonic height, creates an array with the available towers names for that height (np array of int) and returns it.

- `height_index (y)`:

  This function takes an height and returns the index position in `sonics_height_array` (int).

- `coordinates_file_creation()`:

  This function opens a file containing a table with raw data of all sonic anemometers, processes that data to obtain a data frame containing coordinates (PT-TM06/ETRS89),

height and sonic full name for all sonic anemometers. That data frame is saved and exported to the data folder where the app is located. Finally, the data framed is returned (Pandas Data Frame).

Module code:

```
# Tower codes and geo location

from netCDF4 import Dataset
import numpy as np
import pandas as pd
import download as dl

date = np.array([20170614])
dl.download(date)

# Array containing all towers names

def towers_name ():
    '''Takes none, Returns array with towers code name (np array of string)'''
    t_name = np.array(["tnw01", "tnw02", "tnw03", "tnw04", "tnw05", "tnw06",
        "tnw07", "tnw08", "tnw09", "tnw10", "tnw11", "tnw12", "tnw13", "tnw14",
        "tnw15", "tnw16", "tse01", "tse02", "tse04", "tse05", "tse06", "tse07",
        "tse08", "tse09", "tse10", "tse11", "tse12", "tse13", "rsw01", "rsw02",
        "rsw03", "rsw04", "rsw05", "rsw06", "rsw07", "rsw08", "rne01", "rne02",
        "rne03", "rne04", "rne06", "rne07", "v01", "v03", "v04", "v05", "v06",
        "v07", "Extreme_SW", "Extreme_NE"])
    return t_name


# Tower index position by name

def tower_index_pos(name):
    '''Takes tower name, Returns tower index position in tower name array
        (int)'''
    t_name = towers_name()
    j = np.where(t_name == name)
    j = int(j[0])

    return j


# Function returns a 2D np array that contains 2 1D np arrays, the first
    contains the lat for every tower and the second contains the lon

def lat_lon_towers ():
    '''Takes none, Returns 2D np array with lat and lon arrays (np arrays of
        float)'''
    # Reading the netcd file
    data = Dataset("20170614.nc", 'r')

    t_name = towers_name()

```

```
39      lat_n = np.empty(50)
40      lon_n = np.empty(50)
41
42      # Saving the coordinates in arrays
43
44      i = 0
45
46      for x in t_name:
47          try:
48              lat_data = data.variables['latitude_{}'.format(x)][:]
49              lon_data = data.variables['longitude_{}'.format(x)][:]
50          except KeyError:
51              #print("The {} variable has no geo coordinates".format(x))
52              continue
53          except:
54              #print("There is other problem with the {} variable".format(x))
55              continue
56          else:
57              lat_n[i] = lat_data
58              lon_n[i] = lon_data
59          finally:
60              i = i + 1
61
62      # The prints that appear in comments where used to learn which towers didn't
            have coordinates info, which will be introduced manually later
63
64      # Insert manually the coordinates of the towers that had missing info
65      # Data retrieved from Perdigao Site, data was in Degrees/Minutes/Seconds
            (DMS) format and was converted to Decimal Degrees (DD) to match the rest
            of the data
66
67      # tnw04 7°44'47.12"W 39°42'44.37"N
68
69      lat_n[3] = 39.712325
70      lon_n[3] = -7.746422
71
72      # tnw12 7°44'7.64"W 39°43'8.65"N
73
74      lat_n[11] = 39.719069
75      lon_n[11] = -7.735456
76
77      # tnw13 7°44'5.74"W 39°43'10.59"N
78
79      lat_n[12] = 39.719608
80      lon_n[12] = -7.734928
81
82      # tnw14 7°44'2.37"W 39°43'11.88"N
83
84      lat_n[13] = 39.719967
85      lon_n[13] = -7.733992
86
87      # tnw15 7°44'0.18"W 39°43'12.72"N
88
```

```
89      lat_n[14] = 39.720200
90      lon_n[14] = -7.733383
91
92      # tnw16 7°43’56.95"W 39°43’13.53"N
93
94      lat_n[15] = 39.720425
95      lon_n[15] = -7.732486
96
97      # tse05 7°44’31.24"W 39°42’24.82"N
98
99      lat_n[19] = 39.706894
100     lon_n[19] = -7.742011
101
102     # Extreme_SW 7°45’51.49"W 39°42’2.64"N
103
104     lat_n[48] = 39.700733
105     lon_n[48] = -7.764303
106
107     # Extreme_NE 7°43’44.56"W 39°43’23.41"N
108
109     lat_n[49] = 39.723169
110     lon_n[49] = -7.729044
111
112     lat_lon_arrays = np.array([lat_n, lon_n])
113     return lat_lon_arrays
114
115
116 # Array containing all sonics heights
117
118 def sonics_height_array ():
119     ’’’Takes none, Returns array with sonics heights (np array of string)’’’
120     # 2m, 4m, 6m, 8m, 10m, 12m, 20m, 30m, 40m, 60m, 80m, 100,
121
122     height = np.array(["2", "4", "6", "8", "10", "12", "20", "30", "40", "60",
            "80", "100"])
123
124     return height
125
126
127 # Map which heights the wind speed sonics are in each tower
128
129 def sonics_towers_map ():
130     ’’’Takes none, Returns np 2D array with availability of sonics for each
            tower (np array of int)’’’
131     # This 2D array is initiallized with zeros "0"
132
133     m = np.zeros((12,50))
134
135     # This ’for loop’ will check in each tower, if there are wind speed values
            for the different heights. If true, it will change the ’m’ array from
            ’0’ to ’1’ in the correspondent position for the height in each tower
136     # In the end, the ’m’ array contains boolean info on whether or not one
            specific tower has a sonic in a specific heihgt, for every tower, and
```

```
        every heights
137
138     t_name = towers_name()
139
140     height = sonics_height_array()
141
142     # Reading the netcd file
143     data = Dataset("20170614.nc", 'r')
144
145     j = 0
146
147     for a in t_name:
148
149         i = 0
150
151         for b in height:
152
153             b = int(b)
154
155             if a == "tnw12" or a == "tnw13" or a == "tnw14" or a == "tnw15" or a
                    == "tnw16":
156
157                 if b == 30:
158
159                     b = 28
160
161             try:
162                 u = data.variables['u_{}m_{}'.format(b, a)][:]
163
164             except KeyError:
165                 #print("The {} height is not present in tower {}".format(b, a))
166                 continue
167             except:
168                 #print("There is other problem with the {} variable".format(a))
169                 continue
170             else:
171                 m[i,j] = 1
172
173             finally:
174                 i = i + 1
175         j = j +1
176
177     # The 'print' on the exceptions where hidden in order not to fill the 'Out'
            whith messages
178     return m
179
180
181 # function retrieves a boolean array containing sonics height for a specific
        tower
182 # input argument is a variable containing a string with tower code name, ex:
        'tnw01'
183
184 def sonics_height (t):
```

```
185     '''Takes tower name (string), Returns array with sonics height availability
            for that tower (np array of int, 0 or 1)'''
186     t_name = towers_name()
187     m = sonics_towers_map()
188     j = np.where(t_name == t)
189     j = j[0]
190     i = 0
191 
192     result_bool = np.array(m[i,j], dtype='i4')
193     i = 1
194     while i < 11:
195         result_bool = np.append(result_bool, m[i,j])
196         i += 1
197         continue
198     return result_bool
199 
200 
201 # function retrieves a boolean array that says if there is a tower that has a
        sonic for the specified heigt
202 # input argument is a variable containing a string with sonic height, ex: '10'
203 
204 def tower_available (d):
205     '''Takes sonic height (string), Returns array with tower availability for
            that sonics height (np array of int, 0 or 1)'''
206     height = sonics_height_array()
207     m = sonics_towers_map()
208     i = np.where(height == d)
209     i = i[0]
210     j = 0
211     global result_bool_h
212     result_bool = np.array(m[i,j], dtype='i4')
213     j = 1
214     while j < 50:
215         result_bool = np.append(result_bool, m[i,j])
216         j += 1
217         continue
218     return result_bool
219 
220 
221 # function retrieves an array with sonics heights available for a given tower
222 
223 def sonics_available_name (j):
224     '''Takes tower position in towers array (int), Returns array with sonics
            heights available (np array of int)'''
225     he = np.empty([12], dtype=int)
226     height = sonics_height_array()
227     m = sonics_towers_map()
228     v = 0
229     i = 0
230     while i < 12:
231         if m[i,j] == 1:
232 
233             he[v] = height[i]
```

```
234             v += 1
235
236         i = i + 1
237         continue
238
239     hei = he[0:v]
240     return hei
241
242
243 # function retrieves an array with towers names available for a given sonic
        height
244
245 def tower_avaliable_name (i):
246     '''Takes sonic position in sonicsz height array (int), Returns array with
            available towers names (np array of string)'''
247     to = towers_name().copy()
248     t_name = towers_name()
249     m = sonics_towers_map()
250
251     t = 0
252     j = 0
253     while j < 50:
254         if m[i,j] == 1:
255
256             to[t] = t_name[j]
257             t += 1
258
259         j = j + 1
260         continue
261
262     tow = to[0:t]
263     return tow
264
265
266 # Function to retrieve index position in sonics_height_array to given height (y)
267
268 def height_index (y):
269     '''Takes height (int), Returns index position in sonics_height_array (int)'''
270     height = sonics_height_array()
271
272     a = 0
273     for h in height:
274         h = int(h)
275         if h == y:
276             i = a
277         a += 1
278     return i
279
280
281 # Creates a file with all sonics coordinates
282
283
284 def coordinates_file_creation():
```

```
285     '''Takes none, Returns data frame containing all sonics coordinates
            (PT-TM06/ETRS89, height, sonic name) (Pandas Data Frame)'''
286     file_name =
            r'C:\Users\Baba\Desktop\João\Tese\Python\Teste_1\Perdigao_python\App
            2.0\raw_files\Relatorio_sonics_data.xlsx'
287     sheet = 'Sheet1'
288
289     df = pd.read_excel(io=file_name, sheet_name=sheet)
290
291     index_array = np.array(-1)
292     index_array2 = np.array(-1)
293
294     lk = np.arange(0,258)
295
296     for i in lk:
297
298         s = df.iat[i,0]
299         s1 = s[0]
300
301         try:
302             a = int(s1)
303
304         except:
305             index_array = np.append(index_array, i)
306
307         else:
308             index_array2 = np.append(index_array2, i)
309
310         finally:
311
312             continue
313
314     index_array = index_array[1:]
315     index_array2 = index_array2[1:]
316
317
318     df = df.loc[index_array2]
319     df = df.reset_index()
320     del df['index']
321
322     lp = np.arange(0,185)
323
324     a=0
325
326     for i in lp:
327
328         if i < 183:
329
330             s = df.iat[i,0]
331             c = 'm'
332
333             j = s.find(c)
334
```

```
335             s = s[j+2:]
336             se = float(s[0:2] + s[3:9])
337             sn = float(s[10:17])
338 
339             if a == 1:
340 
341                 est = np.append(est,se)
342                 nor = np.append(nor,sn)
343 
344             if a == 0:
345 
346                 est = np.array(se)
347                 nor = np.array(sn)
348                 a = 1
349 
350             df.iat[i,0] = s
351 
352 
353         if i > 182:
354 
355             s = df.iat[i,0]
356             c = 'm'
357 
358             j = s.find(c)
359 
360             s = s[j+6:]
361             se = float(s[0:2] + s[3:9])
362             sn = float(s[10:17])
363 
364             est = np.append(est,se)
365             nor = np.append(nor,sn)
366 
367             df.iat[i,0] = s
368 
369     df['Easting'] = est
370     df['Northing'] = nor
371 
372     df2 =
        pd.read_csv(r'C:\Users\Baba\Desktop\João\Tese\Python\Teste_1\Perdigao_python\App
        2.0\raw_files\sonics_coord_corrected.csv')
373 
374     z = df2.loc[:,'Z']
375     sonic = df2.loc[:,'sonic']
376 
377     df['Z'] = z
378     df['sonic'] = sonic
379 
380     del df['sonics']
381 
382     df.to_csv('sonics_coord_est_nor_z.csv', index=False)
383 
384     #df.to_excel('sonics_coordinates.xls')
385 
```

```
386     df.to_csv('sonics_coord_est_nor_z.dat', sep = " ", index=False, header=False)
387 
388     return df
```

B.1.8 Download

This module only contains one function that downloads the .nc files from selected dates.

It was an optional requirement that the files could be downloaded not just from (Gomes et al., 2020) but also from (EOL, 2022) (optional). Option could not be implemented since it required a formal request put in place before the data is sent via email.

Download functions:

- `download (dates\_def):`

  This function takes an array with defined dates (np array of int) and downloads the corresponding '.nc' files from the Perdigão website ((Gomes et al., 2020)).

Module code:

```
1  import requests
2  import numpy as np
3  
4  # Download the netcd files defined by the user from perdigao site
5  
6  def download (dates_def):
7      '''Takes dates def (np array of int), returns none'''
8      print("Downloading...")
9  
10     # Just to explain user what is happening
11 
12     if dates_def.size == 1:
13         date = int(dates_def)
14         url =
               'http://windsptds.fe.up.pt/thredds/fileServer/flux/NCAR-EOL%20Quality%20Controlled%
15         205-minute%20ISFS%20surface%20flux%20data,%20geographic%20coordinate,%20tilt%20corrected/
16         isfs_qc_tiltcor_'+ str(date) +'.nc'
17         r = requests.get(url)
18         open(str(date)+'.nc','wb').write(r.content)
19     else:
20         for date in dates_def:
21             url =
                   'http://windsptds.fe.up.pt/thredds/fileServer/flux/NCAR-EOL%20Quality%20Control
22             led%205-minute%20ISFS%20surface%20flux%20data,%20geographic%20coordinate,%20tilt%20co
23             rrected/isfs_qc_tiltcor_'+ str(date) +'.nc'
24             r = requests.get(url)
25             open(str(date)+'.nc','wb').write(r.content)
```

B.1.9 Processing functions

This module only contains `process_routines`, a function that compiles all functions to extract, process and export the data for one sonic anemometer:

- `process_routines (dates_def, z1, z2, z3, z4, x, y, k, sm, coord_index_array):`

  This function uses the variables defined by the user to obtain the target data using the `data_gathering(dates_def, z1, z2, x, y)` function.

  Then uses the `turbulence_5min(dfc)` function to obtain a new data frame with calculated turbulence parameters (or, if period is different than 5-min, uses `tke_k(dfc,k)`, `ti_k(dfc, k)`, `tih_k(dfc, k)` to obtain turbulence parameters, `period_adjust(z1, z2, dfc, k)` to obtain a new time averaged data frame, and `turb_append(dfc, tke_arr, ti_arr, tih_arr)` to obtain a new time averaged data frame with the calculated turbulence parameters).

  Afterwards uses `section_time_sm2(dfc, z1, z2, z3, z4, k, dates_def)` or `section_time_sm3(dfc, z1, z2, z3, z4, k)` to obtain a new data frame that fulfils the time sectioning requests.

  The data frame is then saved and exported using the `save_export(dfc, x, y, z1, z2, z3, z4, k)` function.

  Finally, using the sonic full name, the index position of that sonic in the sonics coordinates file is appended to the `coord_index_array (np array of int)` that is returned.

Module code:

```
import data_gathering as dg
import save_export_function as se
import time_process as tp
import turbulence_parameters as tu
import pandas as pd
import numpy as np

# From Data gathering to Export routines compiled

def process_routines (dates_def, z1, z2, z3, z4, x, y, k, sm, coord_index_array):
    '''Takes an array with the selected dates (np array of int), start date
       (int), end date (int), start hour (int), end hour (int), tower code name
       (str), sonic height (int), time period conversion value (int),
       sectioning mode variable (int) and coordinates index array (np array of
       int), Returns coordinates index array (np array of int), extracts,
       processes, saves and exports data frames containing the interest
       variables of the Netcd Files'''
    #  Call the extraction module and concatenate the data in order, using time
       as index
    '''----- PART 7 ----------'''

    # Reading the netcd file
    #fi, x and y will be given by the main input

```

```
18      dfc = dg.data_gathering(dates_def, z1, z2, x, y)
19
20      '''----- PART 8 and 9 ----------'''
21
22      # Add turbulence parameters to the 5 min period data frame.
23
24      if k == 1:
25
26          dfc = tu.turbulence_5min(dfc)
27
28      '''----- PART 9 ----------'''
29      # If time period is different from 5 min, retrieves a data frame with the
            mean variables values, according to the time period selected
30      # Turbulence parameters are also recalculated, simple mean of turbulence
            parameters is not accurate
31
32      if k>1:
33
34          uu, vv, ww = tu.var_k(dfc, k)
35          tke_arr = tu.tke_k(dfc, k)
36          ti_arr = tu.ti_k(dfc, k)
37          tih_arr = tu.tih_k(dfc, k)
38
39          dfc = tp.period_adjust(z1, z2, dfc, k)
40          uv = dfc.iloc[:,10].to_numpy()
41          uw = dfc.iloc[:,11].to_numpy()
42          vw = dfc.iloc[:,12].to_numpy()
43
44          dfc = dfc.iloc[:,:7]
45          dfc = tu.var_append(dfc, uu, vv, ww)
46          dfc = tu.covar_append(dfc, uv, uw, vw)
47
48          dfc = tu.turb_append(dfc, tke_arr, ti_arr, tih_arr)
49
50      '''----- PART 10 ----------'''
51      # If the user asked for a specific time period of the day (ex: from 11h to
            17h) this function updates the df to fulfill that order.
52
53      if sm == 2:
54
55          dfc = tp.section_time_sm2(dfc, z1, z2, z3, z4, k, dates_def)
56
57      if sm == 3:
58
59          dfc = tp.section_time_sm3(dfc, z1, z2, z3, z4, k)
60
61      '''----- PART 11 ----------'''
62
63      se.save_export(dfc, x, y, z1, z2, z3, z4, k)
64
65      # When converting the df to a np array the file is loosing the info on the
            time variable (used as index in the df)
66      # Possible solution is create a new df whith the time series as a column, as
```

```
        well as index
67
68      '''----- PART 12 ----------'''
69
70      df_coord = pd.read_csv ('sonics_coord_est_nor_z.csv', usecols=
            ['Easting','Northing','Z','sonic'])
71      sonic_array = df_coord.iloc[:,3]
72
73      name = '{}.{}m'.format(x,y)
74      i = 0
75
76      for sonic in sonic_array:
77          if name in sonic:
78
79              coord_index_array = np.append(coord_index_array, i)
80
81          i +=1
82
83
84      return coord_index_array
```

B.1.10 Data gathering

This module only contains `data_gathering`, a function that aggregates all raw data after extraction for one sonic anemometer, in the defined dates:

- `data_gathering (dates_def, z1, z2, x, y):`

  This function loops through the dates defined array and calls the `extract(z, x, y, n)` function to obtain a data frame (Pandas DataFrame) containing the target raw data for each day. This function also joins in a single continuous data frame all data extracted for every date.

Module code:

```
1   # Data frame extract and concatenate module
2
3   import extract_function as ex
4   import concatenate_df as co
5   #import numpy as np
6   #import turbulence_parameters as tu
7   #import time_process as tp
8
9   # Data gathering Function
10  # The function data_gathering outputs a pandas data frame containing the
        variables of interest of a series of consecutive dates defined by the user
        for 1 sonic in 1 tower
11
12  def data_gathering (dates_def, z1, z2, x, y):
13      '''Takes dates to retrieve data (np array of int), start date (int), end
            date (int), tower code name (str), sonic height(int), Returns data frame
            with sonic data for that period (pandas DataFrame)'''
14      n = 0
```

```
    m = 0

    if z1 == z2:

        dfc = ex.extract(z1, x, y, n)

    else:
        for z in dates_def:
            if m == 1:
                df = ex.extract(z, x, y, n)
                dfc = co.concatenate(dfc, df)

            if m == 0:

                dfc = ex.extract(z, x, y, n)
                m = 1
            n +=1
    return dfc
```

B.1.11 Extract function

This module contains a function that directly extracts data from the .nc files for one sonic anemometer, in the defined dates.

It also contains functions that calculate parameters using variables directly extracted from the .nc files:

- `extract (fi, x, y, n):`

  This function opens the .nc file corresponding to the selected date, extracts the target variables, stores them in a data frame (Pandas DataFrame) and then returns the data frame.

- `extract_2 (fi, x, y):`

  This function opens the .nc file corresponding to the selected date, extracts the size of the time variable, creates an array with the size of the time variable containing index numbers starting in zero (numpy array of int) and then returns the array.

- `extract_start_time (z1, k, h):`

  This function opens the .nc file corresponding to the selected date, extracts the data, uses the time variable, the selected hour and the period conversion constant to define the start period time data (str) and then returns it.

- `extract_end_time (z2, k, h):`

  This function opens the .nc file corresponding to the selected date, extracts the data, uses the time variable, the selected hour and the period conversion constant to define the end period time data (str) and then returns it.

Module code:

```
from netCDF4 import Dataset
```

```
import numpy as np
import pandas as pd
import time

# Extracting from netcdf into a pandas data frame
# Input args: fi - day code, x - tower code, y - sonic height, n - number of the day in the time series requested (if you want 2 consecutive days, in the first day n=0 and in the second day n=1)
# Output: df - dataframe containing the variables of interest for that day, tower and sonic

def extract (fi, x, y, n):
    '''Takes fi - day code (int), x - tower code name (str), y - sonic height (int) and n - number that increments by 1 for consecutive days starting in zero (int), Returns data frame with the extracted variables (pandas DataFrame)'''
    # n is used to garantee continuous data in the time stamp variable 'time'
    data = Dataset("{}.nc".format(fi) , 'r')

    if x == 'tnw12' or x == 'tnw13' or x == 'tnw14' or x == 'tnw15' or x == 'tnw16':

        if y == 30:

            y = 28

    # Storing the netCDF data into variables

    u = data.variables['u_{}m_{}'.format(y, x)]
    v = data.variables['v_{}m_{}'.format(y, x)]
    w = data.variables['w_{}m_{}'.format(y, x)]

    uu = data.variables['u_u__{}m_{}'.format(y, x)]
    uv = data.variables['u_v__{}m_{}'.format(y, x)]
    uw = data.variables['u_w__{}m_{}'.format(y, x)]
    vv = data.variables['v_v__{}m_{}'.format(y, x)]
    vw = data.variables['v_w__{}m_{}'.format(y, x)]
    ww = data.variables['w_w__{}m_{}'.format(y, x)]

    direc = data.variables['dir_{}m_{}'.format(y, x)]
    spd = data.variables['spd_{}m_{}'.format(y, x)]

    basetime = data.variables['base_time']
    reltime = data.variables['time']

    # Creating an empty pandas dataframe
    starting_time = data.variables['time'].units[14:29]+ '2:30'
    ending_time = data.variables['time'].units[14:25]+ '23:57:30'
    time_range = pd.date_range(start= starting_time, end= ending_time, periods= 288)

    df = pd.DataFrame(0, columns= ['basetime', 'time', 'u','v', 'w', 'vh', 'dir','uu', 'vv', 'ww', 'uv', 'uw', 'vw'], index = time_range)
```

```
46
47      # Create a numpy array with the size of the time variable
48
49      dt = np.arange(0, data.variables['time'].size)
50
51      # Filling the empty pandas data frame with the values of the variables for
            each time value
52
53      for time_index in dt:
54          df.iloc[time_index] = basetime[time_index], reltime[time_index] +
                86400*n, u[time_index], v[time_index], w[time_index],
                spd[time_index], direc[time_index], uu[time_index], vv[time_index],
                ww[time_index], uv[time_index], uw[time_index], vw[time_index]
55
56      return df
57
58
59  #extract_2(fi, x, y)
60
61  def extract_2 (fi, x, y):
62      '''Takes day code (int), tower code name (str), sonic height (int), Returns
            array with the size of the time variable and containing index numbers
            starting in zero (numpy array of int)'''
63      data = Dataset("{}.nc".format(fi) , 'r')
64      dt = np.arange(0, data.variables['time'].size)
65      return dt
66
67
68  #extract_start_time(z1, k)
69
70  def extract_start_time (z1, k, h):
71      '''Takes day (int), period conversion constant (int) and start hour (int),
            Returns first period time data (str)'''
72      data = Dataset("{}.nc".format(z1) , 'r')
73
74      st = 150*k + 3600*h
75      stf = time.strftime('%H:%M:%S', time.gmtime(st))
76      start = data.variables['time'].units[14:25]+ stf
77      return start
78
79  #extract_end_time(z2, k)
80
81  def extract_end_time (z2, k, h):
82      '''Takes day (int), period conversion constant (int) and start hour (int),
            Returns last period time data (str)'''
83      data = Dataset("{}.nc".format(z2) , 'r')
84
85      en = 86400 - (150*k + (24-h)*3600)
86      e = time.strftime('%H:%M:%S', time.gmtime(en))
87      end = data.variables['time'].units[14:25]+ e
88      return end
```

B.1.12 Concatenate function

This module only contains `concatenate`, a function that appends one data frame to another:

- `concatenate (df1, df2):`

  This function takes two data frames, concatenates them and returns the result (pandas DataFrame).

Module code:

```
# Concatenate 2 data frames of 1 sonic

def concatenate (df1, df2):
    '''Takes two data frames (pandas DataFrame), Returns concatenated data frame
       built from the args (pandas DataFrame)'''
    frames = [df1]
    frames.append(df2)
    result = pd.concat(frames)

    return result
```

B.1.13 Turbulence parameters

This module contains a function that calculates the three turbulence parameters (`tke, ti, tih`) and appends them to the data frame from where those parameters where calculated, for 5-min averaged data.

This module also contains functions that calculates the variances (`uu, vv, ww`) and three turbulence parameters (`tke, ti, tih`) for data different than 5-min averaged data (e.g. 20 min averaged data). There's also a function that appends these parameters to the corresponding data frame.

Functions:

- `turbulence_5min (dfc):`

  Takes the data frame containing the variables retrieved from the sonic (pandas Data Frame), creates three new columns with calculated values of turbulence kinetic energy, turbulence intensity and turbulence intensity from horizontal components for data of 5 min period and appends it to the same data frame, which is returned (pandas Data Frame).

- `tke_k (dfc, k):`

  This function uses the data frame containing the variables retrieved from the sonic (pandas Data Frame) and the time period conversion value (int) to calculate the values of turbulence kinetic energy for the desired time periods and stores them in an array which is returned (np array of float).

- `ti_k (dfc, k):`

  This function uses the data frame containing the variables retrieved from the sonic

(pandas Data Frame) and the time period conversion value (int) to calculate the values of turbulence intensity for the desired time periods and stores them in an array which is returned (np array of float).

- `tih_k (dfc, k):`

  This function uses the data frame containing the variables retrieved from the sonic (pandas Data Frame) and the time period conversion value (int) to calculate the values of turbulence intensity from horizontal components for the desired time periods and stores them in an array which is returned (np array of float).

- `turb_append(dfc, tke_arr, ti_arr, tih_arr):`

  This function takes the time period adjusted data frame and appends it with the three turbulence parameters arrays for the same time period. The resulting data frame is returned (pandas Data Frame).

Module code:

```
# Turbulence Module - contains functions to calc turbulence parameters
# For time periods different than 5 min the turbulence parameters calculated
    should be reviewed

import numpy as np

# Adding turbulence parameters to the original Data Frame

def turbulence_5min (dfc):
    '''Takes the data frame containing the variables retrieved from the sonic
       (pandas Data Frame), Returns the same data frame with three new columns
       with calculated values of turbulence kinetic energy, turbulence
       intensity and turbulence intensity from horizontal components for data
       of 5 min period (pandas Data Frame)'''
    dt = np.arange(0, dfc.size/13, dtype=int)

    i = 0
    for a in dt:

        if i == 0:

            u = dfc.iat[a,2]
            v = dfc.iat[a,3]
            w = dfc.iat[a,4]

            uu = dfc.iat[a,7]
            vv = dfc.iat[a,8]
            ww = dfc.iat[a,9]

            tke = (1/2)*(uu+vv+ww)
            ti = np.sqrt(tke*(2/3))/np.sqrt(u*u+v*v+w*w)
            tih = np.sqrt((1/3)*(uu+vv))/np.sqrt(u*u+v*v)

            tke_arr = np.array(tke, dtype='float')
            ti_arr = np.array(ti, dtype='float')
```

```
            tih_arr = np.array(tih, dtype='float')

            i = 1

        else:
            u = dfc.iat[a,2]
            v = dfc.iat[a,3]
            w = dfc.iat[a,4]

            uu = dfc.iat[a,7]
            vv = dfc.iat[a,8]
            ww = dfc.iat[a,9]

            tke = (1/2)*(uu+vv+ww)
            ti = np.sqrt(tke*(2/3))/np.sqrt(u*u+v*v+w*w)
            tih = np.sqrt((1/3)*(uu+vv))/np.sqrt(u*u+v*v)


            temp = np.array(tke, dtype='float')
            temp2 = np.array(ti, dtype='float')
            temp3 = np.array(tih, dtype='float')

            tke_arr = np.append(tke_arr, [temp])
            ti_arr = np.append(ti_arr, [temp2])
            tih_arr = np.append(tih_arr, [temp3])

    dfc['tke'] = tke_arr.tolist()
    dfc['ti'] = ti_arr.tolist()
    dfc['tih'] = tih_arr.tolist()

    return dfc


# Creation of arrays with variances and covariances for time periods different
    from 5 min

def var_k (dfc, k):
    '''Takes the data frame containing the variables retrieved from the sonic
        (pandas Data Frame) and time period conversion value (int), Returns
        arrays containing the values of variances for the desired time periods
        (np array of float)'''
    dt = np.arange(0, dfc.size/13, dtype=int)

    i = 0
    j = 0

    for a in dt:

        u = dfc.iat[a,2]
        v = dfc.iat[a,3]
        w = dfc.iat[a,4]

        uu = dfc.iat[a,7]
```

```
        vv = dfc.iat[a,8]
        ww = dfc.iat[a,9]

        if i == 0:
            u_m1 = u*u
            uu_m2 = uu
            u_m3 = u

            v_m1 = v*v
            vv_m2 = vv
            v_m3 = v

            w_m1 = w*w
            ww_m2 = ww
            w_m3 = w

        if i > 0 and i < k:
            u_m1 = u_m1 + u*u
            uu_m2 = uu_m2*uu
            u_m3 = u_m3 + u

            v_m1 = v_m1 + v*v
            vv_m2 = vv_m2*vv
            v_m3 = v_m3 + v

            w_m1 = w_m1 + w*w
            ww_m2 = ww_m2*ww
            w_m3 = w_m3 + w

        i += 1

        if i == k:
            u_var = (u_m1/k)+(uu_m2/k)-(u_m3/k)*(u_m3/k)
            v_var = (v_m1/k)+(vv_m2/k)-(v_m3/k)*(v_m3/k)
            w_var = (w_m1/k)+(ww_m2/k)-(w_m3/k)*(w_m3/k)

            if j == 0:
                uu_arr = np.array(u_var, dtype='float')
                vv_arr = np.array(v_var, dtype='float')
                ww_arr = np.array(w_var, dtype='float')

            if j > 0:

                temp_uu = np.array(u_var, dtype='float')
                temp_vv = np.array(v_var, dtype='float')
                temp_ww = np.array(w_var, dtype='float')

                uu_arr = np.append(uu_arr, [temp_uu])
                vv_arr = np.append(vv_arr, [temp_vv])
                ww_arr = np.append(ww_arr, [temp_ww])

            j += 1
```

```
133             i = 0
134
135     return uu_arr, vv_arr, ww_arr
136
137
138 # Creation of array with turbulence kinetic energy for time periods different
        from 5 min
139
140 def tke_k (dfc, k):
141     '''Takes the data frame containing the variables retrieved from the sonic
            (pandas Data Frame) and time period conversion value (int), Returns
            array containing the values of turbulence kinetic energy for the desired
            time periods (np array of float)'''
142     dt = np.arange(0, dfc.size/13, dtype=int)
143
144     i = 0
145     j = 0
146
147     for a in dt:
148
149         u = dfc.iat[a,2]
150         v = dfc.iat[a,3]
151         w = dfc.iat[a,4]
152
153         uu = dfc.iat[a,7]
154         vv = dfc.iat[a,8]
155         ww = dfc.iat[a,9]
156
157         if i == 0:
158             u_m1 = u*u
159             uu_m2 = uu
160             u_m3 = u
161
162             v_m1 = v*v
163             vv_m2 = vv
164             v_m3 = v
165
166             w_m1 = w*w
167             ww_m2 = ww
168             w_m3 = w
169
170         if i > 0 and i < k:
171             u_m1 = u_m1 + u*u
172             uu_m2 = uu_m2*uu
173             u_m3 = u_m3 + u
174
175             v_m1 = v_m1 + v*v
176             vv_m2 = vv_m2*vv
177             v_m3 = v_m3 + v
178
179             w_m1 = w_m1 + w*w
180             ww_m2 = ww_m2*ww
181             w_m3 = w_m3 + w
```

```
182
183         i += 1
184
185         if i == k:
186             u_var = (u_m1/k)+(uu_m2/k)-(u_m3/k)*(u_m3/k)
187             v_var = (v_m1/k)+(vv_m2/k)-(v_m3/k)*(v_m3/k)
188             w_var = (w_m1/k)+(ww_m2/k)-(w_m3/k)*(w_m3/k)
189
190             tke = (1/2)*(u_var+v_var+w_var)
191
192             if j == 0:
193                 tke_arr = np.array(tke, dtype='float')
194
195             if j > 0:
196
197                 temp = np.array(tke, dtype='float')
198
199                 tke_arr = np.append(tke_arr, [temp])
200
201             j += 1
202
203             i = 0
204
205     return tke_arr
206
207
208 # Creation of array with turbulence intensity for time periods different from 5
        min
209
210 def ti_k (dfc, k):
211     '''Takes the data frame containing the variables retrieved from the sonic
            (pandas Data Frame) and time period conversion value (int), Returns
            array containing the values of turbulence intensity for the desired time
            periods (np array of float)'''
212     dt = np.arange(0, dfc.size/13, dtype=int)
213
214     i = 0
215     j = 0
216
217     for a in dt:
218
219         u = dfc.iat[a,2]
220         v = dfc.iat[a,3]
221         w = dfc.iat[a,4]
222
223         uu = dfc.iat[a,7]
224         vv = dfc.iat[a,8]
225         ww = dfc.iat[a,9]
226
227         if i == 0:
228             u_m1 = u*u
229             uu_m2 = uu
230             u_m3 = u
```

```
231
232             v_m1 = v*v
233             vv_m2 = vv
234             v_m3 = v
235
236             w_m1 = w*w
237             ww_m2 = ww
238             w_m3 = w
239
240         if i > 0 and i < k:
241             u_m1 = u_m1 + u*u
242             uu_m2 = uu_m2*uu
243             u_m3 = u_m3 + u
244
245             v_m1 = v_m1 + v*v
246             vv_m2 = vv_m2*vv
247             v_m3 = v_m3 + v
248
249             w_m1 = w_m1 + w*w
250             ww_m2 = ww_m2*ww
251             w_m3 = w_m3 + w
252
253         i += 1
254
255         if i == k:
256             u_var = (u_m1/k)+(uu_m2/k)-(u_m3/k)*(u_m3/k)
257             v_var = (v_m1/k)+(vv_m2/k)-(v_m3/k)*(v_m3/k)
258             w_var = (w_m1/k)+(ww_m2/k)-(w_m3/k)*(w_m3/k)
259
260             tke = (1/2)*(u_var+v_var+w_var)
261             ti =
                    np.sqrt(tke*(2/3))/(np.sqrt((u_m3/k)*(u_m3/k)+(v_m3/k)*(v_m3/k)+(w_m3/k)*(w_m3/k)))
262
263             if j == 0:
264                 ti_arr = np.array(ti, dtype='float')
265
266
267             if j > 0:
268
269                 temp = np.array(ti, dtype='float')
270
271                 ti_arr = np.append(ti_arr, [temp])
272             j += 1
273
274             i = 0
275
276     return ti_arr
277
278
279 # Creation of array with turbulence intensity from horizontal components for
      time periods different from 5 min
280
281 def tih_k (dfc, k):
```

```
282     '''Takes the data frame containing the variables retrieved from the sonic
            (pandas Data Frame) and time period conversion value (int), Returns
            array containing the values of turbulence intensity from horizontal
            components for the desired time periods (np array of float)'''
283     dt = np.arange(0, dfc.size/13, dtype=int)
284
285     i = 0
286     j = 0
287
288     for a in dt:
289
290         u = dfc.iat[a,2]
291         v = dfc.iat[a,3]
292
293         uu = dfc.iat[a,7]
294         vv = dfc.iat[a,8]
295
296         if i == 0:
297             u_m1 = u*u
298             uu_m2 = uu
299             u_m3 = u
300
301             v_m1 = v*v
302             vv_m2 = vv
303             v_m3 = v
304
305         if i > 0 and i < k:
306             u_m1 = u_m1 + u*u
307             uu_m2 = uu_m2*uu
308             u_m3 = u_m3 + u
309
310             v_m1 = v_m1 + v*v
311             vv_m2 = vv_m2*vv
312             v_m3 = v_m3 + v
313
314         i += 1
315
316         if i == k:
317             u_var = (u_m1/k)+(uu_m2/k)-(u_m3/k)*(u_m3/k)
318             v_var = (v_m1/k)+(vv_m2/k)-(v_m3/k)*(v_m3/k)
319
320             tih =
                    np.sqrt((u_var+v_var)/3)/(np.sqrt((u_m3/k)*(u_m3/k)+(v_m3/k)*(v_m3/k)))
321
322             if j == 0:
323                 tih_arr = np.array(tih, dtype='float')
324
325             if j > 0:
326
327                 temp = np.array(tih, dtype='float')
328
329                 tih_arr = np.append(tih_arr, [temp])
330             j += 1
```

```
331
332             i = 0
333
334     return tih_arr
335
336
337 # Append variance arrays to the new data frame with the time period adjusted
338
339 def var_append(dfc, uu, vv, ww):
340     '''Takes time period adjusted data frame (pandas Data Frame), three arrays
            containing variances time period adjusted parameteres(np arrays of
            float), Returns complete data frame (pandas Data Frame)'''
341     dfc['uu'] = uu.tolist()
342     dfc['vv'] = vv.tolist()
343     dfc['ww'] = ww.tolist()
344
345     return dfc
346
347
348 # Append variance arrays to the new data frame with the time period adjusted
349
350 def covar_append(dfc, uv, uw, vw):
351     '''Takes time period adjusted data frame (pandas Data Frame), three arrays
            containing covariances time period adjusted parameteres(np arrays of
            float), Returns complete data frame (pandas Data Frame)'''
352     dfc['uv'] = uv.tolist()
353     dfc['uw'] = uw.tolist()
354     dfc['vw'] = vw.tolist()
355
356     return dfc
357
358
359 # Append turbulence arrays to the new data frame with the time period adjusted
360
361 def turb_append(dfc, tke_arr, ti_arr, tih_arr):
362     '''Takes time period adjusted data frame (pandas Data Frame), three arrays
            containing turbulence time period adjusted parameteres(np arrays of
            float), Returns complete data frame (pandas Data Frame)'''
363     dfc['tke'] = tke_arr.tolist()
364     dfc['ti'] = ti_arr.tolist()
365     dfc['tih'] = tih_arr.tolist()
366
367     return dfc
```

B.1.14 Time process

This module contains functions that perform time sectioning, period adjustments and temporal re-sampling to existing data frames:

- `period_adjust (z1, z2, dfc, k)`:

  This function takes a 5-min averaged data frame and creates a new data frame,

with a different time period, from the data of the first. The new data frame is then returned (pandas DataFrame).

- `section_time_sm2 (dfc, z1, z2, z3, z4, k, dates_def):`

  This function takes a time indexed data frame (pandas Data Frame), sections the data frame according to the start and end hour for every day and returns the new data frame (pandas DataFrame).

- `section_time_sm3 (dfc, z1, z2, z3, z4, k):`

  This function takes a time indexed data frame (pandas Data Frame), sections the data frame according to the start hour for the first day and end hour for the last day and returns the new data frame (pandas DataFrame).

Module code:

```
# Time process functions

import pandas as pd
import numpy as np
import extract_function as ex
import concatenate_df as co

# - Period adjust function

def period_adjust (z1, z2, dfc, k):
    '''Takes start date (int), end date (int), 5 min period data frame (pandas
       Data Frame) and time period conversion value (int), Returns period
       adjusted data frame (pandas Data Frame)'''
    n = z2 - z1 +1
    t = (288*n)/k
    h = 0

    starting_time = ex.extract_start_time(z1, k, h)

    ending_time = ex.extract_end_time(z2, k, h)

    time_range = pd.date_range(start= starting_time, end= ending_time, periods=
        t)
    dfmean = pd.DataFrame(0, columns= ['basetime', 'time', 'u','v', 'w', 'vh',
        'dir','uu', 'vv', 'ww', 'uv', 'uw', 'vw'], index = time_range)
    dt = np.arange(0, 288*n)

    i = 0
    j = 0
    b_mean = 0
    rel_time_m = 0
    u_m = 0
    v_m = 0
    w_m = 0
    vh_m = 0
    dir_m = 0
    uu_m = 0
```

```
34      vv_m = 0
35      ww_m = 0
36      uv_m = 0
37      uw_m = 0
38      vw_m = 0
39
40      for a in dt:
41
42          if i < k:
43
44              basetime = dfc.iat[a, 0]
45              b_mean = b_mean + basetime
46
47              reltime = dfc.iat[a, 1]
48              rel_time_m = rel_time_m + reltime
49
50              u = dfc.iat[a, 2]
51              u_m = u_m + u
52
53              v = dfc.iat[a, 3]
54              v_m = v_m + v
55
56              w = dfc.iat[a, 4]
57              w_m = w_m + w
58
59              vh = dfc.iat[a, 4]
60              vh_m = vh_m + vh
61
62              di = dfc.iat[a, 6]
63              dir_m = dir_m + di
64
65              uu = dfc.iat[a, 7]
66              uu_m = uu_m + uu
67
68              vv = dfc.iat[a, 8]
69              vv_m = vv_m + vv
70
71              ww = dfc.iat[a, 9]
72              ww_m = ww_m + ww
73
74              uv = dfc.iat[a, 10]
75              uv_m = uv_m + uv
76
77              uw = dfc.iat[a, 11]
78              uw_m = uw_m + uw
79
80              vw = dfc.iat[a, 12]
81              vw_m = vw_m + vw
82
83          if i == (k-1):
84
85              dfmean.iloc[j] = b_mean/k, rel_time_m/k, u_m/k, v_m/k, w_m/k, vh_m/k,
                    dir_m/k, uu_m/k, vv_m/k, ww_m/k, uv_m/k, uw_m/k, vw_m/k
```

```
86
87          i += 1
88
89          if i == k:
90
91              i = 0
92
93              b_mean = 0
94              rel_time_m = 0
95              u_m = 0
96              v_m = 0
97              w_m = 0
98              vh_m = 0
99              dir_m = 0
100             uu_m = 0
101             vv_m = 0
102             ww_m = 0
103             uv_m = 0
104             uw_m = 0
105             vw_m = 0
106
107             j += 1
108
109     return dfmean
110
111
112 # Function to section the data frame according to the start and end hour of mode
        sm = 2
113
114 def section_time_sm2 (dfc, z1, z2, z3, z4, k, dates_def):
115     '''Takes time indexed data frame (pandas Data Frame), start date (int), end
            date (int), start hour for every day (int), end hour for every day
            (int), time period conversion value (int) and array with the selected
            dates (np array of int), Returns sectioned data frame (pandas Data
            Frame)'''
116     n = 0
117
118     if z1 == z2:
119
120         n0 = z2-z1
121
122         a = int(((12*z3)/k))
123         b = int(((12*z4)/k)+(288/k)*n0)
124
125         df = dfc.iloc[a:b]
126
127     else:
128         for a in dates_def:
129
130             a1 = int(((12*z3)/k)+(288/k)*n)
131             b1 = int(((12*z4)/k)+(288/k)*n)
132
133             df = dfc.iloc[a1:b1]
```

```
134
135             if n == 0:
136
137                 df1 = df.copy()
138
139             else:
140
141                 df = co.concatenate(df1, df)
142                 df1 = df.copy()
143
144             n += 1
145
146     return df
147
148
149 # Function to section the data frame according to the start and end hour of mode
        sm = 3
150
151 def section_time_sm3 (dfc, z1, z2, z3, z4, k):
152     '''Takes time indexed data frame (pandas Data Frame), start date (int), end
            date (int), start hour (int), end hour (int) and time period conversion
            value (int), Returns sectioned data frame (pandas Data Frame)'''
153     n0 = z2-z1
154
155     a = int(((12*z3)/k))
156     b = int(((12*z4)/k)+(288/k)*n0)
157
158     df = dfc.iloc[a:b]
159
160     return df
```

B.1.15 Save export function

This module contains a function that saves and exports into easily readable files the existing data frames:

- `save_export (dfc, x, y, z1, z2, z3, z4, k):`

  This function takes a data frame (pandas DataFrame), uses the time and sonic anemometer related variables (x, y, z1, z2, z3, z4, k) to create the name of the files, saves and exports the data frame to a data folder where the app is located. Three file types are created, '.csv', '.xls' and '.txt.', returns nothing.

Module code:

```
1 # Save and export module
2
3 import numpy as np
4 import os.path
5
6 def save_export (dfc, x, y, z1, z2, z3, z4, k):
7     '''Takes time indexed data frame (pandas Data Frame), tower code name (str),
            sonic height (int), start date (int), end date (int), start hour (int),
```

```
        end hour (int) and time period conversion value (int), Returns none,
        saves and exports files'''
    p = k*5

    direc = r'C:\Users\Baba\Desktop\João\Tese\Python\Teste_1\Perdigao_python\App
        2.0\data_pf'

    if z1 == z2:
        if z3 == 0 and z4 == 24:
            dfc.to_csv(os.path.join(r'{}'.format(direc),'{}m_{}_{}_P-{}min.csv'.format(y,x,z1,p)))

            dfc.to_excel(os.path.join(r'{}'.format(direc),'{}m_{}_{}_P-{}min.xls'.format(y,x,z1,p))

            df_np = dfc.to_numpy()
            np.savetxt(os.path.join(r'{}'.format(direc),"{}m_{}_{}_P-{}min.txt".format(y,x,z1,p)),
                df_np, fmt = "%.4f")
        else:
            dfc.to_csv(os.path.join(r'{}'.format(direc),'{}m_{}_{}_{}-{}h_P-{}min.csv'.format(y,x,z

            dfc.to_excel(os.path.join(r'{}'.format(direc),'{}m_{}_{}_{}-{}h_P-{}min.xls'.format(y,x

            df_np = dfc.to_numpy()
            np.savetxt(os.path.join(r'{}'.format(direc),"{}m_{}_{}_{}-{}h_P-{}min.txt".format(y,x,z
                df_np, fmt = "%.4f")

    else:
        if z3 == 0 and z4 == 24:
            dfc.to_csv(os.path.join(r'{}'.format(direc),'{}m_{}_{}_{}_P-{}min.csv'.format(y,x,z1,z2

            dfc.to_excel(os.path.join(r'{}'.format(direc),'{}m_{}_{}_{}_P-{}min.xls'.format(y,x,z1,

            df_np = dfc.to_numpy()
            np.savetxt(os.path.join(r'{}'.format(direc),"{}m_{}_{}_{}_P-{}min.txt".format(y,x,z1,z2
                df_np, fmt = "%.4f")
        else:
            dfc.to_csv(os.path.join(r'{}'.format(direc),'{}m_{}_{}_{}_{}-{}h_P-{}min.csv'.format(y,

            dfc.to_excel(os.path.join(r'{}'.format(direc),'{}m_{}_{}_{}_{}-{}h_P-{}min.xls'.format(

            df_np = dfc.to_numpy()
            np.savetxt(os.path.join(r'{}'.format(direc),"{}m_{}_{}_{}_{}-{}h_P-{}min.txt".format(y,
                df_np, fmt = "%.4f")
```

B.2 Availability Map

Instructions for re-use:

Availability map from scratch:

- A function is called to create the first column of the availability map. The column index is the date of the first day of the Perdigão campaign ('29/11/2016') and the rows index is a list with the sonic anemometers full name ('<tower>_<height>m');

- The .nc file for the first day is downloaded, all sonic anemometers are looped and an availability function is called for each one of them;
- The resulting data frame is stored in a file. That file will be used in an iterative process where the data frame will be filled with values for the rest of the days;
- The user defines the next set of consecutive dates in the iterative process;
- The .nc files for the dates defined by the user are downloaded;
- The file that should contain the availability data for all the dates previous to the dates defined by the user is opened and it's contents are stored in a data frame;
- all sonic anemometers are looped and an availability function is called for each one of them;
- The availability data for each day it's appended to the existing data frame;
- The resulting data frame contains the previous values and the recently calculated values for data availability. That data frame is then stored and exported to overwrite the same file;
- The process is repeated until the last day of the campaign ('01/07/2017').

Variable filters:

An averaged 5-min period is considered valid if:

Table B.5: Variable filters

Variable	Description
`u` - wind component	-30 $m\,s^{-1}$ > u < 30 $m\,s^{-1}$
`v` - wind component	-30 $m\,s^{-1}$ > v < 30 $m\,s^{-1}$
`w` - wind component	-30 $m\,s^{-1}$ > w < 30 $m\,s^{-1}$
`vh` - wind speed	-30 $m\,s^{-1}$ > vh < 30 $m\,s^{-1}$
`dir` - direction	0 ° > dir < 360 °
`uu` - wind variance	$-50\,m^2\,s^{-2}$ > uu < $50\,m^2\,s^{-2}$
`vv` - wind variance	$-50\,m^2\,s^{-2}$ > vv < $50\,m^2\,s^{-2}$
`ww` - wind variance	$-50\,m^2\,s^{-2}$ > ww < $50\,m^2\,s^{-2}$
`uv` - wind covariance	$-50\,m^2\,s^{-2}$ > uv < $50\,m^2\,s^{-2}$
`uw` - wind covariance	$-50\,m^2\,s^{-2}$ > uw < $50\,m^2\,s^{-2}$
`vw` - wind covariance	$-50\,m^2\,s^{-2}$ > vw < $50\,m^2\,s^{-2}$

Functions:

- `wind_speed_count (var)`:

  This function loops through an array that contains a wind speed variable ('u', 'v', 'w' or 'vh') and checks if each value is located inside the designated interval (in this case from -30 to 30 [m/s]). If the criteria is accomplished, the count variable is incremented. This filter allows to detect if the data has missing values or aberrant values. The count variable is returned.

- `variance_count (var)`:

  This function loops through an array that contains a wind speed variance (or covariance) variable ('uu', 'vv', 'ww', 'uv', 'uw' or 'vw') and checks if each value is located inside the designated interval (in this case from -50 to 50 [m^2/s^2]). If the criteria is accomplished, the count variable is incremented. This filter allows to detect if the data has missing values or aberrant values. The count variable is returned.

- `availability_one_sonic(fi, x, y)`:

  For the day and sonic anemometer defined by the arguments, this function extracts the data, calls the count functions for all wind speed and variances variables, counts the data availability for the direction variable (using the interval [0 : 360º]), uses all of the counts variables to determine the value of the data availability flag variable and then returns it (int)

  This function loops through an array that contains a wind speed variance (or covariance) variable ('uu', 'vv', 'ww', 'uv', 'uw' or 'vw') and checks if each value is located inside the designated interval (in this case from -50 to 50 [m/s]). If the criteria is accomplished, the count variable is incremented. This filter allows to detect if the data has missing values or aberrant values. The count variable is returned.

- `availability_towers_arr (fi)`:

  This function takes the day code, downloads the corresponding '.nc' file, browses the variables in the file and returns an array containing the names of the available towers for that file (np array of str).

- `name_arr ()`:

  Using functions from the tower location module this function creates an array containing all the sonic anemometers full names and returns it (np array of str).

- `avail_map_creation ()`:

  This function loops through all the sonic anemometers and calls the data availability function for each one in the date of the first day of the Perdigão campaign ('29/11/2016'). The resulting array is used as the first column of the data frame (pandas DataFrame) that will be used for filling to obtain the availability map. The data frame is exported to a file ("Availability_map.pkl") that will be used for filling of the map.

- `avail_map_filling (dates_def, df)`:

  This function downloads the '.nc' files corresponding to the defined dates, loops through every defined date and every sonic anemometer and calls the data availability function for each one in all defined dates. The resulting arrays are appended to the data frame taken as argument. The new data frame is then exported to overwrite the ("`Availability_map.pkl`"), which will be used to retrieve the data frame that will be used as argument the next time this function is called.

Module code:

```
# Mapping of data availability

from netCDF4 import Dataset
import extract_function as ex
import download as dl
import numpy as np
import pandas as pd
import tower_location as tl
import dates_array as da

# This Function is used to count how many times the wind speed variable passed as an argument (var) has values between the defined interval

def wind_speed_count (var):
    '''Takes wind speed variable (np array of float), Returns data count (int)'''
    c = 0
    for a3 in var:
        if a3 > 30 or a3 < -30:
            continue
        if a3 < 30 and a3 > -30:
            c = c+1

    global count
    count = c
    return count

# This Function is used to count how many times the variance variable passed as an argument (var) has values between the defined interval

def variance_count (var):
    '''Takes wind speed variance (np array of float), Returns data count (int)'''
    c = 0
    for a3 in var:
        if a3 > 50 or a3 < -50:
            continue
        if a3 < 50 and a3 > -50:
            c = c+1

    global count
    count = c
    return count

# This function loops through every time period and every variable to check the data availability for the selected sonic (x, y)
# Outputs a value that represents the data availabilty for the specified sonic in the specified day (fi)

def availability_one_sonic(fi, x, y):
    '''Takes day code (int), tower code name (str) and sonic height (int),
        Returns data availability flag variable (int)'''
    n = 0
    m = 3
```

```
48
49      try:
50
51          dfc = ex.extract(fi, x, y, n)
52
53          u = dfc[["u"]].to_numpy()
54          v = dfc[["v"]].to_numpy()
55          w = dfc[["w"]].to_numpy()
56          vh = dfc[["vh"]].to_numpy()
57          dire = dfc[["dir"]].to_numpy()
58          uu = dfc[["uu"]].to_numpy()
59          vv = dfc[["vv"]].to_numpy()
60          ww = dfc[["ww"]].to_numpy()
61          uv = dfc[["uv"]].to_numpy()
62          uw = dfc[["uw"]].to_numpy()
63          vw = dfc[["vw"]].to_numpy()
64
65
66          wind_speed_count(u)
67          uc = count
68
69          wind_speed_count(v)
70          vc = count
71
72          wind_speed_count(w)
73          wc = count
74
75          wind_speed_count(vh)
76          vhc = count
77
78          dirc = 0
79          for a3 in dire:
80              if a3 > 360 or a3 < 0:
81                  continue
82              if a3 < 360 and a3 > 0:
83                  dirc = dirc+1
84
85          variance_count(uu)
86          uuc = count
87
88          variance_count(vv)
89          vvc = count
90
91          variance_count(ww)
92          wwc = count
93
94          variance_count(uv)
95          uvc = count
96
97          variance_count(uw)
98          uwc = count
99
100         variance_count(vw)
```

```
101         vwc = count
102
103     except KeyError:
104         m = 0
105
106     except:
107         m = 3
108
109     else:
110         if uc == 288 and vc == 288 and wc == 288 and vhc == 288 and dirc == 288
                and uuc == 288 and vvc == 288 and wwc == 288 and uvc == 288 and uwc
                == 288 and vwc == 288:
111             m = 1
112
113         if uc == 0 and vc == 0 and wc == 0 and vhc == 0 and dirc == 0 and uuc ==
                0 and vvc == 0 and wwc == 0 and uvc == 0 and uwc == 0 and vwc == 0:
114             m = 0
115         if (uc < 288 or vc < 288 or wc < 288 or vhc < 288 or dirc < 288 or uuc <
                288 or vvc < 288 or wwc < 288 or uvc < 288 or uwc < 288 or vwc < 288)
                and (uc > 0 and vc > 0 and wc > 0 and vhc > 0 and dirc > 0 and uuc >
                0 and vvc > 0 and wwc > 0 and uvc > 0 and uwc > 0 and vwc > 0):
116             m = 2
117
118         if m != 0 and m != 1 and m != 2:
119             m = 3
120
121     return m
122
123 # This function outputs an array with the towers that have available data for
        that day
124
125 def availability_towers_arr (fi):
126     '''Takes day code (int), Returns array with available towers for that day
            (np array of str)'''
127     data = Dataset("{}.nc".format(fi) , 'r')
128
129     sites = data.variables['sites']
130
131     #print(sites)
132     #print(data.variables.keys())
133
134     t_name = tl.towers_name()
135
136     i=0
137     for a in data.variables.keys():
138
139         b = str(a)
140
141         if b[0:3] == 'lat' or b[0:3] == 'lon':
142             continue
143
144         if i == 0:
145
```

```
146             for a2 in t_name:
147                 if b[-5:] == a2:
148                     arr = np.array(a2)
149                     i = 1
150                     break
151 
152                 if b[-3:] == a2:
153                     arr = np.array(a2)
154                     i = 1
155                     break
156 
157         if i == 1:
158 
159             for a2 in t_name:
160                 if b[-5:] == a2:
161                     arr = np.append(arr, [a2])
162                 if b[-3:] == a2:
163                     arr = np.append(arr, [a2])
164 
165     arr = np.unique(arr)
166     global ar
167     ar = arr.copy()
168     return ar
169 
170 # This function outputs an array with all the names of the sonics and respective
      tower in the format: ex.: 'tnw01_2m'
171 
172 def name_arr ():
173 
174     name_array = np.array(["tnw01_2m"])
175 
176     t_name = tl.towers_name()
177 
178     for x in t_name:
179 
180         j = np.where(t_name == x)
181         j = j[0]
182 
183         hei = tl.sonics_available_name(j)
184 
185         for y in hei:
186 
187             name = "{}_{}m".format(x, y)
188             name_array = np.append(name_array, [name])
189 
190     name_array = name_array[1:]
191     global nam
192     nam = name_array.copy()
193     return nam
194 
195 # This function creates the data frame of the availability map with only the
      first column, if the '20161129' file is not present in the directory, the
      first line must come out of comment
```

```
196
197 def avail_map_creation ():
198
199     '''
200     dl.download(dates_def)
201     '''
202     fi = '20161129'
203
204     #Creating day array for the info for each sonic per day
205     #Creating of name_array which will be the index of the data frame, with the
        code name of each sonic
206
207     day_arr = np.zeros(183)
208
209     name_arr()
210     name_array = nam
211
212     i = 0
213     for a in name_array:
214
215         if i < 164:
216
217             x_a = a[0:5]
218             x = str(x_a)
219
220             a_s = len(a)
221
222             if a_s == 8:
223
224                 y_a = a[-2:-1]
225                 y = str(y_a)
226
227             if a_s == 9:
228
229                 y_a = a[-3:-1]
230                 y = str(y_a)
231
232             if a_s == 10:
233
234                 y_a = a[-4:-1]
235                 y = str(y_a)
236
237
238             day_arr[i] = availability_one_sonic(fi, x, y)
239             i +=1
240             continue
241
242         if i >= 164:
243
244             x_a = a[0:3]
245             x = str(x_a)
246
247             a_s = len(a)
```

```
248
249             if a_s == 6:
250
251                 y_a = a[-2:-1]
252                 y = str(y_a)
253
254             if a_s == 7:
255
256                 y_a = a[-3:-1]
257                 y = str(y_a)
258
259
260
261             day_arr[i] = availability_one_sonic(fi, x, y)
262             i +=1
263
264     # Create the data frame
265
266     df = pd.DataFrame(data=day_arr, index=name_array, columns=["20161129"])
267
268     # Saving the data frame into an external file
269
270     df.to_pickle("Availability_map2.pkl")
271
272 # This function fills the availability map, using the dates_def input, the
      function creates new columns for data availability in the defined dates, and
      appends them to the existing data frame (df)
273 # Outputs a new availabilty map with data for more days (dates_def)
274
275 def avail_map_filling (dates_def, df):
276     '''Takes dates defined (np array of int) and data frame (pandas DataFrame),
          Returns none'''
277     dl.download(dates_def)
278
279     #Creating of name_array which will be the index of the data frame, with the
          code name of each sonic
280     name_arr()
281     name_array = nam
282
283     for fi in dates_def:
284
285         #Creating day array for the info for each sonic per day
286
287         day_arr = np.zeros(183)
288
289         i = 0
290         for a in name_array:
291
292             if i < 164:
293
294                 x_a = a[0:5]
295                 x = str(x_a)
296
```

```
297                 a_s = len(a)
298
299                 if a_s == 8:
300
301                     y_a = a[-2:-1]
302                     y = str(y_a)
303
304                 if a_s == 9:
305
306                     y_a = a[-3:-1]
307                     y = str(y_a)
308
309                 if a_s == 10:
310
311                     y_a = a[-4:-1]
312                     y = str(y_a)
313
314                 day_arr[i] = availability_one_sonic(fi, x, y)
315                 i +=1
316                 continue
317
318             if i >= 164:
319
320                 x_a = a[0:3]
321                 x = str(x_a)
322
323                 a_s = len(a)
324
325                 if a_s == 6:
326
327                     y_a = a[-2:-1]
328                     y = str(y_a)
329
330                 if a_s == 7:
331
332                     y_a = a[-3:-1]
333                     y = str(y_a)
334
335
336                 day_arr[i] = availability_one_sonic(fi, x, y)
337                 i +=1
338
339             df['{}'.format(fi)] = day_arr.tolist()
340
341
342
343     # Saving the data frame into an external file
344
345     df.to_pickle("Availability_map2.pkl")
346
347
348 # To create a new availability map, call the avail_map_creation function and
      make sure the '20161129' file is present in the directory, if not, check the
```

```
    avail_map_creation function and take the first line out of comment
349
350 avail_map_creation()
351
352 # Code lines for filling the map, in regular PC's it is recommended to set
    dates_def to 10 days sets, it takes time to create the full map and it is
    safer that way
353
354 dates_tot = da.dates_tot_function()
355
356 # Change the interval for each iteration
357
358 dates_def = dates_tot[2:10]
359
360 df = pd.read_pickle("Availability_map2.pkl")
361
362 avail_map_filling(dates_def, df)
```

B.3 Data Comparison Tools

B.3.1 Graph ereation environment

Module code:

```
1  # Graph Creation environment
2
3  import graph_functions as gf
4  import numpy as np
5  import pandas as pd
6  import tower_location as tl
7
8  def rmse_bias (hour_period,va,sta,he,nvars,heights,rmse,bias,df_rmse_bias):
9
10     i = df.columns.get_loc(v_name)
11     j = df_ventos.columns.get_loc(v_name)
12
13
14     exp = pf_2d_array[:, i]
15     sim = data[:, j]
16
17     if v_name == 'dir':
18
19         for k in range(nhour):
20
21             exp = pf_2d_array[k*hour_period*12:k*hour_period*12+12*hour_period,i]
22             sim = data[k*hour_period*12:k*hour_period*12+12*hour_period,j]
23
24             err[k,va,he], mse[k,va,he], rmse[k,va,he], bias[k,va,he] =
                   gf.rmse_f(exp, sim, True)
25
26             #if height == heights[-1]:
27
```

```
28                #mean_e[k,va], mean_s[k,va] = gf.mean_vars(exp, sim)
29
30        va +=1
31
32    else:
33
34        for k in range(nhour):
35
36            exp = pf_2d_array[k*hour_period*12:k*hour_period*12+12*hour_period,i]
37            sim = data[k*hour_period*12:k*hour_period*12+12*hour_period,j]
38
39            err[k,va,he], mse[k,va,he], rmse[k,va,he], bias[k,va,he] =
                  gf.rmse_f(exp, sim)
40
41            #if height == heights[-1]:
42
43                #mean_e[k,va], mean_s[k,va] = gf.mean_vars(exp, sim)
44
45        va +=1
46
47    if hour_period == 6:
48
49        pass
50
51    return rmse, bias, va, he, df_rmse_bias
52
53 # Changeable parameters
54
55 plot_type = 1
56 table_rmse = 0
57 table_bias = 0
58
59 rmse_var = 0
60 bias_var = 0
61
62 towers = np.array(["tnw01", "tnw09", "tnw13", "rsw01", "rsw02", "v03"])#,
       "tnw09", "tnw10"])#"tnw01", "tnw02", "tnw03", "tnw04", "tnw05"])#, "tnw06",
       "tnw07", "tnw08", "tnw09", "tnw10"])#
63 #heights_def = np.array([2])
64 he_def = 0
65
66 highlight = 1
67
68 hour_period = 1
69
70 index_df = np.arange(185)
71 df_rmse_bias = pd.DataFrame(0, columns= ['sonic', 'rmse_24h',
       'bias_24h','rmse_0-6h', 'bias_0-6h', 'rmse_6-12h',
       'bias_6-12h','rmse_12-18h', 'bias_12-18h', 'rmse_18-24h', 'bias_18-24h'],
       index=index_df)
72
73 if towers.size == 1:
74
```

```
towers = towers[0]
j = tl.tower_index_pos(towers)

if he_def == 1:

    heights = heights_def

else:

    heights = tl.sonics_available_name(j)

direc_pf =
    r'C:\Users\Baba\Desktop\João\Tese\Python\Teste_1\Perdigao_python\App
    2.0\data_pf'
path_ventos_sonics_coord =
    r'C:\Users\Baba\Desktop\João\Tese\Python\Teste_1\Perdigao_python\App
    2.0\sonics_coord_est_nor_z.csv'
direc_sim = r'C:\Users\Baba\Desktop\João\Tese'

'''-------------'''

if plot_type == 1:

    a_name = "vh"
    b_name = "dir"
    c_name = "tke"

if plot_type == 2:

    a_name = "u"
    b_name = "v"
    c_name = "w"

if plot_type == 3:

    a_name = "uu"
    b_name = "vv"
    c_name = "ww"

if plot_type == 4:

    a_name = "vh"
    b_name = "dir"
    c_name = "tke"

# This last option does not plot anything, just produces RMSE's tables

if plot_type == 5:

    a_name = "vh"
    b_name = "dir"
    c_name = "tke"
    table_rmse = 1
```

```
124
125
126     nvars = np.array(a_name)
127     nvars = np.append(a_name, [b_name, c_name])
128
129     sta = 0
130     he = 0
131
132     if heights.size == 1:
133
134         height = heights[0]
135
136         ''' Perdigao File '''
137
138         path_pf =
                r'{}\{}m_{}_20170514_20170515_0-18h_P-5min.csv'.format(direc_pf,height,towers)
139
140         df, pf_2d_array, pf_num_rows = gf.df_perdigao_file(path_pf)
141
142         ''' VENTOS File V2'''
143
144         df_ventos, data =
                gf.df_ventos_file(towers,height,direc_sim,path_ventos_sonics_coord)
145
146         va = 0
147
148         for v_name in nvars:
149
150             nh = int(pf_num_rows/hour_period)
151
152             if sta == 0:
153
154                 nhour = int(nh/12)
155                 var = len(nvars)
156                 h = len(heights)
157
158                 err = np.zeros((nhour,var,h))
159                 mse = np.zeros((nhour,var,h))
160                 rmse = np.zeros((nhour,var,h))
161                 bias = np.zeros((nhour,var,h))
162                 r = np.zeros((nhour,var,h))
163                 ss = np.zeros((nhour,var,h))
164                 mean_e = np.zeros((nhour,var))
165                 mean_s = np.zeros((nhour,var))
166                 sta = 1
167
168             rmse, bias ,va ,he ,df_rmse_bias = rmse_bias(hour_period, va, sta,
                    he, nvars, heights,rmse,bias,df_rmse_bias)
169
170         he +=1
171
172
173
```

```
174     else:
175
176         for height in heights:
177
178             ''' Perdigao File '''
179
180             path_pf =
                    r'{}\{}m_{}_20170514_20170515_0-18h_P-5min.csv'.format(direc_pf,height,towers)
181
182             df, pf_2d_array, pf_num_rows = gf.df_perdigao_file(path_pf)
183
184             ''' VENTOS File V2'''
185
186             df_ventos, data =
                    gf.df_ventos_file(towers,height,direc_sim,path_ventos_sonics_coord)
187
188             va = 0
189
190             for v_name in nvars:
191
192                 nh = int(pf_num_rows/hour_period)
193
194                 if sta == 0:
195
196                     nhour = int(nh/12)
197                     var = len(nvars)
198                     h = len(heights)
199
200                     err = np.zeros((nhour,var,h))
201                     mse = np.zeros((nhour,var,h))
202                     rmse = np.zeros((nhour,var,h))
203                     bias = np.zeros((nhour,var,h))
204                     r = np.zeros((nhour,var,h))
205                     ss = np.zeros((nhour,var,h))
206                     mean_e = np.zeros((nhour,var))
207                     mean_s = np.zeros((nhour,var))
208                     sta = 1
209
210                 rmse, bias ,va ,he ,df_rmse_bias = rmse_bias(hour_period, va,
                        sta, he, nvars, heights,rmse,bias,df_rmse_bias)
211
212             he +=1
213             continue
214
215
216
217         rmse_min = np.zeros((nhour,var))
218         rmse_max = np.zeros((nhour,var))
219
220
221         for va in range(var):
222
223             for k in range(nhour):
```

```
224
225                 a = 0
226                 for he in range(h):
227
228                     rm = rmse[k,va,he]
229
230                     if a == 0:
231                         rmse_min[k,va] = rm
232                         rmse_max[k,va] = rm
233                         a = 1
234
235                     if rm < rmse_min[k,va]:
236                         rmse_min[k,va] = rm
237
238                     if rm > rmse_max[k,va]:
239                         rmse_max[k,va] = rm
240
241
242
243     '''-------------'''
244
245     # Plot
246
247     if heights.size == 1:
248
249         height = heights[0]
250
251         ''' Perdigao File '''
252
253         path_pf =
                r'{}\{}m_{}_20170514_20170515_0-18h_P-5min.csv'.format(direc_pf,height,towers)
254
255         df, pf_2d_array, pf_num_rows = gf.df_perdigao_file(path_pf)
256
257         ''' VENTOS File V2'''
258
259         df_ventos, data =
                gf.df_ventos_file(towers,height,direc_sim,path_ventos_sonics_coord)
260
261         if plot_type == 1:
262
263             gf.plot_v_dir_t(path_pf, pf_2d_array, a_name, b_name, c_name, df,
                    df_ventos, data, highlight)
264
265         if plot_type == 2:
266
267             gf.plot_uvw(path_pf,pf_2d_array,df,df_ventos,data,highlight)
268
269         if plot_type == 3:
270
271             gf.plot_variances(path_pf,pf_2d_array,df,df_ventos,data,highlight)
272
273         if plot_type == 4:
```

```
274
275             gf.plot_rmse_v_dir_t(path_pf,pf_2d_array,nvars,df,df_ventos,rmse_min,rmse_max,nhour,dat
276
277
278     else:
279
280         for height in heights:
281
282             ''' Perdigao File '''
283
284             path_pf =
                    r'{}\{}m_{}_20170514_20170515_0-18h_P-5min.csv'.format(direc_pf,height,towers)
285
286             df, pf_2d_array, pf_num_rows = gf.df_perdigao_file(path_pf)
287
288             ''' VENTOS File V2'''
289
290             df_ventos, data =
                    gf.df_ventos_file(towers,height,direc_sim,path_ventos_sonics_coord)
291
292             if plot_type == 1:
293
294                 gf.plot_v_dir_t(path_pf, pf_2d_array, a_name, b_name, c_name, df,
                        df_ventos, data, highlight)
295
296             if plot_type == 2:
297
298                 gf.plot_uvw(path_pf,pf_2d_array,df,df_ventos,data,highlight)
299
300             if plot_type == 3:
301
302                 gf.plot_variances(path_pf,pf_2d_array,df,df_ventos,data,highlight)
303
304             if plot_type == 4:
305
306                 gf.plot_rmse_v_dir_t(path_pf,pf_2d_array,nvars,df,df_ventos,rmse_min,rmse_max,nhour
307
308
309     # Tables
310
311     if table_rmse == 1:
312
313         df_rmse, rmse_v = gf.tab_rmse_v(rmse, rmse_var,nhour,heights)
314         gf.save_rmse_table(df_rmse,path_pf,rmse_var,a_name,b_name,c_name,towers)
315
316     if table_bias == 1:
317
318         df_bias, bias_v = gf.tab_bias_v(bias, bias_var,nhour,heights)
319         gf.save_rmse_table(df_bias,path_pf,bias_var,a_name,b_name,c_name,towers)
320
321
322
323 else:
```

```
324
325     for tower_p in towers:
326
327         j = tl.tower_index_pos(tower_p)
328
329         if he_def == 1:
330
331             heights = heights_def
332
333         else:
334
335             heights = tl.sonics_available_name(j)
336
337         direc_pf =
                r'C:\Users\Baba\Desktop\João\Tese\Python\Teste_1\Perdigao_python\App
                2.0\data_pf'
338         path_ventos_sonics_coord =
                r'C:\Users\Baba\Desktop\João\Tese\Python\Teste_1\Perdigao_python\App
                2.0\sonics_coord_est_nor_z.csv'
339         direc_sim = r'C:\Users\Baba\Desktop\João\Tese'
340
341         '''_____________'''
342
343         if plot_type == 1:
344
345             a_name = "vh"
346             b_name = "dir"
347             c_name = "tke"
348
349         if plot_type == 2:
350
351             a_name = "u"
352             b_name = "v"
353             c_name = "w"
354
355         if plot_type == 3:
356
357             a_name = "uu"
358             b_name = "vv"
359             c_name = "ww"
360
361         if plot_type == 4:
362
363             a_name = "vh"
364             b_name = "dir"
365             c_name = "tke"
366
367         # This last option does not plot anything, just produces RMSE's tables
368
369         if plot_type == 5:
370
371             a_name = "vh"
372             b_name = "dir"
```

```
373             c_name = "tke"
374             table_rmse = 1
375 
376 
377         nvars = np.array(a_name)
378         nvars = np.append(a_name, [b_name, c_name])
379 
380         sta = 0
381         he = 0
382 
383         if heights.size == 1:
384 
385             height = heights[0]
386 
387             ''' Perdigao File '''
388 
389             path_pf =
                    r'{}\{}m_{}_20170514_20170515_0-18h_P-5min.csv'.format(direc_pf,height,towers)
390 
391             df, pf_2d_array, pf_num_rows = gf.df_perdigao_file(path_pf)
392 
393             ''' VENTOS File V2'''
394 
395             df_ventos, data =
                    gf.df_ventos_file(towers,height,direc_sim,path_ventos_sonics_coord)
396 
397             va = 0
398 
399             for v_name in nvars:
400 
401                 nh = int(pf_num_rows/hour_period)
402 
403                 if sta == 0:
404 
405                     nhour = int(nh/12)
406                     var = len(nvars)
407                     h = len(heights)
408 
409                     err = np.zeros((nhour,var,h))
410                     mse = np.zeros((nhour,var,h))
411                     rmse = np.zeros((nhour,var,h))
412                     bias = np.zeros((nhour,var,h))
413                     r = np.zeros((nhour,var,h))
414                     ss = np.zeros((nhour,var,h))
415                     mean_e = np.zeros((nhour,var))
416                     mean_s = np.zeros((nhour,var))
417                     sta = 1
418 
419                 rmse, bias ,va ,he ,df_rmse_bias = rmse_bias(hour_period, va,
                        sta, he, nvars, heights,rmse,bias,df_rmse_bias)
420 
421             he +=1
422 
```

```
423
424
425         else:
426
427             for height in heights:
428
429                 ''' Perdigao File '''
430
431                 path_pf =
                        r'{}\{}m_{}_20170514_20170515_0-18h_P-5min.csv'.format(direc_pf,height,towers)
432
433                 df, pf_2d_array, pf_num_rows = gf.df_perdigao_file(path_pf)
434
435                 ''' VENTOS File V2'''
436
437                 df_ventos, data =
                        gf.df_ventos_file(towers,height,direc_sim,path_ventos_sonics_coord)
438
439                 va = 0
440
441                 for v_name in nvars:
442
443                     nh = int(pf_num_rows/hour_period)
444
445                     if sta == 0:
446
447                         nhour = int(nh/12)
448                         var = len(nvars)
449                         h = len(heights)
450
451                         err = np.zeros((nhour,var,h))
452                         mse = np.zeros((nhour,var,h))
453                         rmse = np.zeros((nhour,var,h))
454                         bias = np.zeros((nhour,var,h))
455                         r = np.zeros((nhour,var,h))
456                         ss = np.zeros((nhour,var,h))
457                         mean_e = np.zeros((nhour,var))
458                         mean_s = np.zeros((nhour,var))
459                         sta = 1
460
461                     rmse, bias ,va ,he ,df_rmse_bias = rmse_bias(hour_period, va,
                            sta, he, nvars, heights,rmse,bias,df_rmse_bias)
462
463                 he +=1
464                 continue
465
466
467
468             rmse_min = np.zeros((nhour,var))
469             rmse_max = np.zeros((nhour,var))
470
471
472             for va in range(var):
```

```
473
474                 for k in range(nhour):
475
476                     a = 0
477                     for he in range(h):
478
479                         rm = rmse[k,va,he]
480
481                         if a == 0:
482                             rmse_min[k,va] = rm
483                             rmse_max[k,va] = rm
484                             a = 1
485
486                         if rm < rmse_min[k,va]:
487                             rmse_min[k,va] = rm
488
489                         if rm > rmse_max[k,va]:
490                             rmse_max[k,va] = rm
491
492
493
494         '''-------------'''
495
496         # Plot
497
498         if heights.size == 1:
499
500             height = heights[0]
501
502             ''' Perdigao File '''
503
504             path_pf =
                    r'{}\{}m_{}_20170514_20170515_0-18h_P-5min.csv'.format(direc_pf,height,towers)
505
506             df, pf_2d_array, pf_num_rows = gf.df_perdigao_file(path_pf)
507
508             ''' VENTOS File V2'''
509
510             df_ventos, data =
                    gf.df_ventos_file(towers,height,direc_sim,path_ventos_sonics_coord)
511
512             if plot_type == 1:
513
514                 gf.plot_v_dir_t(path_pf, pf_2d_array, a_name, b_name, c_name, df,
                        df_ventos, data, highlight)
515
516             if plot_type == 2:
517
518                 gf.plot_uvw(path_pf,pf_2d_array,df,df_ventos,data,highlight)
519
520             if plot_type == 3:
521
522                 gf.plot_variances(path_pf,pf_2d_array,df,df_ventos,data,highlight)
```

```
523
524             if plot_type == 4:
525
526                 gf.plot_rmse_v_dir_t(path_pf,pf_2d_array,nvars,df,df_ventos,rmse_min,rmse_max,nhour
527
528
529         else:
530
531             for height in heights:
532
533                 ''' Perdigao File '''
534
535                 path_pf =
                        r'{}\{}m_{}_20170514_20170515_0-18h_P-5min.csv'.format(direc_pf,height,towers)
536
537                 df, pf_2d_array, pf_num_rows = gf.df_perdigao_file(path_pf)
538
539                 ''' VENTOS File V2'''
540
541                 df_ventos, data =
                        gf.df_ventos_file(towers,height,direc_sim,path_ventos_sonics_coord)
542
543                 if plot_type == 1:
544
545                     gf.plot_v_dir_t(path_pf, pf_2d_array, a_name, b_name, c_name,
                            df, df_ventos, data, highlight)
546
547                 if plot_type == 2:
548
549                     gf.plot_uvw(path_pf,pf_2d_array,df,df_ventos,data,highlight)
550
551                 if plot_type == 3:
552
553                     gf.plot_variances(path_pf,pf_2d_array,df,df_ventos,data,highlight)
554
555                 if plot_type == 4:
556
557                     gf.plot_rmse_v_dir_t(path_pf,pf_2d_array,nvars,df,df_ventos,rmse_min,rmse_max,nh
558
559
560         # Tables
561
562         if table_rmse == 1:
563
564             df_rmse, rmse_v = gf.tab_rmse_v(rmse, rmse_var,nhour,heights)
565             gf.save_rmse_table(df_rmse,path_pf,rmse_var,a_name,b_name,c_name,towers)
566
567         if table_bias == 1:
568
569             df_bias, bias_v = gf.tab_bias_v(bias, bias_var,nhour,heights)
570             gf.save_rmse_table(df_bias,path_pf,bias_var,a_name,b_name,c_name,towers)
```

B.3.2 Graph functions

Functions from `graph_functions` module:

- `towers_name ()`:

  This function creates an array with all the available tower names (np array of string) and returns it.

- `sonics_height_array ()`:

  This function creates an array with all the available heights (np array of string) and returns it.

- `tower_title(path)`:

  This function uses the path (url) of the file used to data plotting to find the tower name (str), which is then returned.

- `sonic_title(path)`:

  This function uses the path (url) of the file used to data plotting to find the sonic height name (str), which is then returned.

- `turbulence_5min (dfc)`:

  Takes the data frame containing the variables retrieved from the sonic (pandas Data Frame), creates three new columns with calculated values of turbulence kinetic energy, turbulence intensity and turbulence intensity from horizontal components for data of 5 min period and appends it to the same data frame, which is returned (pandas Data Frame).

  This function is designed for VENTOS®/M files only.

- `rmse_f(exp, sim, azimuth=False)`:

  This function takes two arrays with the same size containing values for any variable from both Perdigão and VENTOS®/M files. Calculates errors, bias and RMSE between measurements and simulation data. Uses azimuth option for Wind Direction error.

- `mean_vars(exp,sim)`:

  This function takes the argument arrays and calculates the mean value of each one. Both mean values (float) are returned.

- `y_labels(var)`:

  This function uses the variable name (str) to return the appropriate variable label (str) to use in the graphs creation.

- `y_labels_rmse(var)`:

  This function uses the variable name (str) to return the appropriate variable label (str) to use in the graphs creation for the rmse graph type.

- `plot_rmse_v_dir_t(path_pf,pf_2d_array,nvars,df,df_ventos,rmse_min,`

`rmse_max,nhour,data,highlight=0)`:

This function is used to plot a graph that shows the hourly RMSE variation for each variable for a selected tower.

The graph contains two lines and a shaded area between them. One of the lines represents the highest value of RMSE of all sonic anemometers in that tower for each hour and the other one represents the lowest value.

The function contains plot parameterization that build the graph in a certain style.

- `plot_uvw(path_pf,pf_2d_array,df,df_ventos,data,highlight=0)`:

  This function is used to plot a graph that shows data for each Wind Component (u,v,w) of selected sonic anemometer.

  The graph contains two lines. One of the lines represents the values of Perdigão data and the other one represents the values from VENTOS®/M data.

  The function contains plot parameterization that build the graph in a certain style.

- `plot_v_dir_t(path_pf,pf_2d_array,a_name,b_name,c_name,df,df_ventos, data,highlight=0)`:

  This function is used to plot a graph that shows data for Wind Speed, Direction and Turbulence Parameter of selected sonic anemometer.

  The graph contains two lines. One of the lines represents the values of Perdigão data and the other one represents the values from VENTOS®/M data.

  The function contains plot parameterization that build the graph in a certain style.

- `plot_variances(path_pf,pf_2d_array,df,df_ventos,data,highlight=0)`:

  This function is used to plot a graph that shows data for each Wind Variances (uu,vv,ww) of selected sonic anemometer.

  The graph contains two lines. One of the lines represents the values of Perdigão data and the other one represents the values from VENTOS®/M data.

  The function contains plot parameterization that build the graph in a certain style.

- `ventos_file_name_v3(dfv2_sonic,tower_p,height,direc)`: path (str)

  This function uses a data frame containing all the sonic anemometers coordinates data and the selected anemometer defining variables (tower, height) to produce the path to the VENTOS®/M file containing the data to the selected sonic anemometer, which is returned (str).

- `df_perdigao_file(path_pf)`:

  This function opens the selected Perdigão file into a data frame (pandas DataFrame), converts the time column to have a similar index to the VENTOS®/M files (in the Perdigão files the time index represents the time in the middle of the 5-min period, in the VENTOS®/M files the time index represents the time in the end of the 5-min period). After the processing the resulting data frame is converted into a 2D

array (np array of float). The data frame, np array and the number of rows of the data are then returned.

- `df_ventos_file(tower_p,height,direc_v,path_v)`:

  This function opens the file containing all the sonic anemometers coordinates data, calls the `ventos_file_name_v3(dfv2_sonic,tower_p,height,direc)` function to obtain the path to the VENTOS®/M file that contains data for the selected sonic anemometer. Opens that VENTOS®/M file into a 2D array (numpy array of float) and converts the 2D array into a data frame. That data frame is used to call the `turbulence_5min (dfc)` function that calculates the three turbulence parameters (tke, ti, tih) to the VENTOS®/M data. The resulting data frame is then converted to a 2D array (np array of float) and both the data frame, np array and the number of rows of the data are returned.

- `tab_rmse_v(rmse,var,nhour,heights)`:

  This function takes the hourly RMSE calculated data, selects the data for the chosen variable and returns a data frame (pandas DataFrame) and 2D array (2D np array) containing hourly RMSE data, for the last 24h for the selected variable, in the selected tower.

- `save_rmse_table(df_rmse,path_pf,rmse_var,a_name,b_name,c_name,tower_p)`:

  This function saves and exports the hourly RMSE data into three file formats ('.csv', '.xls' and '.txt'), it uses the arguments to define the name of the exported files.

- `tab_bias_v(bias,var,nhour,heights)`:

  This function takes the hourly Bias calculated data, selects the data for the chosen variable and returns a data frame (pandas DataFrame) and 2D array (2D np array) containing hourly Bias data, for the last 24h for the selected variable, in the selected tower.

- `save_bias_table(df_bias,path_pf,bias_var,a_name,b_name,c_name,tower_p)`:

  This function saves and exports the hourly Bias data into three file formats ('.csv', '.xls' and '.txt'), it uses the arguments to define the name of the exported files.

Module code:

```
# Graph Functions

import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from matplotlib.ticker import (MultipleLocator, FormatStrFormatter,
    AutoMinorLocator)
import os.path

direc_graphs =
    r'C:\Users\Baba\Desktop\João\Tese\Python\Teste_1\Perdigao_python\App
    2.0\graphs'

```

```
11 # Array containing all towers names
12
13 def towers_name ():
14     '''Takes none, Returns array with towers code name (np array of string)'''
15     t_name = np.array(["tnw01", "tnw02", "tnw03", "tnw04", "tnw05", "tnw06",
           "tnw07", "tnw08", "tnw09", "tnw10", "tnw11", "tnw12", "tnw13", "tnw14",
           "tnw15", "tnw16", "tse01", "tse02", "tse04", "tse05", "tse06", "tse07",
           "tse08", "tse09", "tse10", "tse11", "tse12", "tse13", "rsw01", "rsw02",
           "rsw03", "rsw04", "rsw05", "rsw06", "rsw07", "rsw08", "rne01", "rne02",
           "rne03", "rne04", "rne06", "rne07", "v01", "v03", "v04", "v05", "v06",
           "v07", "Extreme_SW", "Extreme_NE"])
16     return t_name
17
18
19 # Array containing all sonics heights
20
21 def sonics_height_array ():
22     '''Takes none, Returns array with sonics heights (np array of string)'''
23     # 2m, 4m, 6m, 8m, 10m, 12m, 20m, 30m, 40m, 60m, 80m, 100,
24
25     height = np.array(["2", "4", "6", "8", "10", "12", "20", "30", "40", "60",
           "80", "100"])
26
27     return height
28
29
30 # Sonic and tower info
31
32 def tower_title(path):
33     '''Takes path (url), Returns tower name (str)'''
34     t_name = towers_name()
35     for a in t_name:
36         if a in path:
37             t_title = a
38     return t_title
39
40
41 def sonic_title(path):
42     '''Takes path (url), Returns sonic height name (str)'''
43     heights = sonics_height_array()
44     for b in heights:
45         b = b +'m'
46         if b in path:
47             h_title = b
48     return h_title
49
50
51 # TKE recalculated for VENTOS File
52
53 def turbulence_5min_ventos (dfc):
54     '''Takes the data frame containing the variables retrieved from the sonic
           (pandas Data Frame), Returns the same data frame with three new columns
           with calculated values of turbulence kinetic energy, turbulence
```

```
        intensity and turbulence intensity from horizontal components for data
        of 5 min period (pandas Data Frame)'''
    dt = np.arange(0, dfc.size/16, dtype=int)

    i = 0
    for a in dt:

        if i == 0:

            u = dfc.iat[a,3]
            v = dfc.iat[a,4]
            w = dfc.iat[a,5]

            uu = dfc.iat[a,13]
            vv = dfc.iat[a,14]
            ww = dfc.iat[a,15]

            tke = (1/2)*(uu+vv+ww)
            ti = np.sqrt(tke*(2/3))/np.sqrt(u*u+v*v+w*w)
            tih = np.sqrt((1/3)*(uu+vv))/np.sqrt(u*u+v*v)

            tke_arr = np.array(tke, dtype='float')
            ti_arr = np.array(ti, dtype='float')
            tih_arr = np.array(tih, dtype='float')

            i = 1

        else:
            u = dfc.iat[a,3]
            v = dfc.iat[a,4]
            w = dfc.iat[a,5]

            uu = dfc.iat[a,13]
            vv = dfc.iat[a,14]
            ww = dfc.iat[a,15]

            tke = (1/2)*(uu+vv+ww)
            ti = np.sqrt(tke*(2/3))/np.sqrt(u*u+v*v+w*w)
            tih = np.sqrt((1/3)*(uu+vv))/np.sqrt(u*u+v*v)

            temp = np.array(tke, dtype='float')
            temp2 = np.array(ti, dtype='float')
            temp3 = np.array(tih, dtype='float')

            tke_arr = np.append(tke_arr, [temp])
            ti_arr = np.append(ti_arr, [temp2])
            tih_arr = np.append(tih_arr, [temp3])

    dfc['tke'] = tke_arr.tolist()
    dfc['ti'] = ti_arr.tolist()
    dfc['tih'] = tih_arr.tolist()

    return dfc
```

```
106
107
108 # Error indicator functions:
109
110 def rmse_f(exp, sim, azimuth=False):
111
112     ''' Calculate errors, bias and RMSE between measurements and forecast
113         data '''
114     ''' Use azimuth option for Wind Direction error '''
115
116     delta = sim - exp
117     if azimuth:
118         # Check if it works.
119         signm = np.reshape(np.concatenate((np.zeros(np.shape(delta)),
120                                            np.sign(np.abs(delta) - 180.))),
121                            (2, np.size(delta)))
122         # print(signm)
123         sign = np.max(signm, axis=0)
124         # print(sign)
125         abs_err = delta - 360. * np.sign(delta) * sign
126         # print(abs_err)
127     else:
128         abs_err = delta
129
130     # Error (prof., check for azimuth)
131     err = np.sum(np.abs(abs_err)/exp)/np.size(sim)
132     # Mean square error
133     mse = (1/np.size(sim)) * np.sum(abs_err**2)
134     # RMSE
135     rmse = np.sqrt(mse)
136     # Bias
137     bias = (1/np.size(sim))*np.sum(abs_err)
138
139     return err, mse, rmse, bias
140
141
142 # Mean Vars
143
144 def mean_vars(exp,sim):
145     '''Takes exp (array), sim (array), Returns mean_e, mean_s, mean values of
            both arrays (float)'''
146     mean_e = sum(exp)/np.size(exp)
147     mean_s = sum(sim)/np.size(sim)
148     return mean_e,mean_s
149
150
151 # Labels
152
153 def y_labels(var):
154     '''Takes variable name (str), Returns appropriate variable label (str)'''
155     if var == 'u':
156         label = '$U$ (m s$^{-1}$)'
157     if var == 'v':
```

```
158         label = '$V$ (m s$^{-1}$)'
159     if var == 'w':
160         label = '$W$ (m s$^{-1}$)'
161     if var == 'vh':
162         label = 'Wind Speed\n(m s$^{-1}$)'
163     if var == 'dir':
164         label = 'Direction ($\degree$)'
165     if var == 'uu':
166         label = '$U$ $Var$\n(m$^2$ s$^{-2}$)'
167     if var == 'vv':
168         label = '$V$ $Var$\n(m$^2$ s$^{-2}$)'
169     if var == 'ww':
170         label = '$W$ $Var$\n(m$^2$ s$^{-2}$)'
171     if var == 'tke':
172         label = 'TKE (m$^2$ s$^{-2}$)'
173     if var == 'ti':
174         label = 'TI'
175     if var == 'tih':
176         label = 'TIH'
177     return label
178
179
180 def y_labels_rmse(var):
181     '''Takes variable name (str), Returns appropriate variable label (str)'''
182     if var == 'u':
183         label = 'RMSE\n$U$ (m s$^{-1}$)'
184     if var == 'v':
185         label = 'RMSE\n$V$ (m s$^{-1}$)'
186     if var == 'w':
187         label = 'RMSE\n$W$ (m s$^{-1}$)'
188     if var == 'vh':
189         label = 'RMSE Wind\nSpeed (m s$^{-1}$)'
190     if var == 'dir':
191         label = 'RMSE\nDirection ($\degree$)'
192     if var == 'uu':
193         label = 'RMSE $U$ $Var$ (m$^2$ s$^{-2}$)'
194     if var == 'vv':
195         label = 'RMSE $V$ $Var$ (m$^2$ s$^{-2}$)'
196     if var == 'ww':
197         label = 'RMSE $W$ $Var$ (m$^2$ s$^{-2}$)'
198     if var == 'tke':
199         label = 'RMSE TKE\n(m$^2$ s$^{-2}$)'
200     if var == 'ti':
201         label = 'RMSE TI'
202     if var == 'tih':
203         label = 'RMSE TIH'
204     return label
205
206
207 # Plot rmse
208
209 def
    plot_rmse_v_dir_t(path_pf,pf_2d_array,nvars,df,df_ventos,rmse_min,rmse_max,nhour,data,highligh
```

```
210     '''Takes path of Perdigão file (str), data from Perdigão file (2D np array
            of float), array containing variables names (array of str), data frame
            with Perdigão data (pandas DataFrame), data frame with VENTOS®/M data
            (pandas DataFrame), array with max RMSE per tower for all variables (np
            array of float), array with min RMSE per tower for all variables (np
            array of float), number of hourly periods to be plotted (int), data from
            VENTOS®/M file (2D np array of float), flag variable to learn if the
            shaded area of the last 24h is to be displayed or not, Returns none'''
211     a_name = nvars[0]
212     b_name = nvars[1]
213     c_name = nvars[2]
214
215     a1 = df.columns.get_loc(a_name)
216     b1 = df.columns.get_loc(b_name)
217     c1 = df.columns.get_loc(c_name)
218     a2 = df_ventos.columns.get_loc(a_name)
219     b2 = df_ventos.columns.get_loc(b_name)
220     c2 = df_ventos.columns.get_loc(c_name)
221
222     # Plot formatting:
223     plt.rcParams['font.family'] = 'sans-serif'
224     #plt.rcParams.update({'font.sans-serif':'Helvetica'})
225     SMALL_SIZE = 10
226     MEDIUM_SIZE = 12
227     ANOT_SIZE = SMALL_SIZE - 2
228     NUMBER_SIZE = SMALL_SIZE
229     plt.rc('font', size=SMALL_SIZE)     # controls default text sizes
230     plt.rc('axes', titlesize=SMALL_SIZE) # fontsize of the axes title
231     plt.rc('axes', labelsize=16) # fontsize of the x and y labels
232     plt.rc('xtick', labelsize=SMALL_SIZE) # fontsize of the tick labels
233     plt.rc('ytick', labelsize=SMALL_SIZE) # fontsize of the tick labels
234     plt.rc('legend', fontsize=SMALL_SIZE) # legend fontsize
235     shadecolor = 'lavenderblush'
236
237     #x = range(nhour)
238     x = np.arange(0.5, nhour+0.5)
239     xlim_l = int(x[0])
240     xlim_r = int(x[-1])
241
242     #
243
244     fig = plt.figure() # an empty figure with no axes
245
246     resolution_value = 200
247
248     plt.rcParams["figure.figsize"] = (12,6)
249
250     plt.rcParams['figure.dpi'] = resolution_value
251
252     t_title = tower_title(path_pf)
253     #h_title = sonic_title(path_pf)
254
255     fig, ax_lst = plt.subplots(3, 1) # a figure with a 2x2 grid of Axes
```

```
256
257     fig.suptitle("Tower {} RMSE's".format(t_title), fontsize=18)
258
259     #subtitle = 'Wind Speed Components U,V,W'
260
261
262     # A
263
264     a_ylabel = y_labels_rmse(a_name)
265
266     axa = plt.subplot(311)
267
268     axa.set_ylabel(r'{}'.format(a_ylabel))
269     #axa.set_ylim(0, 8.5)
270     axa.set_xlim(xlim_l, xlim_r)
271     axa.xaxis.set_tick_params(labelbottom=False)
272     axa.grid(color='k', linestyle='-', linewidth=0.7, alpha=0.2)
273     axa.yaxis.set_major_locator(MultipleLocator(2))
274     axa.yaxis.set_minor_locator(MultipleLocator(0.4))
275     axa.tick_params(which='major', length=7, width=1, direction='in', top=True,
            right=True)
276     axa.tick_params(which='minor', length=3, width=1, direction='in', top=True,
            right=True)
277
278     #x = pf_2d_array[:,1]
279     y1 = rmse_max[:,0]
280     y2 = rmse_min[:,0]
281     #y7 = mean_e[:,0]
282     #y8 = mean_s[:,0]
283     #plt.plot(x, y7, 'teal', x, y8, 'orangered')
284     plt.plot(x, y1, 'b')
285     plt.plot(x, y2, 'b')
286
287     axa.fill_between(x, rmse_max[:,0], rmse_min[:,0], color='b',alpha=0.3,
            linewidth=0, label=r'RMSE')
288
289     if highlight == 1:
290
291         axa.fill_between(x, 0, 1, where=x > 18, color=shadecolor, alpha=0.5,
                transform=axa.get_xaxis_transform())
292
293
294
295     #leg = axa.legend(loc='upper left', frameon=False)
296     leg = axa.legend(frameon=False)
297
298     # B
299
300     b_ylabel = y_labels_rmse(b_name)
301
302     axb = plt.subplot(312, sharex=axa)
303     axb.set_ylabel(r'{}'.format(b_ylabel))
304     # axb.xaxis.set_visible(False)
```

```
305     axb.set_xticklabels([])
306     axb.xaxis.set_tick_params(labelbottom=False)
307     #axb.set_ylim(0, 360)
308     # plt.setp(axb.get_xticklabels(), visible=False)
309     axb.grid(color='k', linestyle='-', linewidth=0.7, alpha=0.2)
310     axb.yaxis.set_major_locator(MultipleLocator(90))
311     axb.yaxis.set_minor_locator(MultipleLocator(18))
312     axb.tick_params(which='major', length=7, width=1, direction='in', top=True,
            right=True)
313     axb.tick_params(which='minor', length=3, width=1, direction='in', top=True,
            right=True)
314
315     y3 = rmse_max[:,1]
316
317     y4 = rmse_min[:,1]
318
319     #plt.plot(x, y3, 'teal', x, y4, 'orangered')
320     plt.plot(x, y3, 'b')
321     plt.plot(x, y4, 'b')
322
323     axb.fill_between(x, rmse_max[:,1], rmse_min[:,1], color='b',alpha=0.3,
            linewidth=0)
324
325     if highlight == 1:
326
327         axb.fill_between(x, 0, 1, where=x > 18, color=shadecolor, alpha=0.5,
                transform=axb.get_xaxis_transform())
328
329
330
331     # C
332
333     c_ylabel = y_labels_rmse(c_name)
334
335     axc = plt.subplot(313, sharex=axa)
336     axc.set_ylabel(r'{}'.format(c_ylabel))
337     axc.set_xlabel(r'Time')
338     # plt.setp(axc.get_xticklabels(), visible=False)
339     axc.grid(color='k', linestyle='-', linewidth=0.7, alpha=0.2)
340     #axc.set_ylim(-0.1, 2.1)
341     axc.yaxis.set_major_locator(MultipleLocator(1))
342     axc.yaxis.set_minor_locator(MultipleLocator(0.2))
343     axc.tick_params(which='major', length=7, width=1, direction='in', top=True,
            right=True)
344     axc.tick_params(which='minor', length=3, width=1, direction='in', top=True,
            right=True)
345     xtcks = np.arange(0, 43, 6)
346     lbl = ['0:00','6:00', '12:00', '18:00', '00:00', '6:00', '12:00', '18:00']
347     axc.annotate('May 14', (0.06, 0.05), xycoords='figure fraction',
            fontsize=ANOT_SIZE, color='k')
348     axc.annotate('May 15', (0.5, 0.05), xycoords='figure fraction',
            fontsize=ANOT_SIZE, color='k')
349
```

```
350     axc.set_xticks(ticks=xtcks)
351     axc.set_xticklabels(lbl)
352     #axc.annotate('a)', (.03, .82), xycoords='figure fraction',
            fontsize=SMALL_SIZE, color='k')
353     #axc.annotate('b)', (.03, .57), xycoords='figure fraction',
            fontsize=SMALL_SIZE, color='k')
354     #axc.annotate('c)', (.03, .3), xycoords='figure fraction',
            fontsize=SMALL_SIZE, color='k')
355
356     y5 = rmse_max[:,2]
357
358     y6 = rmse_min[:,2]
359
360     #plt.plot(x, y5, 'teal', x, y6, 'orangered')
361     plt.plot(x, y5, 'b')
362     plt.plot(x, y6, 'b')
363
364     axc.fill_between(x, rmse_max[:,2], rmse_min[:,2], color='b',alpha=0.3,
            linewidth=0)
365
366
367
368     if highlight == 1:
369
370         axc.fill_between(x, 0, 1, where=x > 18, color=shadecolor, alpha=0.5,
                transform=axc.get_xaxis_transform())
371
372
373
374     plt.savefig(os.path.join(r'{}'.format(direc_graphs),"rmse_{}_{}_{}_{}.png".format(t_title,a_na
            format="png", dpi=resolution_value)
375
376
377 # Plot u, v, w
378
379 def plot_uvw(path_pf,pf_2d_array,df,df_ventos,data,highlight=0):
380     '''Takes path of Perdigão file (str), data from Perdigão file (2D np array
            of float), data frame with Perdigão data (pandas DataFrame), data frame
            with VENTOS®/M data (pandas DataFrame), data from VENTOS®/M file (2D np
            array of float), flag variable to learn if the shaded area of the last
            24h is to be displayed or not, Returns none'''
381     a_name = "u"
382     b_name = "v"
383     c_name = "w"
384     a1 = df.columns.get_loc(a_name)
385     b1 = df.columns.get_loc(b_name)
386     c1 = df.columns.get_loc(c_name)
387     a2 = df_ventos.columns.get_loc(a_name)
388     b2 = df_ventos.columns.get_loc(b_name)
389     c2 = df_ventos.columns.get_loc(c_name)
390
391     # Plot formatting:
392     plt.rcParams['font.family'] = 'sans-serif'
```

```
393     #plt.rcParams.update({'font.sans-serif':'Helvetica'})
394     SMALL_SIZE = 10
395     MEDIUM_SIZE = 12
396     ANOT_SIZE = SMALL_SIZE - 2
397     NUMBER_SIZE = SMALL_SIZE
398     plt.rc('font', size=SMALL_SIZE)     # controls default text sizes
399     plt.rc('axes', titlesize=SMALL_SIZE) # fontsize of the axes title
400     plt.rc('axes', labelsize=16) # fontsize of the x and y labels
401     plt.rc('xtick', labelsize=SMALL_SIZE) # fontsize of the tick labels
402     plt.rc('ytick', labelsize=SMALL_SIZE) # fontsize of the tick labels
403     plt.rc('legend', fontsize=SMALL_SIZE) # legend fontsize
404     shadecolor = 'lavenderblush'
405
406     x = pf_2d_array[:, 1]
407     xlim_l = int(x[0])
408     xlim_r = int(x[-1])
409
410     #
411
412     fig = plt.figure() # an empty figure with no axes
413
414     resolution_value = 200
415
416     plt.rcParams["figure.figsize"] = (12,6)
417
418     plt.rcParams['figure.dpi'] = resolution_value
419
420     t_title = tower_title(path_pf)
421     h_title = sonic_title(path_pf)
422
423     fig, ax_lst = plt.subplots(3, 1) # a figure with a 2x2 grid of Axes
424
425     fig.suptitle('Tower {} at {} height'.format(t_title, h_title), fontsize=18)
426
427     #subtitle = 'Wind Speed Components U,V,W'
428
429
430     # A
431
432     a_ylabel = y_labels(a_name)
433
434     axa = plt.subplot(311)
435
436     axa.set_ylabel(r'{}'.format(a_ylabel))
437     #axa.set_ylim(0, 8.5)
438     axa.set_xlim(xlim_l, xlim_r)
439     axa.xaxis.set_tick_params(labelbottom=False)
440     axa.grid(color='k', linestyle='-', linewidth=0.7, alpha=0.2)
441     axa.yaxis.set_major_locator(MultipleLocator(4))
442     axa.yaxis.set_minor_locator(MultipleLocator(0.8))
443     axa.tick_params(which='major', length=7, width=1, direction='in', top=True,
            right=True)
444     axa.tick_params(which='minor', length=3, width=1, direction='in', top=True,
```

```
        right=True)
445
446     x = pf_2d_array[:,1]
447     y1 = pf_2d_array[:,a1]
448
449     y2 = data[:,a2]
450
451     plt.plot(x, y1, 'teal', label='Measurements')
452     plt.plot(x, y2, 'orangered', label='Simulation')
453
454     if highlight == 1:
455
456         axa.fill_between(x, 0, 1, where=x > 18*3600, color=shadecolor, alpha=0.5,
            transform=axa.get_xaxis_transform())
457
458     #leg = axa.legend(loc='upper left', frameon=False)
459     leg = axa.legend(frameon=False)
460
461     # B
462
463     b_ylabel = y_labels(b_name)
464
465     axb = plt.subplot(312, sharex=axa)
466     axb.set_ylabel(r'{}'.format(b_ylabel))
467     # axb.xaxis.set_visible(False)
468     axb.set_xticklabels([])
469     axb.xaxis.set_tick_params(labelbottom=False)
470     #axb.set_ylim(0, 360)
471     # plt.setp(axb.get_xticklabels(), visible=False)
472     axb.grid(color='k', linestyle='-', linewidth=0.7, alpha=0.2)
473     axb.yaxis.set_major_locator(MultipleLocator(2))
474     axb.yaxis.set_minor_locator(MultipleLocator(0.4))
475     axb.tick_params(which='major', length=7, width=1, direction='in', top=True,
        right=True)
476     axb.tick_params(which='minor', length=3, width=1, direction='in', top=True,
        right=True)
477
478     y3 = pf_2d_array[:,b1]
479
480     y4 = data[:,b2]
481
482     plt.plot(x, y3, 'teal', x, y4, 'orangered')
483
484     if highlight == 1:
485
486         axb.fill_between(x, 0, 1, where=x > 18*3600, color=shadecolor, alpha=0.5,
            transform=axb.get_xaxis_transform())
487
488     # C
489
490     c_ylabel = y_labels(c_name)
491
492     axc = plt.subplot(313, sharex=axa)
```

```
493     axc.set_ylabel(r'{}'.format(c_ylabel))
494     axc.set_xlabel(r'Time')
495     # plt.setp(axc.get_xticklabels(), visible=False)
496     axc.grid(color='k', linestyle='-', linewidth=0.7, alpha=0.2)
497     #axc.set_ylim(-0.1, 2.1)
498     axc.yaxis.set_major_locator(MultipleLocator(1))
499     axc.yaxis.set_minor_locator(MultipleLocator(0.2))
500     axc.tick_params(which='major', length=7, width=1, direction='in', top=True,
            right=True)
501     axc.tick_params(which='minor', length=3, width=1, direction='in', top=True,
            right=True)
502     xtcks = np.arange(0*3600, 43*3600, 3600*6)
503     lbl = ['0:00','6:00', '12:00', '18:00', '00:00', '6:00', '12:00', '18:00']
504     axc.annotate('May 14', (0.06, 0.05), xycoords='figure fraction',
            fontsize=ANOT_SIZE, color='k')
505     axc.annotate('May 15', (0.5, 0.05), xycoords='figure fraction',
            fontsize=ANOT_SIZE, color='k')
506
507     axc.set_xticks(ticks=xtcks)
508     axc.set_xticklabels(lbl)
509     #axc.annotate('a)', (.03, .82), xycoords='figure fraction',
            fontsize=SMALL_SIZE, color='k')
510     #axc.annotate('b)', (.03, .57), xycoords='figure fraction',
            fontsize=SMALL_SIZE, color='k')
511     #axc.annotate('c)', (.03, .3), xycoords='figure fraction',
            fontsize=SMALL_SIZE, color='k')
512
513     y5 = pf_2d_array[:,c1]
514
515     y6 = data[:,c2]
516
517     plt.plot(x, y5, 'teal', x, y6, 'orangered')
518
519     if highlight == 1:
520
521         axc.fill_between(x, 0, 1, where=x > 18*3600, color=shadecolor, alpha=0.5,
                transform=axc.get_xaxis_transform())
522
523
524     plt.savefig(os.path.join(r'{}'.format(direc_graphs),"{}_{}_{}_{}_{}.png".format(t_title,h_titl
            format="png", dpi=resolution_value)
525
526
527 # Plot v, dir, t
528
529 def
        plot_v_dir_t(path_pf,pf_2d_array,a_name,b_name,c_name,df,df_ventos,data,highlight=0):
530     '''Takes path of Perdigão file (str), data from Perdigão file (2D np array
            of float), first variable name(str), second variable name(str), third
            variable name(str), data frame with Perdigão data (pandas DataFrame),
            data frame with VENTOS®/M data (pandas DataFrame), data from VENTOS®/M
            file (2D np array of float), flag variable to learn if the shaded area
            of the last 24h is to be displayed or not, returns none'''
```

```
531     a1 = df.columns.get_loc(a_name)
532     b1 = df.columns.get_loc(b_name)
533     c1 = df.columns.get_loc(c_name)
534     a2 = df_ventos.columns.get_loc(a_name)
535     b2 = df_ventos.columns.get_loc(b_name)
536     c2 = df_ventos.columns.get_loc(c_name)
537
538     # Plot formatting:
539     plt.rcParams['font.family'] = 'sans-serif'
540     #plt.rcParams.update({'font.sans-serif':'Helvetica'})
541     SMALL_SIZE = 10
542     MEDIUM_SIZE = 12
543     ANOT_SIZE = SMALL_SIZE - 2
544     NUMBER_SIZE = SMALL_SIZE
545     plt.rc('font', size=SMALL_SIZE)    # controls default text sizes
546     plt.rc('axes', titlesize=SMALL_SIZE) # fontsize of the axes title
547     plt.rc('axes', labelsize=16) # fontsize of the x and y labels
548     plt.rc('xtick', labelsize=SMALL_SIZE) # fontsize of the tick labels
549     plt.rc('ytick', labelsize=SMALL_SIZE) # fontsize of the tick labels
550     plt.rc('legend', fontsize=SMALL_SIZE) # legend fontsize
551     shadecolor = 'lavenderblush'
552
553     x = pf_2d_array[:, 1]
554     xlim_l = int(x[0])
555     xlim_r = int(x[-1])
556
557     #
558
559     fig = plt.figure() # an empty figure with no axes
560
561     resolution_value = 200
562
563     plt.rcParams["figure.figsize"] = (12,6)
564
565     plt.rcParams['figure.dpi'] = resolution_value
566
567     t_title = tower_title(path_pf)
568     h_title = sonic_title(path_pf)
569
570     fig, ax_lst = plt.subplots(3, 1) # a figure with a 2x2 grid of Axes
571
572     #plt.subplots_adjust(hspace=0.3)
573
574     fig.suptitle('Tower {} at {} height'.format(t_title, h_title), fontsize=18)
575
576     #subtitle = 'Wind Speed Components U,V,W'
577
578
579     # A
580
581     a_ylabel = y_labels(a_name)
582
583     axa = plt.subplot(311)
```

```
584
585     axa.set_ylabel(r'{}'.format(a_ylabel))
586     #axa.set_ylim(0, 8.5)
587     axa.set_xlim(xlim_l, xlim_r)
588     axa.xaxis.set_tick_params(labelbottom=False)
589     axa.grid(color='k', linestyle='-', linewidth=0.7, alpha=0.2)
590     axa.yaxis.set_major_locator(MultipleLocator(2))
591     axa.yaxis.set_minor_locator(MultipleLocator(0.4))
592     axa.tick_params(which='major', length=7, width=1, direction='in', top=True,
            right=True)
593     axa.tick_params(which='minor', length=3, width=1, direction='in', top=True,
            right=True)
594
595     x = pf_2d_array[:,1]
596     y1 = pf_2d_array[:,a1]
597
598     y2 = data[:,a2]
599
600     plt.plot(x, y1, 'teal', label='Measurements')
601     plt.plot(x, y2, 'orangered', label='Simulation')
602
603     if highlight == 1:
604
605         axa.fill_between(x, 0, 1, where=x > 18*3600, color=shadecolor, alpha=0.5,
                transform=axa.get_xaxis_transform())
606
607
608
609     #leg = axa.legend(loc='upper left', frameon=False)
610     leg = axa.legend(frameon=False)
611
612     # B
613
614     b_ylabel = y_labels(b_name)
615
616     axb = plt.subplot(312, sharex=axa)
617     axb.set_ylabel(r'{}'.format(b_ylabel))
618     # axb.xaxis.set_visible(False)
619     axb.set_xticklabels([])
620     axb.xaxis.set_tick_params(labelbottom=False)
621     axb.set_ylim(0, 360)
622     # plt.setp(axb.get_xticklabels(), visible=False)
623     axb.grid(color='k', linestyle='-', linewidth=0.7, alpha=0.2)
624     axb.yaxis.set_major_locator(MultipleLocator(90))
625     axb.yaxis.set_minor_locator(MultipleLocator(18))
626     axb.tick_params(which='major', length=7, width=1, direction='in', top=True,
            right=True)
627     axb.tick_params(which='minor', length=3, width=1, direction='in', top=True,
            right=True)
628
629     y3 = pf_2d_array[:,b1]
630
631     y4 = data[:,b2]
```

```
632
633     plt.plot(x, y3, 'teal', x, y4, 'orangered')
634
635     if highlight == 1:
636
637         axb.fill_between(x, 0, 1, where=x > 18*3600, color=shadecolor, alpha=0.5,
                transform=axb.get_xaxis_transform())
638
639
640     # C
641
642     c_ylabel = y_labels(c_name)
643
644     axc = plt.subplot(313, sharex=axa)
645     axc.set_ylabel(r'{}'.format(c_ylabel))
646     axc.set_xlabel(r'Time')
647     # plt.setp(axc.get_xticklabels(), visible=False)
648     axc.grid(color='k', linestyle='-', linewidth=0.7, alpha=0.2)
649     #axc.set_ylim(-0.1, 2.1)
650     axc.yaxis.set_major_locator(MultipleLocator(1))
651     axc.yaxis.set_minor_locator(MultipleLocator(0.2))
652     axc.tick_params(which='major', length=7, width=1, direction='in', top=True,
            right=True)
653     axc.tick_params(which='minor', length=3, width=1, direction='in', top=True,
            right=True)
654     xtcks = np.arange(0*3600, 43*3600, 3600*6)
655     lbl = ['0:00','6:00', '12:00', '18:00', '00:00', '6:00', '12:00', '18:00']
656     axc.annotate('May 14', (0.06, 0.05), xycoords='figure fraction',
            fontsize=ANOT_SIZE, color='k')
657     axc.annotate('May 15', (0.5, 0.05), xycoords='figure fraction',
            fontsize=ANOT_SIZE, color='k')
658
659     axc.set_xticks(ticks=xtcks)
660     axc.set_xticklabels(lbl)
661     #axc.annotate('a)', (.03, .82), xycoords='figure fraction',
            fontsize=SMALL_SIZE, color='k')
662     #axc.annotate('b)', (.03, .57), xycoords='figure fraction',
            fontsize=SMALL_SIZE, color='k')
663     #axc.annotate('c)', (.03, .3), xycoords='figure fraction',
            fontsize=SMALL_SIZE, color='k')
664
665     y5 = pf_2d_array[:,c1]
666
667     y6 = data[:,c2]
668
669     plt.plot(x, y5, 'teal', x, y6, 'orangered')
670
671     if highlight == 1:
672
673         axc.fill_between(x, 0, 1, where=x > 18*3600, color=shadecolor, alpha=0.5,
                transform=axc.get_xaxis_transform())
674
675
```

```
676
677     plt.savefig(os.path.join(r'{}'.format(direc_graphs),"{}_{}_{}_{}_{}.png".format(t_title,h_titl
            format="png", dpi=resolution_value)
678
679
680 # Plot uu, vv, ww
681
682 def plot_variances(path_pf,pf_2d_array,df,df_ventos,data,highlight=0):
683     '''Takes path of Perdigão file (str), data from Perdigão file (2D np array
            of float), data frame with Perdigão data (pandas DataFrame), data frame
            with VENTOS®/M data (pandas DataFrame), data from VENTOS®/M file (2D np
            array of float), flag variable to learn if the shaded area of the last
            24h is to be displayed or not, returns none'''
684     a_name = "uu"
685     b_name = "vv"
686     c_name = "ww"
687     a1 = df.columns.get_loc(a_name)
688     b1 = df.columns.get_loc(b_name)
689     c1 = df.columns.get_loc(c_name)
690     a2 = df_ventos.columns.get_loc(a_name)
691     b2 = df_ventos.columns.get_loc(b_name)
692     c2 = df_ventos.columns.get_loc(c_name)
693
694     # Plot formatting:
695     plt.rcParams['font.family'] = 'sans-serif'
696     #plt.rcParams.update({'font.sans-serif':'Helvetica'})
697     SMALL_SIZE = 10
698     MEDIUM_SIZE = 12
699     ANOT_SIZE = SMALL_SIZE - 2
700     NUMBER_SIZE = SMALL_SIZE
701     plt.rc('font', size=SMALL_SIZE)     # controls default text sizes
702     plt.rc('axes', titlesize=SMALL_SIZE) # fontsize of the axes title
703     plt.rc('axes', labelsize=16) # fontsize of the x and y labels
704     plt.rc('xtick', labelsize=SMALL_SIZE) # fontsize of the tick labels
705     plt.rc('ytick', labelsize=SMALL_SIZE) # fontsize of the tick labels
706     plt.rc('legend', fontsize=SMALL_SIZE) # legend fontsize
707     shadecolor = 'lavenderblush'
708
709     x = pf_2d_array[:, 1]
710     xlim_l = int(x[0])
711     xlim_r = int(x[-1])
712
713     #
714
715     fig = plt.figure() # an empty figure with no axes
716
717     resolution_value = 200
718
719     plt.rcParams["figure.figsize"] = (12,6)
720
721     plt.rcParams['figure.dpi'] = resolution_value
722
723     t_title = tower_title(path_pf)
```

```
724     h_title = sonic_title(path_pf)
725
726     fig, ax_lst = plt.subplots(3, 1) # a figure with a 2x2 grid of Axes
727
728     fig.suptitle('Tower {} at {} height'.format(t_title, h_title), fontsize=18)
729
730     #subtitle = 'Wind Speed Components U,V,W'
731
732
733     # A
734
735     a_ylabel = y_labels(a_name)
736
737     axa = plt.subplot(311)
738
739     axa.set_ylabel(r'{}'.format(a_ylabel))
740     #axa.set_ylim(0, 3)
741     axa.set_xlim(xlim_l, xlim_r)
742     axa.xaxis.set_tick_params(labelbottom=False)
743     axa.grid(color='k', linestyle='-', linewidth=0.7, alpha=0.2)
744     axa.yaxis.set_major_locator(MultipleLocator(1))
745     axa.yaxis.set_minor_locator(MultipleLocator(0.2))
746     axa.tick_params(which='major', length=7, width=1, direction='in', top=True,
            right=True)
747     axa.tick_params(which='minor', length=3, width=1, direction='in', top=True,
            right=True)
748
749     x = pf_2d_array[:,1]
750     y1 = pf_2d_array[:,a1]
751
752     y2 = data[:,a2]
753
754     plt.plot(x, y1, 'teal', label='Measurements')
755     plt.plot(x, y2, 'orangered', label='Simulation')
756
757     if highlight == 1:
758
759         axa.fill_between(x, 0, 1, where=x > 18*3600, color=shadecolor, alpha=0.5,
                transform=axa.get_xaxis_transform())
760
761
762
763     #leg = axa.legend(loc='upper left', frameon=False)
764     leg = axa.legend(frameon=False)
765
766     # B
767
768     b_ylabel = y_labels(b_name)
769
770     axb = plt.subplot(312, sharex=axa)
771     axb.set_ylabel(r'{}'.format(b_ylabel))
772     # axb.xaxis.set_visible(False)
773     axb.set_xticklabels([])
```

```
774     axb.xaxis.set_tick_params(labelbottom=False)
775     #axb.set_ylim(0, 3)
776     # plt.setp(axb.get_xticklabels(), visible=False)
777     axb.grid(color='k', linestyle='-', linewidth=0.7, alpha=0.2)
778     axb.yaxis.set_major_locator(MultipleLocator(1))
779     axb.yaxis.set_minor_locator(MultipleLocator(0.2))
780     axb.tick_params(which='major', length=7, width=1, direction='in', top=True,
            right=True)
781     axb.tick_params(which='minor', length=3, width=1, direction='in', top=True,
            right=True)
782
783     y3 = pf_2d_array[:,b1]
784
785     y4 = data[:,b2]
786
787     plt.plot(x, y3, 'teal', x, y4, 'orangered')
788
789     if highlight == 1:
790
791         axb.fill_between(x, 0, 1, where=x > 18*3600, color=shadecolor, alpha=0.5,
                transform=axb.get_xaxis_transform())
792
793
794     # C
795
796     c_ylabel = y_labels(c_name)
797
798     axc = plt.subplot(313, sharex=axa)
799     axc.set_ylabel(r'{}'.format(c_ylabel))
800     axc.set_xlabel(r'Time')
801     # plt.setp(axc.get_xticklabels(), visible=False)
802     axc.grid(color='k', linestyle='-', linewidth=0.7, alpha=0.2)
803     #axc.set_ylim(0, 1)
804     axc.yaxis.set_major_locator(MultipleLocator(0.5))
805     axc.yaxis.set_minor_locator(MultipleLocator(0.1))
806     axc.tick_params(which='major', length=7, width=1, direction='in', top=True,
            right=True)
807     axc.tick_params(which='minor', length=3, width=1, direction='in', top=True,
            right=True)
808     xtcks = np.arange(0*3600, 43*3600, 3600*6)
809     lbl = ['0:00','6:00', '12:00', '18:00', '00:00', '6:00', '12:00', '18:00']
810     axc.annotate('May 14', (0.06, 0.05), xycoords='figure fraction',
            fontsize=ANOT_SIZE, color='k')
811     axc.annotate('May 15', (0.505, 0.05), xycoords='figure fraction',
            fontsize=ANOT_SIZE, color='k')
812
813     axc.set_xticks(ticks=xtcks)
814     axc.set_xticklabels(lbl)
815     #axc.annotate('a)', (.03, .82), xycoords='figure fraction',
            fontsize=SMALL_SIZE, color='k')
816     #axc.annotate('b)', (.03, .57), xycoords='figure fraction',
            fontsize=SMALL_SIZE, color='k')
817     #axc.annotate('c)', (.03, .3), xycoords='figure fraction',
```

```
            fontsize=SMALL_SIZE, color='k')
818
819         y5 = pf_2d_array[:,c1]
820
821         y6 = data[:,c2]
822
823         plt.plot(x, y5, 'teal', x, y6, 'orangered')
824
825         if highlight == 1:
826
827             axc.fill_between(x, 0, 1, where=x > 18*3600, color=shadecolor, alpha=0.5,
                    transform=axc.get_xaxis_transform())
828
829
830
831         plt.savefig(os.path.join(r'{}'.format(direc_graphs),"{}_{}_{}_{}_{}.png".format(t_title,h_titl
                format="png", dpi=resolution_value)
832
833
834     # Path VENTOS V2
835
836     def ventos_file_name_v3(dfv2_sonic,tower_p,height,direc):
837         '''Takes VENTOS total sonic list (pandas data frame), designation of the
                tower (str), sonic height (str) and directory path (str), Returns file
                path (str)'''
838         sonic_list = dfv2_sonic.loc[:,'sonic']
839
840         sl = '!{}.{}m'.format(tower_p,height)
841         if sl == '!v03.2m':
842
843             i = 166
844
845         else:
846
847             i = sonic_list[sonic_list == sl].index[0]
848
849         i +=1
850
851         if i < 10:
852
853             path_ventos_v = direc +
                    r'\Simulações\all_sonics_v2\all_sonics_v2\af_series0000{}.plt'.format(i)
854
855         if i > 9 and i < 100:
856
857             path_ventos_v = direc +
                    r'\Simulações\all_sonics_v2\all_sonics_v2\af_series000{}.plt'.format(i)
858
859         if i > 99:
860
861             path_ventos_v = direc +
                    r'\Simulações\all_sonics_v2\all_sonics_v2\af_series00{}.plt'.format(i)
862
```

```
863     return path_ventos_v
864
865
866 # Get df and np 2d array on perdigao data
867
868 def df_perdigao_file(path_pf):
869     '''Takes path to Perdigão file (str), Returns data frame (pandas DataFrame)
            and 2D array (2D np array) containing processed data from the Perdigão
            file and data number of rows (int)'''
870     df = pd.read_csv(r'{}'.format(path_pf), index_col=[0])
871
872     k = int((df.iat[1, 1] - df.iat[0, 1])/2)
873
874     dt = np.arange(0, len(df.index), dtype=int)
875
876     for a in dt:
877
878         df.iat[a, 1] = df.iat[a, 1] + k
879
880     pf_2d_array = df.to_numpy()
881
882     pf_num_rows = len(pf_2d_array)
883     #pf_num_cols = len(pf_2d_array[0])
884
885     return df, pf_2d_array, pf_num_rows
886
887
888 # Get df and np 2d array on VENTOS data
889
890 def df_ventos_file(tower_p,height,direc_v,path_v):
891     '''Takes tower name (str), height (int), directory path (str), file path
            (str), Returns data frame (pandas DataFrame) and 2D array (2D np array)
            containing processed data from the VENTOS file'''
892     dfv2_sonic = pd.read_csv(r'{}'.format(path_v))
893
894     path_ventos_v = ventos_file_name_v3(dfv2_sonic, tower_p, height, direc_v)
895
896     data = np.genfromtxt(r'{}'.format(path_ventos_v), skip_header=7)
897
898     data = data[1:]
899
900     df_ventos = pd.DataFrame(data, columns = ['seconds', 'TP', 'N^2', 'u', 'v',
            'w', 'vh', 'dir', 'wdeg', 'shear', 'te_v', 'ti_v', 'tih_v', 'uu', 'vv',
            'ww'])
901
902     df_ventos = turbulence_5min_ventos(df_ventos)
903
904     data = df_ventos.to_numpy()
905
906     return df_ventos, data
907
908
909 # RMSE table
```

```
910
911 def tab_rmse_v(rmse,var,nhour,heights):
912     '''Takes RMSE 3D array (np 3D array of float), variable flag (int), number
            of hours of the data (int), array with all heights for selected tower
            (np array of int), Returns data frame (pandas DataFrame) and 2D array
            (2D np array) containing hourly RMSE data, for the last 24h for the
            selected variable, in the selected tower'''
913     rmse_v = rmse[-24:,var,:]
914     rmse_v = rmse_v.T
915
916     hours_array = np.arange(nhour)
917     hours_array = hours_array[-24:]
918
919     i1 = 0
920     for a in hours_array:
921
922         b = a%24
923         hours_array[i1] = b
924         i1 += 1
925
926     df_rmse = pd.DataFrame(rmse_v,index=heights,columns=hours_array)
927
928     return df_rmse, rmse_v
929
930
931 def tab_bias_v(bias,var,nhour,heights):
932     '''Takes RMSE 3D array (np 3D array of float), variable flag (int), number
            of hours of the data (int), array with all heights for selected tower
            (np array of int), Returns data frame (pandas DataFrame) and 2D array
            (2D np array) containing hourly RMSE data, for the last 24h for the
            selected variable, in the selected tower'''
933     bias_v = bias[-24:,var,:]
934     bias_v = bias_v.T
935
936     hours_array = np.arange(nhour)
937     hours_array = hours_array[-24:]
938
939     i1 = 0
940     for a in hours_array:
941
942         b = a%24
943         hours_array[i1] = b
944         i1 += 1
945
946     df_bias = pd.DataFrame(bias_v,index=heights,columns=hours_array)
947
948     return df_bias, bias_v
949
950
951 # Save function for rmse table
952
953 def save_rmse_table(df_rmse,path_pf,rmse_var,a_name,b_name,c_name,tower_p):
954     '''Takes hourly RMSE data frame (pandas DataFrame), Perdigão file path
```

```
        (str), variable flag (int), names of the three chosen variables for
        plotting (3*str), tower name (str), Returns none'''
955     if rmse_var == 0:
956
957         name = a_name
958
959     if rmse_var == 1:
960
961         name = b_name
962
963     if rmse_var == 2:
964
965         name = c_name
966
967     date_index = path_pf.find('201')
968     s = path_pf[date_index:date_index+23]
969
970     direc = r'C:\Users\Baba\Desktop\João\Tese\Python\Teste_1\Perdigao_python\App
        2.0\tables'
971
972     df_rmse.to_csv(os.path.join(r'{}'.format(direc),'rmse_{}_{}_table_last_24h_{}.csv'.format(name
973     df_rmse.to_excel(os.path.join(r'{}'.format(direc),'rmse_{}_{}_table_last_24h_{}.xls'.format(na
974     df_rmse = df_rmse.to_numpy()
975     np.savetxt(os.path.join(r'{}'.format(direc),'rmse_{}_{}_table_last_24h_{}.txt'.format(name,tow
        df_rmse, fmt = "%.4f")
```

B.3.3 Tables RMSE and Bias

Equations used for RMSE and Bias calculation:

$$\text{RMSE} = \sqrt{\left(\frac{1}{N}\sum_{i=1}^{N}(sim_i - exp_i)^2\right)} \tag{B.1}$$

$$\text{Bias} = \frac{1}{N}\sum_{i=1}^{N}(sim_i - exp_i)^2 \tag{B.2}$$

Module code:

```
1   # RMSE, BIAS tables
2
3   import graph_functions as gf
4   import numpy as np
5   import pandas as pd
6   import tower_location as tl
7   import os.path
8
9   def rmse_bias_2 (pf_2d_array,data,v_name,period):
```

```
10
11      i = df.columns.get_loc(v_name)
12      j = df_ventos.columns.get_loc(v_name)
13
14      pf_num_rows = len(pf_2d_array)
15
16      if period == 24:
17
18          exp = pf_2d_array[-24*12:,i]
19          sim = data[-24*12:,j]
20
21          if v_name == 'dir':
22
23              err, mse, rmse, bias = gf.rmse_f(exp, sim, True)
24
25          else:
26
27              err, mse, rmse, bias = gf.rmse_f(exp, sim)
28
29      else:
30
31          nhour = int(pf_num_rows/period/12)
32          err = np.zeros(nhour)
33          mse = np.zeros(nhour)
34          rmse = np.zeros(nhour)
35          bias = np.zeros(nhour)
36
37          for k in range(nhour):
38
39              exp = pf_2d_array[k*period*12:k*period*12+period*12,i]
40              sim = data[k*period*12:k*period*12+period*12,j]
41
42              if v_name == 'dir':
43
44                  err[k], mse[k], rmse[k], bias[k] = gf.rmse_f(exp, sim, True)
45
46              else:
47
48                  err[k], mse[k], rmse[k], bias[k] = gf.rmse_f(exp, sim)
49
50      return rmse, bias
51
52  #t_name = np.array(["tnw01", "tnw02", "tnw03", "tnw04", "tnw05", "tnw06",
        "tnw07", "tnw08", "tnw09", "tnw10", "tnw11", "tnw12", "tnw13", "tnw14",
        "tnw15", "tnw16", "tse01", "tse02", "tse04", "tse05", "tse06", "tse07",
        "tse08", "tse09", "tse10", "tse11", "tse12", "tse13", "rsw01", "rsw02",
        "rsw03", "rsw04", "rsw05", "rsw06", "rsw07", "rsw08", "rne01", "rne02",
        "rne03", "rne04", "rne06", "rne07", "v01", "v03", "v04", "v05", "v06",
        "v07", "Extreme_SW", "Extreme_NE"])
53  towers = np.array(["tnw01", "tnw02", "tnw03"])#"v01", "v03", "v04", "v05",
        "v06", "v07"])#, "tse05", "tse06", "tse07", "tse08", "tse09", "tse10",
        "tse11", "tse12", "tse13"])#"tnw01", "tnw02", "tnw03", "tnw04", "tnw05"])#,
        "tnw06", "tnw07", "tnw08", "tnw09", "tnw10"])#
```

```
54
55 #towers = t_name
56 #tower_p = 'tnw05'
57 period = 6
58
59 if period == 24:
60
61     #df_24 = pd.DataFrame(columns = ['sonic', 'rmse_24h_vh', 'bias_24h_vh',
       'rmse_24h_dir', 'bias_24h_dir', 'rmse_24h_tke', 'bias_24h_tke'])
62
63     df_24 = pd.read_csv('rmse_24h_table_20170514_20170515_0-18h.csv',
       index_col=False)
64
65 if period == 6:
66
67     df_6_vh = pd.DataFrame(columns = ['sonic', 'rmse_0-6h_vh', 'bias_0-6h_vh',
       'rmse_6-12h_vh', 'bias_6-12h_vh', 'rmse_12-18h_vh', 'bias_12-18h_vh',
       'rmse_18-24h_vh', 'bias_18-24h_vh'])
68     df_6_dir = pd.DataFrame(columns = ['sonic', 'rmse_0-6h_dir',
       'bias_0-6h_dir', 'rmse_6-12h_dir', 'bias_6-12h_dir', 'rmse_12-18h_dir',
       'bias_12-18h_dir', 'rmse_18-24h_dir', 'bias_18-24h_dir'])
69     df_6_tke = pd.DataFrame(columns = ['sonic', 'rmse_0-6h_tke',
       'bias_0-6h_tke', 'rmse_6-12h_tke', 'bias_6-12h_tke', 'rmse_12-18h_tke',
       'bias_12-18h_tke', 'rmse_18-24h_tke', 'bias_18-24h_tke'])
70
71     #df_6_vh = pd.read_csv('rmse_6h_vh_table_20170514_20170515_0-18h.csv',
       index_col=False)
72     #df_6_dir = pd.read_csv('rmse_6h_dir_table_20170514_20170515_0-18h.csv',
       index_col=False)
73     #df_6_tke = pd.read_csv('rmse_6h_tke_table_20170514_20170515_0-18h.csv',
       index_col=False)
74
75 for tower_p in towers:
76
77     j = tl.tower_index_pos(tower_p)
78
79     heights = tl.sonics_available_name(j)
80
81     direc_pf =
       r'C:\Users\Baba\Desktop\João\Tese\Python\Teste_1\Perdigao_python\App
       2.0\data_pf'
82     path_ventos_sonics_coord =
       r'C:\Users\Baba\Desktop\João\Tese\Python\Teste_1\Perdigao_python\App
       2.0\sonics_coord_est_nor_z.csv'
83     direc_sim = r'C:\Users\Baba\Desktop\João\Tese'
84
85     a_name = "vh"
86     b_name = "dir"
87     c_name = "tke"
88
89     nvars = np.array(a_name)
90     nvars = np.append(a_name, [b_name, c_name])
91
```

```
92
93      sta = 0
94      he = 0
95
96      for height in heights:
97
98          ''' Perdigao File '''
99
100         path_pf =
                r'{}\{}m_{}_20170514_20170515_0-18h_P-5min.csv'.format(direc_pf,height,tower_p)
101
102         df, pf_2d_array, pf_num_rows = gf.df_perdigao_file(path_pf)
103
104         ''' VENTOS File V2'''
105
106         df_ventos, data =
                gf.df_ventos_file(tower_p,height,direc_sim,path_ventos_sonics_coord)
107
108         sonic = '{}.{}m'.format(tower_p, height)
109
110         if sta ==0:
111             nh = int(pf_num_rows/period/12)
112             var = len(nvars)
113             h = len(heights)
114
115             rmse_s = np.zeros((nh,var,h))
116             bias_s = np.zeros((nh,var,h))
117             rmse_24 = np.zeros((var,h))
118             bias_24 = np.zeros((var,h))
119
120             sta=1
121
122         va = 0
123
124         for v_name in nvars:
125
126             rmse, bias = rmse_bias_2(pf_2d_array,data,v_name,period)
127
128             if period == 24:
129
130                 rmse_24[va,he] = rmse
131                 bias_24[va,he] = bias
132
133             else:
134
135                 rmse_s[:,va,he] = rmse
136                 bias_s[:,va,he] = bias
137
138             va+=1
139
140         if period == 24:
141
142             df_24 = df_24.append({'sonic' : sonic, 'rmse_24h_vh':rmse_24[0,he],
```

```
                'bias_24h_vh':bias_24[0,he], 'rmse_24h_dir':rmse_24[1,he],
                'bias_24h_dir':bias_24[1,he], 'rmse_24h_tke':rmse_24[2,he],
                'bias_24h_tke':bias_24[2,he]}, ignore_index = True)

        if period == 6:

            df_6_vh = df_6_vh.append({'sonic' : sonic,
                'rmse_0-6h_vh':rmse_s[-4,0,he], 'bias_0-6h_vh':bias_s[-4,0,he],
                'rmse_6-12h_vh':rmse_s[-3,0,he], 'bias_6-12h_vh':bias_s[-3,0,he],
                'rmse_12-18h_vh':rmse_s[-2,0,he],
                'bias_12-18h_vh':bias_s[-2,0,he],
                'rmse_18-24h_vh':rmse_s[-1,0,he],
                'bias_18-24h_vh':bias_s[-1,0,he]}, ignore_index = True)
            df_6_dir = df_6_dir.append({'sonic' : sonic,
                'rmse_0-6h_dir':rmse_s[-4,1,he], 'bias_0-6h_dir':bias_s[-4,1,he],
                'rmse_6-12h_dir':rmse_s[-3,1,he],
                'bias_6-12h_dir':bias_s[-3,1,he],
                'rmse_12-18h_dir':rmse_s[-2,1,he],
                'bias_12-18h_dir':bias_s[-2,1,he],
                'rmse_18-24h_dir':rmse_s[-1,1,he],
                'bias_18-24h_dir':bias_s[-1,1,he]}, ignore_index = True)
            df_6_tke = df_6_tke.append({'sonic' : sonic,
                'rmse_0-6h_tke':rmse_s[-4,2,he], 'bias_0-6h_tke':bias_s[-4,2,he],
                'rmse_6-12h_tke':rmse_s[-3,2,he],
                'bias_6-12h_tke':bias_s[-3,2,he],
                'rmse_12-18h_tke':rmse_s[-2,2,he],
                'bias_12-18h_tke':bias_s[-2,2,he],
                'rmse_18-24h_tke':rmse_s[-1,2,he],
                'bias_18-24h_tke':bias_s[-1,2,he]}, ignore_index = True)


        he+=1

if period == 24:

    df_24.to_csv('rmse_24h_table_20170514_20170515_0-18h.csv', index=None)

if period == 6:

    df_6_vh.to_csv('rmse_6h_vh_table_20170514_20170515_0-18h.csv', index=None)
    df_6_dir.to_csv('rmse_6h_dir_table_20170514_20170515_0-18h.csv', index=None)
    df_6_tke.to_csv('rmse_6h_tke_table_20170514_20170515_0-18h.csv', index=None)
```

Appendix C

Turbulence Parameters Calculation

Equations used for turbulence parameters calculation:

$$\overline{\mathrm{u'u'}} = \left(\frac{\overline{u_1^2} + \overline{u_2^2} + \cdots + \overline{u_n^2}}{n}\right) + \left(\frac{u'u'_1 \cdot u'u'_2 \cdot \cdots \cdot u'u'_n}{n}\right) + \left(\frac{u_1 + u_2 + \cdots + u_n}{n}\right)^2 \quad \text{(C.1)}$$

$$\overline{\mathrm{v'v'}} = \left(\frac{\overline{v_1^2} + \overline{v_2^2} + \cdots + \overline{v_n^2}}{n}\right) + \left(\frac{v'v'_1 \cdot u'u'_2 \cdot \cdots \cdot v'v'_n}{n}\right) + \left(\frac{v_1 + v_2 + \cdots + v_n}{n}\right)^2 \quad \text{(C.2)}$$

$$\overline{\mathrm{w'w'}} = \left(\frac{\overline{w_1^2} + \overline{w_2^2} + \cdots + \overline{w_n^2}}{n}\right) + \left(\frac{w'w'_1 \cdot w'w'_2 \cdot \cdots \cdot w'w'_n}{n}\right) + \left(\frac{w_1 + w_2 + \cdots + w_n}{n}\right)^2 \quad \text{(C.3)}$$

$$\mathrm{U} = \left(\frac{u_1 + u_2 + \cdots + u_n}{n}\right) \quad \text{(C.4)}$$

$$\mathrm{V} = \left(\frac{v_1 + v_2 + \cdots + v_n}{n}\right) \quad \text{(C.5)}$$

$$\mathrm{W} = \left(\frac{w_1 + w_2 + \cdots + w_n}{n}\right) \quad \text{(C.6)}$$

$$\mathrm{TKE} = \frac{1}{2}\left(\overline{u'u'} + \overline{v'v'} + \overline{w'w'}\right) \tag{C.7}$$

$$\mathrm{TI} = \frac{\sqrt{\frac{1}{3}\left(\overline{u'u'} + \overline{v'v'} + \overline{w'w'}\right)}}{\sqrt{U^2 + V^2 + W^2}} \tag{C.8}$$

$$\mathrm{TIH} = \frac{\sqrt{\frac{1}{2}\left(\overline{u'u'} + \overline{v'v'}\right)}}{\sqrt{U^2 + V^2}} \tag{C.9}$$

Appendix D

Tables

This section contains the tables with RMSE and Bias between 5-min averaged simulation results and measurements from all sonic anemometers deployed in Perdigão-2017. The errors were obtained for the full 24-hour period of interest (from 18:00 UTC May 14, Table D.1), and in 6-hour intervals, for wind speed (Table D.2), direction (Table D.3) and TKE (Table D.4).

Table D.1: RMSE and Bias calculated for wind speed, direction and TKE, from 18:00 May 14 to 18:00 May 15, 2017, for all sonic anemometers and the full 24-hour period. Highlighted maximum (green) and minimum (red) values.

	Wind speed ($m\,s^{-1}$)		Direction (°)		TKE ($m^2\,s^{-2}$)	
Sonic	RMSE	Bias	RMSE	Bias	RMSE	Bias
tnw01.2m	0.9802	0.7062	99.3042	16.5745	0.5006	0.329
tnw01.10m	1.4157	1.0135	90.326	23.2835	0.5891	0.0934
tnw01.20m	1.5014	0.8886	84.9842	8.2622	0.6505	-0.0738
tnw02.2m	1.3094	1.1276	77.7695	8.9289	0.5355	0.3618
tnw02.10m	1.9105	1.5965	75.5996	25.8817	0.427	0.0596
tnw02.20m	1.8758	1.3346	79.5561	26.14	0.5664	-0.2412
tnw03.2m	1.6174	1.3105	43.5371	9.2377	0.5835	0.4611
tnw03.10m	1.7162	0.8583	33.6137	6.0806	0.2621	-0.0203
tnw04.2m	1.0336	0.7709	83.3062	2.8677	0.3123	0.2269
tnw04.4m	1.1886	0.9313	80.4115	-4.8932	0.2447	0.1455
tnw04.6m	1.2806	1.0134	78.2724	-12.0671	0.2246	0.0855
tnw04.8m	1.2981	1.011	75.3123	-11.4944	0.216	0.0322
tnw04.10m	1.3212	1.0071	76.1116	-15.4123	0.2263	-0.0031
tnw04.12m	1.3382	1.0353	77.3252	-9.4724	0.2307	-0.0179
tnw04.20m	1.3345	0.9684	68.2784	-12.5233	0.2755	-0.0714
tnw05.2m	–	–	–	–	–	–
tnw05.4m	–	–	–	–	–	–
tnw05.6m	–	–	–	–	–	–
tnw05.8m	–	–	–	–	–	–
tnw05.10m	–	–	–	–	–	–

Continued on next page

Table D.1 – continued from previous page

	Wind speed ($m\,s^{-1}$)		Direction (°)		TKE ($m^2\,s^{-2}$)	
Sonic	RMSE	Bias	RMSE	Bias	RMSE	Bias
tnw05.12m	–	–	–	–	–	–
tnw05.20m	–	–	–	–	–	–
tnw06.2m	0.9176	0.7344	73.2832	-29.2715	0.3321	0.2139
tnw06.4m	1.0313	0.7676	71.7157	-23.6229	0.3178	0.1355
tnw06.6m	1.2708	0.9256	72.9758	-15.6136	0.3354	0.098
tnw06.8m	1.3611	0.9518	74.1086	-9.3707	0.3374	0.0681
tnw06.10m	1.4533	0.9741	72.9131	-5.9084	0.356	0.0217
tnw06.12m	1.4954	0.9805	73.411	-10.8694	0.3703	-0.0016
tnw06.20m	1.6182	1.0172	72.2349	-17.9577	0.3681	-0.0144
tnw07.4m	1.2792	1.0739	80.4507	26.9246	0.441	0.2816
tnw07.6m	1.3706	1.0896	81.6154	28.5558	0.4519	0.2508
tnw07.8m	1.3396	0.9591	82.6863	25.5941	0.4248	0.1874
tnw07.10m	1.3274	0.8095	84.7489	19.6072	0.4084	0.125
tnw07.12m	1.3325	0.7144	87.1172	19.784	0.4041	0.0682
tnw07.20m	1.3514	0.3864	87.5813	23.9381	0.4025	0.0355
tnw07.30m	1.3527	0.2729	87.9084	19.9542	0.4073	0.006
tnw07.40m	1.3425	0.2615	90.6798	22.2945	0.4248	-0.0241
tnw07.60m	1.4436	0.3623	88.1595	12.8319	0.4504	-0.0277
tnw08.2m	0.8403	0.625	78.9682	20.2158	0.3883	0.2215
tnw08.10m	1.4056	0.8061	94.5582	18.5579	0.4529	0.0889
tnw08.20m	1.6639	0.6704	96.1366	22.4443	0.4509	0.0366
tnw09.2m	0.5318	0.1824	94.2611	20.8304	0.3543	0.1755
tnw09.10m	1.254	0.654	82.332	20.9856	0.4476	0.0491
tnw10.10m	1.3698	1.0188	65.215	-3.9855	0.3549	-0.0142
tnw10.20m	1.4014	0.7906	57.0447	0.7198	0.4709	-0.179
tnw10.30m	1.4496	0.7123	52.1558	1.9192	0.4068	-0.1241
tnw10.40m	1.54	0.7401	44.509	0.6568	0.3607	-0.0993
tnw10.60m	1.6441	0.6962	41.855	2.0871	0.343	-0.101
tnw11.2m	1.7786	1.6045	40.1899	-4.7933	0.4813	0.3823
tnw11.10m	1.3119	0.7959	34.614	-0.8886	0.2504	0.0142
tnw11.20m	1.4201	0.7351	41.0632	-2.9181	0.2479	-0.0508
tnw12.2m	1.4357	1.2896	59.2425	-9.1709	0.3894	0.2846
tnw12.4m	1.3349	1.1863	56.045	-3.9129	0.2793	0.1599
tnw12.6m	1.3204	1.1313	49.1293	-3.458	0.2531	0.0721
tnw12.8m	1.2601	1.029	52.88	-1.7017	0.2455	0.0405
tnw12.10m	1.248	0.9826	50.2716	-4.3847	0.2461	0.0161
tnw12.12m	1.2639	0.9485	49.9251	-8.5222	0.2433	-0.0013
tnw12.20m	1.4496	0.9457	46.3148	-7.6803	0.2402	-0.0464
tnw12.30m	1.4325	0.8519	43.4537	-7.9177	0.2475	-0.0712
tnw13.2m	0.5064	0.2424	82.1445	-18.6724	0.1778	0.1181
tnw13.4m	1.0789	0.7917	84.0798	-14.6144	0.3329	0.2241
tnw13.6m	1.3335	1.109	82.1659	-17.5463	0.3336	0.2126
tnw13.8m	1.5952	1.3853	77.7104	-15.7791	0.342	0.2125

Continued on next page

Table D.1 – continued from previous page

	Wind speed ($m\,s^{-1}$)		Direction (°)		TKE ($m^2\,s^{-2}$)	
Sonic	RMSE	Bias	RMSE	Bias	RMSE	Bias
tnw13.10m	1.5361	1.3429	73.1509	-11.9115	0.3092	0.1723
tnw13.12m	1.4445	1.2494	68.9326	-2.8352	0.2605	0.113
tnw13.20m	1.0499	0.7669	62.7972	2.7128	0.2403	-0.0166
tnw13.30m	0.9888	0.6575	62.5806	-1.091	0.2678	-0.0487
tnw14.2m	0.8348	0.475	92.4815	-3.2827	0.3177	0.2234
tnw14.4m	0.9421	0.6989	87.3306	-2.1144	0.3199	0.2212
tnw14.6m	1.2551	1.0188	87.0529	0.7835	0.3439	0.2286
tnw14.8m	1.3598	1.0962	87.9145	-2.3761	0.3321	0.2155
tnw14.10m	1.4268	1.1758	87.4951	2.7462	0.3029	0.1874
tnw14.12m	1.366	1.1173	84.7939	6.7492	0.2471	0.1308
tnw14.20m	1.0596	0.7935	77.4007	6.4922	0.2236	-0.0277
tnw14.30m	0.95	0.5704	77.0457	3.6543	0.2561	-0.0571
tnw15.2m	0.8686	0.4667	89.8431	1.6782	0.3321	0.2423
tnw15.4m	1.0029	0.6692	84.4007	-3.7619	0.3047	0.2108
tnw15.6m	1.2164	0.9175	87.3592	9.1422	0.275	0.1666
tnw15.8m	1.279	0.9867	86.873	10.802	0.2455	0.1213
tnw15.10m	1.2975	0.9883	88.081	6.1877	0.2322	0.0882
tnw15.12m	1.2707	0.9454	89.2759	5.2904	0.2225	0.0543
tnw15.20m	1.1681	0.8078	89.5599	1.8293	0.2248	-0.0257
tnw15.30m	1.1075	0.6804	83.3297	9.349	0.2432	-0.0547
tnw16.2m	0.5896	0.2475	89.9452	-10.4117	0.23	0.1587
tnw16.4m	0.9356	0.5018	91.175	-8.6265	0.2717	0.1752
tnw16.6m	1.1616	0.7253	88.9528	-11.8832	0.2562	0.1361
tnw16.8m	1.2984	0.9002	88.8294	-7.0202	0.2441	0.0957
tnw16.10m	1.3992	1.0001	89.4266	-3.3191	0.243	0.0652
tnw16.12m	1.4452	1.0087	91.0638	0.013	0.2433	0.0327
tnw16.20m	1.4652	0.9241	89.6199	7.8139	0.2372	-0.0336
tnw16.30m	1.3634	0.7723	83.3512	10.2088	0.2494	-0.054
tse01.10m	1.634	1.1866	96.0187	4.9788	0.4543	0.1891
tse01.30m	1.7851	1.0484	86.3314	4.4725	0.4916	-0.025
tse02.10m	1.6022	1.377	65.4771	-2.763	0.4551	0.1324
tse02.30m	1.5593	0.8968	72.3536	20.5888	0.5845	-0.258
tse04.10m	1.4193	0.5429	44.2266	0.6793	0.3123	-0.032
tse04.20m	1.4399	0.4939	39.195	4.8323	0.3185	-0.0647
tse04.30m	1.4463	0.5043	42.9858	6.1567	0.3047	-0.0705
tse04.40m	1.4837	0.4736	42.3668	5.3557	0.3046	-0.0748
tse04.60m	1.5163	0.3988	40.2638	4.5674	0.3022	-0.0714
tse04.80m	1.5251	0.387	38.5532	-1.5757	0.2824	-0.0675
tse04.100m	1.5091	0.4091	37.7072	-4.0219	0.2774	-0.0556
tse05.2m	–	–	–	–	–	–
tse06.10m	1.645	1.167	72.325	-6.6285	0.3054	0.0098
tse06.20m	1.6409	1.1091	67.6222	-8.4868	0.3065	-0.0236
tse06.30m	1.6618	1.0976	65.1159	-7.9865	0.3005	-0.058

Continued on next page

Table D.1 – continued from previous page

	Wind speed ($m\,s^{-1}$)		Direction (°)		TKE ($m^2\,s^{-2}$)	
Sonic	RMSE	Bias	RMSE	Bias	RMSE	Bias
tse06.40m	1.6868	1.0992	58.0681	-8.4298	0.2872	-0.0477
tse06.60m	1.8191	1.1137	53.994	-4.6614	0.2706	-0.0673
tse07.10m	1.7328	1.3693	83.5204	12.9962	0.3133	0.1658
tse07.20m	1.9219	1.3868	82.1411	-7.7453	0.2744	0.0459
tse08.10m	1.6321	1.0365	85.7103	5.1392	0.3549	0.1478
tse08.20m	1.7512	1.0234	84.6442	1.7205	0.3215	0.0343
tse09.10m	1.2976	0.8051	104.7911	31.6807	0.4174	0.2107
tse09.20m	1.2499	0.5246	102.776	16.2447	0.3674	0.0424
tse09.30m	1.3168	0.5415	100.3732	16.6588	0.3775	0.0265
tse09.40m	1.3937	0.511	103.8645	15.6841	0.387	0.0232
tse09.60m	1.5381	0.6383	95.1257	13.3292	0.3862	0.0449
tse09.80m	1.7141	0.8172	85.3922	-4.5885	0.4107	0.028
tse09.100m	1.7625	0.7937	71.3758	-4.3233	0.4407	0.0181
tse10.10m	1.5875	1.352	83.5812	4.5578	0.5579	0.3196
tse10.30m	1.506	0.9104	108.1202	-1.0583	0.4456	0.0325
tse11.10m	1.7288	1.4282	91.032	2.8012	0.421	0.1569
tse11.20m	1.667	1.1971	101.6059	16.9374	0.4168	0.0068
tse11.30m	1.6682	0.9557	103.9094	21.7575	0.4701	-0.0724
tse11.40m	1.7086	0.7859	95.2998	13.475	0.4946	-0.0889
tse11.60m	1.7321	0.6728	77.5482	3.4226	0.4892	-0.0807
tse12.10m	1.9705	1.6874	76.4335	-6.2916	0.4487	0.2612
tse12.20m	1.5746	1.0079	76.6731	-7.1249	0.4233	-0.0663
tse13.10m	2.026	1.6421	46.5217	-2.4237	0.2485	-0.0541
tse13.20m	1.6104	0.8338	42.0506	-1.0371	0.2549	-0.0791
tse13.30m	1.637	0.6763	40.063	-3.9753	0.2714	-0.0801
tse13.40m	1.6632	0.6397	37.7525	-1.2698	0.2725	-0.0779
tse13.60m	1.6994	0.6101	42.6498	1.0562	0.2857	-0.0834
tse13.80m	1.6546	0.5002	47.2839	-0.5487	0.2776	-0.0845
tse13.100m	1.6031	0.4301	45.585	-0.3413	0.2723	-0.0832
rsw01.10m	1.4213	0.8405	37.9613	6.4987	0.2794	0.0085
rsw01.20m	1.4113	0.7231	31.4073	5.5773	0.2669	-0.0395
rsw02.10m	2.4016	2.0963	33.0303	-0.9516	0.4725	-0.0747
rsw02.20m	1.4112	0.7283	32.6482	1.2442	0.2668	-0.0468
rsw03.10m	1.5327	0.7885	33.0657	0.2424	0.244	-0.0167
rsw03.20m	1.3987	0.6549	40.1483	3.4142	0.2466	-0.0505
rsw03.30m	1.367	0.611	50.1397	8.1597	0.2458	-0.0598
rsw03.40m	1.4082	0.5227	52.5282	2.1832	0.2577	-0.0711
rsw03.60m	1.4534	0.4536	51.8243	-0.4603	0.2637	-0.07
rsw04.10m	1.3971	0.4864	55.3623	15.4884	0.2689	-0.0037
rsw05.10m	1.6621	0.9451	35.4847	8.2602	0.2841	-0.044
rsw06.10m	1.5972	0.6651	33.694	4.4193	0.2648	-0.0394
rsw06.20m	1.6046	0.6465	39.2033	4.1046	0.2855	-0.0993
rsw06.30m	1.5541	0.5537	43.1869	2.269	0.2929	-0.1134

Continued on next page

Table D.1 – continued from previous page

	Wind speed ($m\,s^{-1}$)		Direction (°)		TKE ($m^2\,s^{-2}$)	
Sonic	RMSE	Bias	RMSE	Bias	RMSE	Bias
rsw06.40m	1.5282	0.4674	42.9613	0.5369	0.2918	-0.1241
rsw06.60m	1.5324	0.3994	42.7473	0.845	0.268	-0.1128
rsw07.10m	1.7277	0.8067	39.0964	7.8816	0.2873	-0.0605
rsw07.20m	1.7367	0.8089	45.0852	4.7887	0.2696	-0.0747
rsw08.10m	1.6733	0.704	43.5488	1.1942	0.2782	-0.0798
rsw08.20m	1.6102	0.6906	46.5572	2.1539	0.2521	-0.0896
rne01.10m	1.5964	1.1715	38.8528	-1.3085	0.2525	-0.0114
rne02.10m	1.9796	1.613	35.4374	-4.114	0.256	0.0103
rne02.20m	1.5083	0.6903	36.6399	0.7302	0.2643	-0.0368
rne03.10m	1.4928	0.7766	45.4237	-1.27	0.2096	0.0233
rne04.10m	–	–	–	–	–	–
rne06.10m	1.6699	1.0222	38.944	-3.1418	0.2454	0.0194
rne06.20m	1.6393	0.8802	39.7711	-2.6702	0.2463	-0.0386
rne07.10m	1.3847	0.5721	34.4711	-0.3243	0.2793	-0.0122
rne07.20m	–	–	–	–	–	–
v01.2m	0.6184	0.2568	91.199	20.3947	0.2962	0.1737
v01.10m	1.2585	0.9257	97.0822	28.6498	0.4038	0.2083
v03.2m	0.7499	0.4848	109.5798	18.7027	0.3323	0.2062
v03.10m	1.3297	0.936	107.9301	25.4241	0.3961	0.1643
v04.2m	0.7245	0.534	98.1991	42.4278	0.3835	0.2527
v04.10m	1.1167	0.4877	91.3459	21.9821	0.4569	0.1463
v05.2m	0.7868	0.5952	80.5646	21.5918	0.3858	0.2669
v05.10m	1.3064	0.9763	84.1664	24.4405	0.4744	0.2369
v05.20m	1.3359	0.3977	91.3138	30.1482	0.4466	0.0939
v06.2m	0.9918	0.8045	79.3906	-6.7436	0.3662	0.2656
v06.10m	1.2484	0.5999	87.8377	32.2092	0.3529	0.1042
v06.20m	1.423	0.4416	86.2747	29.6245	0.3663	0.0442
v07.2m	0.9135	0.7365	84.1561	36.6378	0.4085	0.2597
v07.4m	1.0537	0.8097	82.0997	35.9378	0.4227	0.2191
v07.6m	1.1886	0.8933	84.3378	36.9759	0.4532	0.2031
v07.8m	1.2159	0.8877	84.688	33.7754	0.4399	0.2001
v07.10m	1.2165	0.7976	84.0199	37.7584	0.4331	0.1547
v07.12m	1.2152	0.7193	84.5459	29.8196	0.4263	0.1156
v07.20m	1.2393	0.4384	86.3765	29.2141	0.4021	0.0624
Extreme_SW	–	–	–	–	–	–
Extreme_NE	–	–	–	–	–	–

Table D.2: Wind speed ($m\,s^{-1}$) RMSE and Bias, from 18:00 May 14 to 18:00 May 15, 2017, for all sonic anemometers, calculated in 6-hour intervals.

	0-6h		6-12h		12-18h		18-24h	
Sonic	RMSE	Bias	RMSE	Bias	RMSE	Bias	RMSE	Bias
tnw01.2m	0.493	0.1295	1.0529	0.6556	0.9286	0.8205	1.2762	1.2192
tnw01.10m	1.0788	0.7084	1.7446	1.2773	0.9398	0.4695	1.7108	1.5987
tnw01.20m	1.2144	0.7345	1.7702	1.4048	1.1505	-0.1612	1.7565	1.5764
tnw02.2m	1.4563	1.3667	1.0954	1.0003	0.637	0.4618	1.7697	1.6815
tnw02.10m	2.1621	1.9965	1.7454	1.5527	0.6576	0.4322	2.5389	2.4045
tnw02.20m	2.2232	1.9279	1.8825	1.4138	0.5535	-0.1145	2.2982	2.1112
tnw03.2m	2.3885	2.137	1.0735	0.6997	1.4826	1.3384	1.1868	1.067
tnw03.10m	2.5782	2.1112	1.3233	-0.5763	0.9226	0.6194	1.5913	1.2788
tnw04.2m	1.1874	0.8304	0.4209	0.1351	0.9911	0.8751	1.3054	1.2429
tnw04.4m	1.4579	1.1135	0.5073	0.3164	1.0542	0.9273	1.4686	1.3679
tnw04.6m	1.6277	1.3036	0.5552	0.3911	1.0869	0.939	1.5558	1.42
tnw04.8m	1.7722	1.489	0.5451	0.3836	0.9928	0.8298	1.5223	1.3417
tnw04.10m	1.8401	1.5641	0.5347	0.3424	0.9996	0.8119	1.5201	1.31
tnw04.12m	1.8761	1.6196	0.5759	0.3877	1.0215	0.8385	1.5061	1.2955
tnw04.20m	1.9049	1.6305	0.6233	0.3054	1.0935	0.8294	1.3822	1.1082
tnw05.2m	1.1766	1.0018	–	–	–	–	1.1704	1.1105
tnw05.4m	1.318	1.1144	–	–	–	–	1.372	1.3017
tnw05.6m	1.5285	1.2707	–	–	–	–	1.5887	1.4858
tnw05.8m	1.4623	1.072	–	–	–	–	1.556	1.4103
tnw05.10m	1.5819	1.2113	–	–	–	–	1.5691	1.4017
tnw05.12m	1.6791	1.3195	–	–	–	–	1.5524	1.3573
tnw05.20m	2.0798	1.7266	–	–	–	–	1.4519	1.2103
tnw06.2m	0.7043	0.5511	0.7466	0.6331	1.0849	0.8044	1.0666	0.9488
tnw06.4m	1.0626	0.8619	0.7611	0.5716	1.0701	0.621	1.1835	1.0159

Continued on next page

Table D.2 – continued from previous page

	0-6h		6-12h		12-18h		18-24h	
Sonic	RMSE	Bias	RMSE	Bias	RMSE	Bias	RMSE	Bias
tnw06.6m	1.5526	1.2582	0.8707	0.633	1.1879	0.6363	1.3711	1.1748
tnw06.8m	1.8308	1.4861	0.8819	0.6071	1.1578	0.5249	1.3929	1.1893
tnw06.10m	2.0845	1.6909	0.9151	0.6149	1.1656	0.4206	1.381	1.1702
tnw06.12m	2.2026	1.7775	0.8998	0.5643	1.1694	0.4085	1.3842	1.1718
tnw06.20m	2.5021	2.0024	0.911	0.4579	1.235	0.4488	1.3633	1.1595
tnw07.4m	1.3886	1.0996	1.0157	0.8684	1.3515	1.0907	1.3262	1.2367
tnw07.6m	1.5616	1.1759	0.9949	0.791	1.4062	1.0578	1.4521	1.3337
tnw07.8m	1.6238	1.1392	0.8642	0.5831	1.3164	0.8418	1.4358	1.2723
tnw07.10m	1.6788	1.0637	0.7536	0.3185	1.2553	0.6323	1.4443	1.2237
tnw07.12m	1.6998	1.0047	0.7241	0.1906	1.2237	0.4258	1.4801	1.2364
tnw07.20m	1.7711	0.7692	0.8859	-0.4701	1.1494	0.1407	1.4361	1.1056
tnw07.30m	1.8595	0.7729	0.9297	-0.5728	1.0633	-0.1336	1.3661	1.0249
tnw07.40m	1.9514	0.9344	0.8233	-0.495	1.0144	-0.3129	1.3018	0.9196
tnw07.60m	2.2798	1.404	0.6057	-0.2763	1.0921	-0.5009	1.2563	0.8224
tnw08.2m	1.0445	0.8641	0.4121	0.2408	0.7228	0.5235	1.0205	0.8717
tnw08.10m	2.1104	1.8162	0.7547	0.2693	0.7915	-0.032	1.5011	1.1708
tnw08.20m	2.4246	2.0611	1.2185	0.0459	1.1221	-0.5955	1.5658	1.17
tnw09.2m	0.5218	0.227	0.6436	-0.0843	0.4188	0.1972	0.5188	0.3895
tnw09.10m	1.8108	1.6082	1.1398	0.2458	0.6704	-0.0374	1.1238	0.7994
tnw10.10m	1.8942	1.6665	1.3619	1.1045	1.3102	0.964	0.5882	0.3403
tnw10.20m	2.2509	1.9212	0.8047	0.1407	1.2705	0.7448	0.726	0.3557
tnw10.30m	2.5242	2.1679	0.5459	-0.1282	0.9564	0.2502	0.906	0.5594
tnw10.40m	2.71	2.3568	0.5812	-0.1223	0.8499	0.0106	1.0402	0.7152
tnw10.60m	2.8679	2.4728	0.7968	-0.2136	0.803	-0.3314	1.1437	0.8571
tnw11.2m	1.2264	1.0944	1.2921	1.1337	2.5819	2.5468	1.6776	1.643
tnw11.10m	1.7235	1.4266	0.5264	-0.1858	0.7744	0.3733	1.7429	1.5696

Continued on next page

Table D.2 – continued from previous page

	0-6h		6-12h		12-18h		18-24h	
Sonic	RMSE	Bias	RMSE	Bias	RMSE	Bias	RMSE	Bias
tnw11.20m	2.1705	1.8093	0.5538	-0.2542	0.6956	0.0466	1.6017	1.3389
tnw12.2m	0.8948	0.7654	1.018	0.9066	1.9421	1.9217	1.6235	1.5646
tnw12.4m	1.1164	0.9463	0.8463	0.7122	1.6063	1.5752	1.6076	1.5117
tnw12.6m	1.2419	1.0376	0.7203	0.5392	1.4518	1.4004	1.6747	1.5482
tnw12.8m	1.3139	1.0894	0.6123	0.3763	1.2548	1.1736	1.6357	1.4766
tnw12.10m	1.4118	1.1638	0.566	0.3009	1.1513	1.0421	1.6096	1.4236
tnw12.12m	1.5692	1.2883	0.5376	0.2298	1.0452	0.8936	1.5957	1.3826
tnw12.20m	2.2169	1.7981	0.5362	0.0833	0.907	0.6336	1.5429	1.2678
tnw12.30m	2.2862	1.8978	0.5223	-0.0241	0.7732	0.3782	1.4528	1.1559
tnw13.2m	0.2844	-0.1816	0.3902	-0.0483	0.6568	0.6353	0.6009	0.5644
tnw13.4m	0.3157	0.0287	0.6935	0.3503	1.474	1.4545	1.3793	1.3333
tnw13.6m	0.5449	0.3996	0.8378	0.6015	1.8105	1.7919	1.6841	1.6428
tnw13.8m	0.811	0.6908	0.9622	0.7676	2.1967	2.1803	1.9414	1.9026
tnw13.10m	0.8289	0.7117	0.9638	0.7751	1.9901	1.9657	1.965	1.9193
tnw13.12m	0.7871	0.6662	0.9338	0.7222	1.8466	1.8127	1.8559	1.7965
tnw13.20m	0.6616	0.4153	0.6072	0.3046	1.2229	1.0876	1.4517	1.2602
tnw13.30m	0.8748	0.5963	0.6228	0.2266	0.9877	0.7298	1.335	1.0774
tnw14.2m	0.7151	0.2369	0.4835	-0.2672	0.9937	0.9728	1.0272	0.9575
tnw14.4m	0.8714	0.5507	0.3627	0.0128	1.1393	1.1204	1.1669	1.1119
tnw14.6m	1.2356	0.9237	0.4463	0.1946	1.5366	1.5184	1.488	1.4383
tnw14.8m	1.3452	0.9985	0.4902	0.1869	1.6708	1.6533	1.5982	1.5462
tnw14.10m	1.5121	1.1607	0.5321	0.278	1.6851	1.6649	1.6535	1.5997
tnw14.12m	1.4824	1.1319	0.5509	0.2953	1.5526	1.5202	1.5975	1.5217
tnw14.20m	1.113	0.8804	0.5596	0.2778	1.0811	0.8956	1.3307	1.1203
tnw14.30m	0.9297	0.5528	0.5537	0.1413	0.9871	0.6811	1.2102	0.9064
tnw15.2m	0.8798	0.365	0.527	-0.3927	1.0304	1.0078	0.9509	0.8866

Continued on next page

Table D.2 – continued from previous page

	0-6h		6-12h		12-18h		18-24h	
Sonic	RMSE	Bias	RMSE	Bias	RMSE	Bias	RMSE	Bias
tnw15.4m	1.151	0.7034	0.4004	-0.1984	1.1147	1.0911	1.1383	1.0807
tnw15.6m	1.6619	1.2479	0.3464	0.0523	1.1312	1.1021	1.3257	1.2679
tnw15.8m	1.8588	1.4781	0.4002	0.2114	1.0172	0.9645	1.3759	1.2926
tnw15.10m	1.9033	1.5202	0.4421	0.2455	0.986	0.9037	1.3941	1.2837
tnw15.12m	1.8431	1.4471	0.4683	0.2414	0.9685	0.8494	1.3801	1.2438
tnw15.20m	1.5873	1.2335	0.5388	0.1781	0.9978	0.7506	1.2856	1.0691
tnw15.30m	1.4201	1.0245	0.5757	0.0742	1.0051	0.6618	1.2441	0.9611
tnw16.2m	0.7796	0.367	0.408	-0.3526	0.512	0.4653	0.5951	0.5102
tnw16.4m	1.243	0.7171	0.5052	-0.39	0.8695	0.8071	0.9722	0.8732
tnw16.6m	1.6673	1.1188	0.42	-0.2284	1.0238	0.9408	1.1802	1.0701
tnw16.8m	2.0111	1.5413	0.3127	0.0323	1.0154	0.9065	1.2532	1.1207
tnw16.10m	2.2463	1.8024	0.376	0.2126	0.9916	0.8572	1.2887	1.1281
tnw16.12m	2.3467	1.8903	0.4106	0.2099	0.9969	0.8262	1.2981	1.1085
tnw16.20m	2.3677	1.8395	0.5424	0.1176	1.0159	0.7334	1.2864	1.0058
tnw16.30m	2.1103	1.572	0.6492	0.0147	0.9982	0.6083	1.2505	0.8942
tse01.10m	0.7792	0.2287	1.6298	1.1564	1.9557	1.6203	1.8953	1.7411
tse01.30m	1.1549	0.6059	2.1583	1.1796	1.7036	0.6703	1.9628	1.738
tse02.10m	1.5808	1.3915	1.6165	1.4945	0.9438	0.6709	2.0652	1.9509
tse02.30m	1.8898	1.6093	0.9936	0.6274	1.0725	-0.4465	2.0043	1.7972
tse04.10m	2.2285	1.7842	1.2688	-0.4552	0.9976	0.5873	0.6974	0.2553
tse04.20m	2.2755	1.8999	1.3584	-0.6023	0.6173	0.1968	0.9426	0.4813
tse04.30m	2.2218	1.8569	1.3602	-0.6724	0.6032	0.1736	1.103	0.659
tse04.40m	2.2448	1.838	1.4153	-0.8057	0.5589	0.1065	1.2044	0.7554
tse04.60m	2.2664	1.7999	1.4566	-0.987	0.5664	-0.0714	1.2719	0.8537
tse04.80m	2.2693	1.7774	1.3758	-0.9851	0.7247	-0.1701	1.3174	0.9257
tse04.100m	2.2263	1.7087	1.2765	-0.9261	0.8184	-0.1378	1.3614	0.9916

Continued on next page

Table D.2 – continued from previous page

	0-6h		6-12h		12-18h		18-24h	
Sonic	RMSE	Bias	RMSE	Bias	RMSE	Bias	RMSE	Bias
tse05	–	–	–	–	–	–	–	–
tse06.10m	2.1494	1.7053	0.6876	0.2491	1.8666	1.452	1.499	1.2615
tse06.20m	2.0622	1.6078	0.7635	0.2652	1.9064	1.3403	1.5166	1.223
tse06.30m	2.1165	1.6312	0.8287	0.3048	1.8982	1.2521	1.5089	1.2021
tse06.40m	2.1611	1.6705	0.8965	0.3128	1.9216	1.2429	1.4882	1.1706
tse06.60m	2.5486	2.0031	1.0071	0.1454	1.9204	1.1971	1.428	1.109
tse07.10m	2.3286	1.8795	0.7196	0.453	1.9675	1.8114	1.4829	1.3333
tse07.20m	2.77	2.2129	0.7587	0.3671	2.0629	1.6797	1.5068	1.2875
tse08.10m	1.9737	1.2363	0.8777	0.0947	1.9794	1.5677	1.4393	1.2472
tse08.20m	2.5399	1.8366	0.7651	0.2359	1.8706	0.9904	1.3158	1.0307
tse09.10m	1.4036	0.6947	0.623	-0.093	1.408	1.1798	1.5473	1.439
tse09.20m	1.7416	0.7984	0.6322	-0.4132	0.9744	0.5393	1.3662	1.174
tse09.30m	2.0163	1.1138	0.5398	-0.3433	0.9497	0.309	1.2952	1.0864
tse09.40m	2.2013	1.1978	0.6515	-0.3352	0.9754	0.1475	1.2441	1.0337
tse09.60m	2.5337	1.7197	0.6467	-0.0622	1.0998	-0.0697	1.1897	0.9657
tse09.80m	2.9025	2.2209	0.672	0.2324	1.2184	-0.1242	1.1797	0.9396
tse09.100m	2.9339	2.2762	0.7941	0.1819	1.3437	-0.1613	1.1752	0.8781
tse10.10m	1.807	1.5584	0.7399	0.5807	1.56	1.3987	1.9581	1.87
tse10.30m	2.2474	1.9334	0.9961	0.2813	0.8077	0.1319	1.5419	1.2949
tse11.10m	2.313	2.1651	1.2435	0.8119	1.0756	0.9085	1.9754	1.8274
tse11.20m	2.4157	2.1964	1.1881	0.5322	0.7808	0.4695	1.8053	1.5904
tse11.30m	2.5122	2.2058	1.2231	0.159	0.7101	0.0472	1.6792	1.4107
tse11.40m	2.6148	2.2528	1.2373	-0.0574	0.89	-0.3253	1.5865	1.2734
tse11.60m	2.7131	2.2827	1.0486	-0.1239	1.2012	-0.5847	1.448	1.1173
tse12.10m	2.8242	2.6516	1.3733	1.1756	0.9528	0.8308	2.182	2.0915
tse12.20m	2.3723	2.0548	0.9143	0.3419	0.662	0.0814	1.7364	1.5536

Continued on next page

Table D.2 – continued from previous page

	0-6h		6-12h		12-18h		18-24h	
Sonic	RMSE	Bias	RMSE	Bias	RMSE	Bias	RMSE	Bias
tse13.10m	2.7823	2.3492	1.2748	0.7968	2.1544	2.0044	1.5526	1.4178
tse13.20m	2.7175	2.3244	0.6113	-0.1319	0.8484	-0.0239	1.3766	1.1665
tse13.30m	2.8047	2.3891	0.6892	-0.2622	0.8175	-0.4426	1.3074	1.021
tse13.40m	2.8374	2.4145	0.7805	-0.3257	0.8425	-0.5372	1.3021	1.007
tse13.60m	2.8544	2.3986	0.8987	-0.376	0.9949	-0.584	1.2676	1.0019
tse13.80m	2.6986	2.2135	0.9541	-0.5584	1.0598	-0.6842	1.2785	1.03
tse13.100m	2.559	2.0764	0.9735	-0.6849	1.0636	-0.7086	1.2853	1.0374
rsw01.10m	1.8364	1.5757	1.1771	-0.3534	1.0277	0.8276	1.5053	1.3121
rsw01.20m	1.8917	1.5892	1.2227	-0.5208	0.7818	0.4919	1.5108	1.3322
rsw02.10m	1.891	1.6373	1.796	1.5155	3.5508	3.4858	1.9133	1.7464
rsw02.20m	1.8301	1.5731	1.2316	-0.5337	0.7912	0.5025	1.573	1.3713
rsw03.10m	2.4621	2.091	1.206	-0.3585	0.8129	0.5071	1.1041	0.9142
rsw03.20m	2.13	1.8207	1.2079	-0.463	0.6639	0.3378	1.1785	0.9241
rsw03.30m	2.0392	1.7271	1.2089	-0.5619	0.6337	0.3296	1.2057	0.9492
rsw03.40m	2.0584	1.6809	1.3483	-0.7919	0.594	0.2322	1.2346	0.9694
rsw03.60m	2.1153	1.6608	1.4089	-0.9256	0.6396	0.0949	1.2574	0.9842
rsw04.10m	2.0825	1.7277	1.4383	-0.6279	0.8633	0.331	0.8101	0.5149
rsw05.10m	2.4202	1.951	1.2129	-0.3376	1.3497	1.0252	1.3784	1.1417
rsw06.10m	2.425	1.9415	1.4036	-0.6886	0.8007	0.4454	1.3085	0.9622
rsw06.20m	2.4519	1.9391	1.3483	-0.6834	0.7951	0.3682	1.3555	0.9621
rsw06.30m	2.3522	1.8386	1.3721	-0.802	0.7071	0.2693	1.3212	0.9088
rsw06.40m	2.2905	1.7957	1.4316	-0.9571	0.6145	0.1486	1.2915	0.8824
rsw06.60m	2.3079	1.8356	1.4584	-1.1203	0.5547	0.0039	1.2773	0.8783
rsw07.10m	2.7078	2.2911	1.4116	-0.6739	0.7491	0.4721	1.4332	1.1374
rsw07.20m	2.7051	2.2255	1.3708	-0.6942	0.7368	0.5116	1.5249	1.1928
rsw08.10m	2.3362	1.9834	1.6396	-0.8895	0.8033	0.502	1.5519	1.22

Continued on next page

Table D.2 – continued from previous page

	0-6h		6-12h		12-18h		18-24h	
Sonic	RMSE	Bias	RMSE	Bias	RMSE	Bias	RMSE	Bias
rsw08.20m	2.2049	1.8439	1.5095	-0.843	0.7498	0.452	1.6334	1.3095
rne01.10m	2.0675	1.6883	0.8765	0.2192	1.2573	0.9717	1.8895	1.8068
rne02.10m	2.3294	1.9409	1.5384	0.8402	2.2736	2.1286	1.6471	1.5423
rne02.20m	2.4657	2.0472	0.7776	-0.186	0.8718	-0.1115	1.2867	1.0114
rne03.10m	2.3557	1.9471	0.7872	-0.1849	0.8824	0.1507	1.4021	1.1934
rne04.10m	–	–	–	–	–	–	–	–
rne06.10m	2.5037	2.1025	0.6999	-0.2274	0.8398	0.4815	1.9212	1.7323
rne06.20m	2.6129	2.1838	0.6295	-0.271	0.6208	0.0776	1.772	1.5302
rne07.10m	2.1881	1.8312	0.7329	-0.4392	0.769	-0.1732	1.3243	1.0694
rne07.20m	–	–	–	–	–	–	–	–
v01.2m	0.4722	0.1313	0.5535	-0.1256	0.6271	0.4528	0.7793	0.5686
v01.10m	1.5901	1.2422	0.7245	0.4177	0.912	0.7203	1.5654	1.3227
v03.2m	0.9744	0.6573	0.4752	0.1619	0.8507	0.63	0.5921	0.49
v03.10m	1.73	1.2689	0.8129	0.4846	1.4215	0.9726	1.1824	1.0179
v04.2m	0.6058	0.4187	0.4067	0.2877	0.8572	0.6508	0.9123	0.7787
v04.10m	1.1801	0.3576	0.5855	-0.3389	1.0785	0.7196	1.4454	1.2125
v05.2m	0.6461	0.4646	0.6073	0.4288	0.8109	0.6299	1.0161	0.8577
v05.10m	1.3802	0.9134	0.7233	0.5162	1.1802	0.9688	1.7339	1.5069
v05.20m	1.744	0.5021	0.8746	-0.5326	0.9653	0.4267	1.5493	1.1948
v06.2m	0.9338	0.7044	0.7962	0.6387	1.0733	0.858	1.1299	1.0169
v06.10m	1.5079	0.7086	0.7487	-0.0917	1.1686	0.6378	1.4263	1.145
v06.20m	1.9056	0.6259	0.9706	-0.4748	1.1638	0.4657	1.4737	1.1498
v07.2m	1.1319	0.94	0.7996	0.6469	0.7972	0.5961	0.8841	0.7632
v07.4m	1.3466	1.1071	0.9395	0.7523	0.8505	0.5568	1.0108	0.8227
v07.6m	1.5208	1.2292	1.0338	0.8045	0.9643	0.6141	1.1575	0.9254
v07.8m	1.5567	1.1859	0.9476	0.6703	0.995	0.6489	1.2657	1.0456

Continued on next page

Table D.2 – continued from previous page

	0-6h		6-12h		12-18h		18-24h	
Sonic	RMSE	Bias	RMSE	Bias	RMSE	Bias	RMSE	Bias
v07.10m	1.5805	1.1367	0.8703	0.5372	0.9692	0.5099	1.3134	1.0068
v07.12m	1.6004	1.0912	0.8262	0.4327	0.9445	0.3852	1.3307	0.9681
v07.20m	1.6203	0.7908	0.8	-0.2287	0.9303	0.1731	1.4186	1.0185
EXTREME_SW	–	–	–	–	–	–	–	–
EXTREME_NE	–	–	–	–	–	–	–	–

Table D.3: Wind direction (°) RMSE and Bias, from 18:00 May 14 to 18:00 May 15, 2017, for all sonic anemometers, calculated in 6-hour intervals.

	0-6h		6-12h		12-18h		18-24h	
Sonic	RMSE	Bias	RMSE	Bias	RMSE	Bias	RMSE	Bias
tnw01.2m	104.0633	-27.3873	104.9079	32.0203	96.7776	9.6054	90.7996	52.0595
tnw01.10m	90.4001	-3.0739	85.5194	31.9593	93.5816	26.5222	91.6074	37.7264
tnw01.20m	86.5331	-6.2941	74.6052	-6.4351	91.5173	2.6233	86.3709	43.1548
tnw02.2m	74.6936	-41.7426	87.7915	-44.5269	60.1099	42.3677	85.3974	79.6172
tnw02.10m	45.1217	8.9729	92.87	-22.3995	72.4969	52.4497	83.3341	64.5036
tnw02.20m	44.174	-9.9425	70.4409	-2.6707	100.8758	42.3003	90.7054	74.8727
tnw03.2m	71.9156	-19.3045	30.7804	25.0837	18.083	13.5438	33.6995	17.6277
tnw03.10m	44.602	19.9276	35.0981	21.5623	10.694	-3.3728	34.4086	-13.7946
tnw04.2m	106.2856	78.1313	66.731	-52.0386	68.0434	-11.403	85.9076	-3.2188
tnw04.4m	92.5854	62.0809	65.8915	-36.4331	71.0939	-18.7313	88.8591	-26.4895
tnw04.6m	83.4952	47.732	69.0844	-29.9273	70.6111	-31.3597	88.1831	-34.7136
tnw04.8m	75.1662	30.3011	70.5384	-21.3891	64.9066	-15.9442	88.596	-38.9455
tnw04.10m	74.888	25.8983	71.9519	-23.3244	64.591	-27.2417	90.6345	-36.9815

Continued on next page

Table D.3 – continued from previous page

	0-6h		6-12h		12-18h		18-24h	
Sonic	RMSE	Bias	RMSE	Bias	RMSE	Bias	RMSE	Bias
tnw04.12m	68.9787	21.0671	75.8032	-4.3408	65.7638	-34.0505	95.3293	-20.5656
tnw04.20m	55.523	7.9191	68.1671	6.3476	53.9233	-32.8454	89.5012	-31.5145
tnw05.2m	90.9887	58.5609	–	–	–	–	88.1891	20.9846
tnw05.4m	93.0202	61.1002	–	–	–	–	94.8832	30.2372
tnw05.6m	91.4164	56.8149	–	–	–	–	93.178	6.3266
tnw05.8m	87.4321	54.6267	–	–	–	–	97.6467	-6.9959
tnw05.10m	78.0772	37.6879	–	–	–	–	95.639	-20.7957
tnw05.12m	75.2989	27.3563	–	–	–	–	94.7828	-23.8431
tnw05.20m	64.8954	-5.0496	–	–	–	–	96.7861	-35.3401
tnw06.2m	80.1937	-4.032	72.698	-59.6618	55.1702	-18.8593	81.9873	-34.5329
tnw06.4m	79.6962	10.1535	73.0967	-55.9727	54.3399	-20.1191	76.975	-28.5534
tnw06.6m	86.1782	25.0518	69.5432	-45.9333	53.5894	-18.591	78.531	-22.9819
tnw06.8m	93.343	29.0043	63.0675	-21.9194	50.2661	-19.701	82.1657	-24.8665
tnw06.10m	89.0911	27.7185	64.7833	-6.1023	47.3326	-17.1605	83.0107	-28.0895
tnw06.12m	88.1245	22.3989	61.5163	-17.275	48.2347	-18.7829	87.6354	-29.8186
tnw06.20m	89.2106	-15.9048	43.4366	-9.4259	49.829	-15.6518	92.4302	-30.8482
tnw07.4m	103.825	25.3836	50.6089	24.7462	81.6924	31.3935	76.6466	26.1753
tnw07.6m	107.414	20.9091	42.2298	29.5101	81.2892	36.2707	81.9464	27.5332
tnw07.8m	108.6655	6.963	41.368	31.0656	82.6083	33.0959	83.6927	31.252
tnw07.10m	111.5887	-10.9669	40.3748	30.825	83.1915	23.6192	87.9005	34.9516
tnw07.12m	112.2505	-16.0853	41.5814	32.7847	87.9771	26.4053	91.041	36.0311
tnw07.20m	117.0402	20.561	37.0931	27.4399	83.1546	25.3562	93.236	22.3952
tnw07.30m	119.1959	44.5108	32.5537	20.2838	85.553	8.9037	91.2405	6.1185
tnw07.40m	127.1079	45.3076	32.9568	20.5884	83.1454	18.1717	93.4642	5.1101
tnw07.60m	124.6613	48.7606	34.3599	15.3265	70.2201	2.414	97.1415	-15.1736
tnw08.2m	95.1388	-35.2281	50.9653	34.0093	83.4825	59.644	79.5343	22.4381

Continued on next page

Table D.3 – continued from previous page

	0-6h		6-12h		12-18h		18-24h	
Sonic	RMSE	Bias	RMSE	Bias	RMSE	Bias	RMSE	Bias
tnw08.10m	100.7629	-39.5043	65.5723	29.7989	110.1357	65.4317	95.824	18.5053
tnw08.20m	110.4692	11.1917	74.1402	32.1183	101.3554	39.5383	94.8467	6.9289
tnw09.2m	85.301	-33.2078	88.5236	60.6663	107.2755	61.9861	94.4453	-6.123
tnw09.10m	48.3857	-13.4031	54.4033	38.9991	106.1408	58.4939	102.7009	-0.1474
tnw10.10m	45.6669	-8.0674	22.1296	7.2705	68.1966	10.41	98.9245	-25.5551
tnw10.20m	39.0659	-5.4806	25.4952	12.8885	38.2828	5.1299	96.8229	-9.6586
tnw10.30m	36.35	1.8112	25.499	15.0924	30.6834	-2.6208	89.2631	-6.6059
tnw10.40m	34.58	4.1829	24.8174	15.9186	24.6792	-7.2624	74.1854	-10.2119
tnw10.60m	32.9069	9.9945	33.4981	21.199	22.8062	-9.1289	65.439	-13.7164
tnw11.2m	65.3813	-7.9426	17.0193	6.3466	13.8593	-2.0305	41.2852	-15.5467
tnw11.10m	40.5796	-3.082	30.2238	15.9396	18.8875	-5.0152	43.3082	-11.397
tnw11.20m	46.1714	-4.5681	28.0946	10.8142	22.8662	-10.2316	57.4523	-7.6869
tnw12.2m	82.1916	-1.6441	65.6448	-12.0049	20.6974	-14.2431	50.4543	-8.7915
tnw12.4m	64.6884	-14.8651	47.1559	13.0221	20.7372	-11.8912	75.6694	-1.9175
tnw12.6m	63.3603	-11.6919	41.9076	17.5584	22.0286	-9.8615	58.2985	-9.8369
tnw12.8m	58.3563	-12.3193	42.5098	21.1874	23.8684	-9.9501	73.5046	-5.7248
tnw12.10m	56.0699	-10.4618	43.3518	21.6673	25.2503	-10.9522	66.6944	-17.7921
tnw12.12m	54.9986	-15.4692	41.8198	17.508	28.1912	-15.5991	66.3444	-20.5287
tnw12.20m	46.3279	-10.5075	42.4808	18.367	27.6833	-13.4597	62.1529	-25.1209
tnw12.30m	42.415	-7.8252	38.5999	14.762	27.9351	-13.8114	59.0214	-24.7961
tnw13.2m	61.8841	-38.17	112.1196	-102.3655	52.9833	47.2706	88.2225	18.5754
tnw13.4m	80.247	12.1272	120.2187	-57.3527	25.5106	-9.094	82.0655	-4.1383
tnw13.6m	84.1447	4.6346	112.2412	-49.8384	26.977	-12.8416	81.2327	-12.1399
tnw13.8m	83.4405	5.3114	99.0487	-25.9627	32.2174	-18.5669	79.6536	-23.8983
tnw13.10m	78.5327	7.2166	90.826	-13.3544	32.6537	-21.0918	76.9495	-20.4165
tnw13.12m	65.8191	10.6046	88.8112	7.4024	33.1698	-16.3401	75.4122	-13.0078

Continued on next page

Table D.3 – continued from previous page

	0-6h		6-12h		12-18h		18-24h	
Sonic	RMSE	Bias	RMSE	Bias	RMSE	Bias	RMSE	Bias
tnw13.20m	63.3506	7.4831	69.5064	22.8652	37.8204	-8.3664	74.1563	-11.1308
tnw13.30m	63.2168	-12.9863	73.2162	43.5726	39.2676	-17.1298	69.0391	-17.8206
tnw14.2m	75.9477	-2.6918	118.2783	4.5153	63.9825	-38.3624	101.7828	23.408
tnw14.4m	79.2453	25.9911	109.9366	-27.244	48.3079	-11.2992	99.0302	4.0944
tnw14.6m	80.8166	25.5464	114.4401	-23.4668	52.1904	-10.1675	89.2251	11.2218
tnw14.8m	80.2941	21.367	117.5042	-36.7511	57.3853	-11.3559	85.8392	17.2358
tnw14.10m	80.5647	23.3637	118.2211	-11.8224	51.5781	-11.1327	86.5702	10.5762
tnw14.12m	75.4854	21.3436	116.1939	4.8387	51.8663	-2.3006	82.8907	3.1152
tnw14.20m	71.5576	6.559	99.881	34.8218	65.1101	-13.5097	68.0255	-1.9022
tnw14.30m	72.6339	-6.6611	97.2434	38.2799	58.4188	-4.1778	74.8292	-12.824
tnw15.2m	83.3322	15.5761	102.2524	-47.1331	53.1964	-17.7379	109.8066	56.0079
tnw15.4m	80.2096	14.2185	102.2248	-43.4104	52.9877	-11.5174	93.8228	25.6618
tnw15.6m	81.6804	22.6045	111.7376	-12.3682	61.771	-12.5681	86.9132	38.9005
tnw15.8m	82.2071	21.6084	110.7593	-14.972	64.8	-4.1654	83.4445	40.7371
tnw15.10m	79.0771	15.3939	113.7565	-6.2585	67.7905	-6.9532	85.1104	22.5688
tnw15.12m	76.2507	3.1156	116.6838	9.8865	73.8798	-5.0481	83.6257	13.2074
tnw15.20m	76.983	-6.1049	116.5935	27.7933	73.9798	-17.0165	84.2047	2.6454
tnw15.30m	74.4041	8.4422	109.3448	50.6051	64.0095	-19.6918	78.6503	-1.9595
tnw16.2m	61.7719	-17.0056	106.8324	-47.5217	80.6565	-0.2605	103.0835	23.141
tnw16.4m	70.5409	1.1266	106.3353	-41.8396	80.5969	-13.0002	102.3351	19.2072
tnw16.6m	72.2678	1.3103	106.0005	-24.8048	76.3008	-23.9859	96.7981	-0.0525
tnw16.8m	71.1856	3.7623	110.1848	-27.9468	77.1211	-12.6859	91.6893	8.7896
tnw16.10m	68.1761	0.9042	114.9033	-5.4166	76.3423	-22.1459	91.1569	13.382
tnw16.12m	62.906	-2.2738	122.5365	12.9316	72.709	-13.4926	94.4006	2.8867
tnw16.20m	64.6414	3.7244	121.125	51.0459	69.3398	-23.2431	92.028	-0.2717
tnw16.30m	59.6015	-1.7316	108.7505	61.7594	67.2972	-11.7576	88.7793	-7.4352

Continued on next page

Table D.3 – continued from previous page

	0-6h		6-12h		12-18h		18-24h	
Sonic	RMSE	Bias	RMSE	Bias	RMSE	Bias	RMSE	Bias
tse01.10m	103.4662	-10.261	106.5134	10.2218	86.3709	-0.5756	85.8376	20.53
tse01.30m	82.5518	0.5372	102.2538	9.3878	70.2219	2.0163	87.2391	5.9486
tse02.10m	58.9021	-37.2143	34.0992	15.3944	86.8071	-29.8091	70.5785	40.5772
tse02.30m	50.3493	1.5543	46.148	30.5738	89.946	17.4917	90.4721	32.7352
tse04.10m	36.3516	14.9358	33.0466	10.2313	16.0833	-9.4794	71.776	-12.9705
tse04.20m	42.3319	23.478	38.865	19.8643	15.1012	-6.2851	51.1319	-17.7282
tse04.30m	45.6477	25.3301	42.5336	22.0596	15.6229	-7.6034	57.0459	-15.1596
tse04.40m	46.5871	13.8493	45.6101	23.7186	15.7807	-7.9689	51.7698	-8.1763
tse04.60m	29.9344	12.6485	49.9582	28.246	14.9799	-5.5987	53.5576	-17.0262
tse04.80m	20.5388	8.3041	53.8594	23.2831	15.1287	-4.6254	48.9268	-33.2647
tse04.100m	19.995	5.4654	53.7939	15.9121	15.221	-4.3562	46.4981	-33.109
tse05	—	—	—	—	—	—	—	—
tse06.10m	71.0373	2.9248	94.1169	-4.8196	54.077	-3.8335	63.9925	-20.7855
tse06.20m	69.3919	-5.3312	81.8727	-2.6868	51.9432	-2.8559	63.8325	-23.0733
tse06.30m	66.5583	3.4376	71.1664	-3.7823	51.1563	-3.3755	69.6327	-28.2257
tse06.40m	55.3238	-5.6578	66.4873	-1.0058	43.7415	-0.893	63.9766	-26.1628
tse06.60m	43.207	-2.4264	64.4094	20.446	40.7473	-1.908	63.132	-34.7572
tse07.10m	100.6294	59.501	98.6884	-16.9434	46.9554	9.8341	76.3684	-0.4068
tse07.20m	84.6499	0.8625	108.306	-14.0177	46.5976	1.1572	76.9512	-18.983
tse08.10m	99.6879	47.1667	85.4299	-17.5413	74.8405	-1.9553	80.9197	-7.1134
tse08.20m	98.5076	33.9609	96.2602	-4.9356	64.368	-0.4653	74.4686	-21.6779
tse09.10m	110.6604	42.4645	94.278	67.6795	113.8512	-5.611	99.1389	22.1899
tse09.20m	120.7573	56.0472	73.4006	44.3337	108.7615	-36.4706	102.238	1.0684
tse09.30m	121.335	71.9497	66.4008	23.0921	106.6486	-16.8271	98.9642	-11.5794
tse09.40m	131.743	78.1728	80.7752	24.058	97.3713	-2.2638	98.9409	-37.2306
tse09.60m	121.0256	83.7365	84.6087	6.1194	72.3177	-17.2972	95.7076	-19.2418

Continued on next page

Table D.3 – continued from previous page

	0-6h		6-12h		12-18h		18-24h	
Sonic	RMSE	Bias	RMSE	Bias	RMSE	Bias	RMSE	Bias
tse09.80m	94.0827	-1.7317	77.4477	24.4745	68.131	-25.2192	98.3654	-15.8777
tse09.100m	68.1697	5.1236	58.9108	31.4555	57.8884	-15.4108	94.3894	-38.4617
tse10.10m	92.496	-14.0476	86.6091	-8.9554	86.9998	23.2069	65.7088	18.0271
tse10.30m	118.1085	-36.5417	101.7421	28.5342	113.3662	4.0719	98.0148	-0.2977
tse11.10m	100.2859	-49.8831	87.184	8.7591	90.4266	43.4982	85.5103	8.8304
tse11.20m	103.9524	-1.5661	105.8558	13.3168	107.6039	53.1066	87.7777	2.8921
tse11.30m	103.9539	-8.0026	107.4256	40.8134	112.5513	48.8812	90.4113	5.3379
tse11.40m	95.5181	5.3126	84.6917	46.3498	106.6716	23.6905	93.0212	-21.4529
tse11.60m	63.7828	0.4062	60.0132	28.3568	84.5398	14.2604	96.115	-29.3331
tse12.10m	65.6285	-38.6771	61.1597	10.3786	78.059	4.2232	96.06	-1.0912
tse12.20m	42.0296	2.5843	32.6713	16.804	99.3564	-8.8556	103.9685	-39.0322
tse13.10m	40.7515	-3.9138	32.1643	15.1786	25.0435	-8.6071	73.0389	-12.3526
tse13.20m	36.9698	-1.896	30.5877	17.8487	22.1869	-8.5747	65.4093	-11.5265
tse13.30m	34.4703	0.5452	28.5657	18.4428	21.3258	-8.6477	62.938	-26.2415
tse13.40m	28.3552	5.6054	30.8827	21.3529	19.8768	-6.953	59.5665	-25.0843
tse13.60m	36.5577	12.2403	40.2997	24.9308	18.9621	-5.4231	62.8961	-27.5233
tse13.80m	37.3269	-0.608	48.4569	29.6829	19.5858	-5.7249	69.4125	-25.5447
tse13.100m	21.9419	6.6431	58.4349	31.4002	19.9359	-6.0065	63.3911	-33.402
rsw01.10m	37.3748	6.3546	13.473	3.3855	11.4636	-5.067	63.6743	21.3215
rsw01.20m	37.3057	19.127	24.7699	11.783	13.8494	-5.4328	41.8163	-3.1679
rsw02.10m	33.8056	10.1761	23.5199	7.7765	20.1539	-11.9562	47.5586	-9.8027
rsw02.20m	35.5268	16.1279	31.0795	13.2229	18.7715	-10.1626	41.0264	-14.2112
rsw03.10m	35.8885	17.3561	29.0211	11.9694	15.7572	-8.6293	44.664	-19.7267
rsw03.20m	37.9924	20.3385	35.6089	17.7758	16.1201	-6.8	58.96	-17.6576
rsw03.30m	45.0263	25.0368	41.3687	22.2625	16.8215	-6.8057	77.6804	-7.8549
rsw03.40m	46.8142	11.8126	45.9283	24.9149	17.5121	-7.4301	80.1823	-20.5646

Continued on next page

Table D.3 – continued from previous page

	0-6h		6-12h		12-18h		18-24h	
Sonic	RMSE	Bias	RMSE	Bias	RMSE	Bias	RMSE	Bias
rsw03.60m	35.8245	8.7861	53.6526	28.0702	17.1544	-5.8292	79.2892	-32.8684
rsw04.10m	39.7119	20.419	38.3426	19.3189	12.7203	-2.4051	95.1365	24.6207
rsw05.10m	46.1144	23.6612	38.8784	19.6393	11.1893	-5.8556	35.6844	-4.404
rsw06.10m	44.275	19.6047	31.7312	18.4911	10.6805	-3.261	38.209	-17.1579
rsw06.20m	45.3022	23.3911	39.1571	24.7305	12.2347	-5.009	49.1155	-26.6943
rsw06.30m	44.3294	23.3184	43.8686	26.7538	12.8224	-6.9596	58.3649	-34.0366
rsw06.40m	36.644	13.4966	47.2224	26.9024	13.907	-8.8422	60.1379	-29.4091
rsw06.60m	22.761	9.3677	59.1866	32.386	12.9641	-6.6582	55.8583	-31.7155
rsw07.10m	50.7702	27.598	41.2234	24.3026	15.2608	-7.2381	40.053	-13.1359
rsw07.20m	47.7326	28.6246	49.9009	30.29	14.4839	-5.6544	56.1465	-34.1052
rsw08.10m	51.7151	23.6287	42.3937	25.6841	17.7709	-10.3693	52.901	-34.1668
rsw08.20m	47.044	25.6062	51.3878	30.7652	16.6104	-9.8219	59.5025	-37.9339
rne01.10m	35.4842	-2.7893	38.6926	24.5999	22.6104	-2.3785	52.6372	-24.666
rne02.10m	38.712	-3.3908	32.5858	19.6114	19.4825	-6.7188	45.6424	-25.9578
rne02.20m	30.3564	4.6587	30.7691	21.4199	19.6784	-2.4505	55.8072	-20.7072
rne03.10m	40.1389	-4.9493	33.3848	16.0119	22.1867	-13.8205	70.9599	-2.3219
rne04.10m	–	–	–	–	–	–	–	–
rne06.10m	46.0464	-3.4819	32.4711	16.4982	19.6201	-9.3404	50.0693	-16.2431
rne06.20m	40.2266	-4.9833	32.5613	17.5648	20.3145	-8.5217	56.8847	-14.7405
rne07.10m	38.8302	3.2573	27.098	17.7763	19.5765	-2.3182	46.127	-20.0128
rne07.20m	–	–	–	–	–	–	–	–
v01.2m	102.6941	35.0528	62.9551	26.9433	107.6017	1.9751	84.7437	17.6075
v01.10m	120.078	61.667	64.9846	22.841	103.5871	19.2502	91.257	10.841
v03.2m	123.9245	85.744	105.9441	49.9426	116.7755	-50.2604	88.3913	-10.6155
v03.10m	117.0981	63.2129	97.3685	41.523	119.7263	-1.6066	95.2296	-1.433
v04.2m	110.9469	69.3291	77.4256	58.2298	105.0814	9.7965	96.0531	32.3559

Continued on next page

Table D.3 – continued from previous page

	0-6h		6-12h		12-18h		18-24h	
Sonic	RMSE	Bias	RMSE	Bias	RMSE	Bias	RMSE	Bias
v04.10m	108.7451	50.9751	59.3729	34.7943	98.2934	-3.3844	91.4552	5.5433
v05.2m	73.6134	36.597	57.8169	16.8339	100.8592	8.5721	83.8348	24.364
v05.10m	95.0936	59.5705	58.9824	21.9575	88.9187	-0.8561	88.925	17.0901
v05.20m	117.6741	62.2692	65.3864	32.5282	78.0552	10.4668	95.5909	15.3287
v06.2m	83.8769	20.0298	52.3942	-32.6878	87.2606	-16.6553	88.4113	2.3387
v06.10m	107.882	61.5997	42.2864	27.6952	94.6941	15.4498	92.023	24.0922
v06.20m	108.4185	69.2616	39.3403	22.9291	88.3445	-3.074	93.0929	29.3814
v07.2m	108.2528	48.5217	53.3309	27.6946	72.5947	22.5364	92.1746	47.7985
v07.4m	107.8505	61.8068	50.1593	15.7279	68.9322	18.1146	89.7893	48.102
v07.6m	114.9847	54.3574	47.1298	22.0286	67.2381	24.4755	92.1293	47.0421
v07.8m	116.4335	56.9809	44.4524	24.5257	67.1256	20.0535	93.0033	33.5416
v07.10m	115.8991	66.2972	37.0424	22.8325	62.0135	17.7981	97.9132	44.106
v07.12m	116.7246	51.7166	36.2177	21.1083	62.335	14.6621	98.8435	31.7914
v07.20m	118.1758	62.1522	25.1017	16.8862	68.438	14.7658	102.7825	23.052
EXTREME_SW	–	–	–	–	–	–	–	–
EXTREME_NE	–	–	–	–	–	–	–	–

Table D.4: Turbulence kinetic energy ($m^2\ s^{-2}$) RMSE and Bias, from 18:00 May 14 to 18:00 May 15, 2017, for all sonic anemometers, calculated in 6-hour intervals.

	0-6h		6-12h		12-18h		18-24h	
Sonic	RMSE	Bias	RMSE	Bias	RMSE	Bias	RMSE	Bias
tnw01.2m	0.1723	0.1091	0.2223	0.1207	0.7941	0.6244	0.5411	0.4618
tnw01.10m	0.1758	0.0688	0.4532	-0.2889	0.8997	0.4338	0.5853	0.1598

Continued on next page

Table D.4 – continued from previous page

	0-6h		6-12h		12-18h		18-24h	
Sonic	RMSE	Bias	RMSE	Bias	RMSE	Bias	RMSE	Bias
tnw01.20m	0.1801	0.0634	0.6614	-0.3936	0.7813	0.087	0.7824	-0.0522
tnw02.2m	0.3231	0.2827	0.1628	0.061	0.3827	0.2173	0.9326	0.8863
tnw02.10m	0.1193	0.0342	0.3017	-0.2304	0.4412	-0.088	0.6553	0.5225
tnw02.20m	0.1418	-0.0586	0.6461	-0.4664	0.7905	-0.5383	0.4698	0.0984
tnw03.2m	0.5539	0.4374	0.2191	0.0763	0.8029	0.7668	0.6021	0.5639
tnw03.10m	0.158	-0.0442	0.116	-0.0818	0.3156	-0.1354	0.3697	0.1801
tnw04.2m	0.2599	0.1982	0.0639	0.0273	0.3907	0.307	0.4073	0.3753
tnw04.4m	0.2023	0.1518	0.0703	-0.0006	0.2851	0.1615	0.3353	0.2694
tnw04.6m	0.1656	0.116	0.0851	-0.0239	0.2686	0.0577	0.3081	0.1921
tnw04.8m	0.1412	0.092	0.1012	-0.04	0.2806	-0.0454	0.2789	0.1221
tnw04.10m	0.1255	0.0693	0.1193	-0.0483	0.3151	-0.1087	0.275	0.0752
tnw04.12m	0.1109	0.0459	0.13	-0.0594	0.3298	-0.1124	0.2738	0.0544
tnw04.20m	0.1095	0.011	0.1156	-0.0594	0.4297	-0.2079	0.3058	-0.0292
tnw05.2m	0.2687	0.1955	–	–	–	–	0.4181	0.3863
tnw05.4m	0.2786	0.1905	–	–	–	–	0.3956	0.3498
tnw05.6m	0.2891	0.1736	–	–	–	–	0.3702	0.2909
tnw05.8m	0.2689	0.1452	–	–	–	–	0.3172	0.1907
tnw05.10m	0.2511	0.119	–	–	–	–	0.2967	0.118
tnw05.12m	0.244	0.1046	–	–	–	–	0.2891	0.0846
tnw05.20m	0.2176	0.0804	–	–	–	–	0.3238	-0.0382
tnw06.2m	0.2772	0.1943	0.125	0.0598	0.4322	0.2538	0.4025	0.3478
tnw06.4m	0.2659	0.1808	0.1212	0.0148	0.4327	0.0817	0.3626	0.2646
tnw06.6m	0.2649	0.1716	0.1291	-0.0059	0.4722	0.0099	0.3744	0.2163
tnw06.8m	0.2452	0.149	0.1252	-0.0112	0.4887	-0.0317	0.3752	0.1664
tnw06.10m	0.2296	0.1237	0.1221	-0.0183	0.5353	-0.1207	0.3908	0.102
tnw06.12m	0.2221	0.115	0.1224	-0.0269	0.5657	-0.1671	0.4053	0.0725

Continued on next page

Table D.4 – continued from previous page

	0-6h		6-12h		12-18h		18-24h	
Sonic	RMSE	Bias	RMSE	Bias	RMSE	Bias	RMSE	Bias
tnw06.20m	0.2121	0.0997	0.1202	-0.0403	0.5542	-0.1548	0.4187	0.038
tnw07.4m	0.2386	0.1604	0.1951	0.1304	0.7509	0.5544	0.3454	0.2812
tnw07.6m	0.23	0.1481	0.1882	0.1112	0.7859	0.5124	0.3329	0.2316
tnw07.8m	0.2176	0.1336	0.1709	0.0898	0.7385	0.3661	0.3162	0.1602
tnw07.10m	0.2052	0.1144	0.1585	0.0701	0.7096	0.2253	0.3104	0.0902
tnw07.12m	0.2147	0.0698	0.155	0.0673	0.6927	0.0564	0.3213	0.0794
tnw07.20m	0.1952	0.1003	0.1485	0.0718	0.6702	-0.0393	0.3724	0.009
tnw07.30m	0.2181	0.0924	0.1229	0.0581	0.6544	-0.1275	0.4157	0.0012
tnw07.40m	0.2529	0.0851	0.1163	0.0325	0.6711	-0.2068	0.4405	-0.0073
tnw07.60m	0.2864	0.0781	0.1685	-0.0084	0.7002	-0.1975	0.4589	0.017
tnw08.2m	0.2926	0.221	0.0784	0.0324	0.5465	0.277	0.4612	0.3559
tnw08.10m	0.2587	0.1104	0.1065	-0.0228	0.678	0.0799	0.5316	0.1878
tnw08.20m	0.2664	0.0942	0.1379	-0.0676	0.619	0.0329	0.5832	0.0868
tnw09.2m	0.2111	0.1597	0.0837	0.0334	0.5061	0.1881	0.4409	0.3208
tnw09.10m	0.2578	0.1066	0.1274	-0.0863	0.6215	-0.036	0.5765	0.2123
tnw10.10m	0.2109	0.0966	0.1661	-0.1391	0.5014	-0.1646	0.4247	0.15
tnw10.20m	0.1925	0.0286	0.2328	-0.1511	0.7735	-0.6257	0.4442	0.0322
tnw10.30m	0.2052	-0.0088	0.0847	-0.0539	0.6529	-0.4418	0.4317	0.0082
tnw10.40m	0.2122	-0.0237	0.0533	-0.0375	0.5292	-0.3053	0.4388	-0.0307
tnw10.60m	0.2103	-0.0417	0.0465	-0.0333	0.4981	-0.2605	0.4197	-0.0686
tnw11.2m	0.3209	0.2328	0.1717	0.1036	0.6922	0.6777	0.5614	0.5151
tnw11.10m	0.2139	0.099	0.0681	-0.0469	0.2981	-0.1448	0.3341	0.1494
tnw11.20m	0.1733	0.022	0.0537	-0.0381	0.3282	-0.2186	0.3244	0.0316
tnw12.2m	0.2493	0.172	0.0773	0.0262	0.5163	0.4783	0.5214	0.4618
tnw12.4m	0.2489	0.1641	0.0583	-0.0273	0.2797	0.1904	0.4106	0.3126
tnw12.6m	0.2476	0.1564	0.075	-0.0532	0.2366	-0.0236	0.3653	0.2086

Continued on next page

Table D.4 – continued from previous page

	0-6h		6-12h		12-18h		18-24h	
Sonic	RMSE	Bias	RMSE	Bias	RMSE	Bias	RMSE	Bias
tnw12.8m	0.2361	0.137	0.0664	-0.0471	0.2428	-0.0785	0.3492	0.1505
tnw12.10m	0.229	0.1229	0.0591	-0.0418	0.2613	-0.1159	0.3434	0.0992
tnw12.12m	0.2234	0.0982	0.0517	-0.0373	0.2694	-0.1321	0.334	0.0662
tnw12.20m	0.1902	0.0106	0.0485	-0.0359	0.2836	-0.1597	0.3344	-0.0006
tnw12.30m	0.1834	-0.0139	0.0486	-0.0337	0.2858	-0.1823	0.3568	-0.0549
tnw13.2m	0.0447	0.0215	0.0241	0.0061	0.2249	0.2032	0.2707	0.2416
tnw13.4m	0.0892	0.0531	0.0399	0.013	0.4259	0.3803	0.5023	0.4501
tnw13.6m	0.1011	0.0561	0.0308	0.0059	0.402	0.3307	0.5219	0.4577
tnw13.8m	0.1215	0.0666	0.027	0.0042	0.4067	0.3165	0.5358	0.4626
tnw13.10m	0.1302	0.0633	0.0266	-0.0024	0.3356	0.211	0.5021	0.4172
tnw13.12m	0.129	0.0461	0.0315	-0.0132	0.2606	0.0925	0.4312	0.3265
tnw13.20m	0.1791	0.044	0.0485	-0.0361	0.2907	-0.1413	0.3348	0.0671
tnw13.30m	0.23	0.0465	0.0486	-0.0376	0.3251	-0.1871	0.3548	-0.0165
tnw14.2m	0.2167	0.1471	0.0201	0.0033	0.3748	0.3277	0.4645	0.4156
tnw14.4m	0.2021	0.1366	0.0186	0.0048	0.3789	0.3211	0.4741	0.4224
tnw14.6m	0.1906	0.1168	0.0187	0.0039	0.4191	0.3406	0.5105	0.453
tnw14.8m	0.1816	0.1046	0.0176	0.0024	0.4111	0.3246	0.4887	0.4306
tnw14.10m	0.1749	0.0909	0.0185	-0.0028	0.3729	0.276	0.4439	0.3853
tnw14.12m	0.1621	0.073	0.0238	-0.0111	0.2805	0.162	0.3725	0.2991
tnw14.20m	0.1455	-0.0076	0.0419	-0.0291	0.2982	-0.1288	0.297	0.0548
tnw14.30m	0.2024	0.0157	0.0441	-0.0306	0.364	-0.2004	0.2947	-0.0131
tnw15.2m	0.2867	0.2068	0.0161	0.0041	0.3977	0.3488	0.4478	0.4093
tnw15.4m	0.2461	0.1717	0.0164	0.0047	0.3354	0.2737	0.4451	0.3931
tnw15.6m	0.2192	0.1363	0.0178	0.0042	0.2658	0.1689	0.4285	0.357
tnw15.8m	0.2063	0.1178	0.0189	0.0034	0.2281	0.0715	0.3824	0.2925
tnw15.10m	0.1906	0.0902	0.0218	-0.0034	0.2326	0.0204	0.3532	0.2459

Continued on next page

Table D.4 – continued from previous page

	0-6h		6-12h		12-18h		18-24h	
Sonic	RMSE	Bias	RMSE	Bias	RMSE	Bias	RMSE	Bias
tnw15.12m	0.1765	0.0557	0.0284	-0.0143	0.2402	-0.0251	0.3292	0.2008
tnw15.20m	0.176	-0.0165	0.047	-0.0301	0.29	-0.124	0.2914	0.0679
tnw15.30m	0.1964	0.0068	0.0539	-0.0334	0.3433	-0.1862	0.2779	-0.0061
tnw16.2m	0.285	0.2108	0.0135	-0.0007	0.2097	0.1777	0.2936	0.2471
tnw16.4m	0.2971	0.2063	0.0243	-0.012	0.256	0.2034	0.3753	0.3029
tnw16.6m	0.2532	0.1502	0.0314	-0.0223	0.2283	0.1397	0.3813	0.2767
tnw16.8m	0.2289	0.1185	0.0315	-0.0216	0.2133	0.0605	0.3735	0.2256
tnw16.10m	0.2105	0.0942	0.0252	-0.013	0.24	0.0006	0.3657	0.179
tnw16.12m	0.1948	0.0543	0.0301	-0.0178	0.2722	-0.0451	0.352	0.1395
tnw16.20m	0.1778	-0.0185	0.0577	-0.0388	0.302	-0.1277	0.3144	0.0507
tnw16.30m	0.196	-0.0062	0.0491	-0.0295	0.3454	-0.1829	0.2978	0.0026
tse01.10m	0.1616	0.0766	0.145	0.0671	0.7503	0.3407	0.4644	0.2721
tse01.30m	0.1683	0.0752	0.2351	-0.0743	0.7581	-0.1817	0.5551	0.0806
tse02.10m	0.1776	0.1225	0.3192	-0.2148	0.537	0.1091	0.6376	0.5129
tse02.30m	0.1258	0.0355	0.6056	-0.4014	0.9015	-0.6614	0.4138	-0.0047
tse04.10m	0.1707	0.0886	0.1458	-0.1009	0.4381	-0.2253	0.3844	0.1095
tse04.20m	0.1342	0.0131	0.1162	-0.0851	0.4431	-0.2571	0.4217	0.0701
tse04.30m	0.1069	-0.0027	0.1133	-0.0853	0.3945	-0.2387	0.4376	0.0449
tse04.40m	0.0913	-0.0098	0.0979	-0.0703	0.3709	-0.2221	0.4642	0.003
tse04.60m	0.1012	-0.0172	0.0769	-0.0515	0.3778	-0.1899	0.4543	-0.027
tse04.80m	0.1096	-0.0172	0.0799	-0.0496	0.3407	-0.1443	0.4297	-0.059
tse04.100m	0.11	-0.0104	0.0828	-0.0468	0.337	-0.093	0.4186	-0.0721
tse05	—	—	—	—	—	—	—	—
tse06.10m	0.1578	0.0387	0.1229	-0.0083	0.376	-0.0462	0.4379	0.0548
tse06.20m	0.1697	0.0329	0.1097	-0.0248	0.3791	-0.1072	0.4374	0.0046
tse06.30m	0.1724	0.0006	0.1139	-0.0557	0.3827	-0.1284	0.4147	-0.0486

Continued on next page

Table D.4 – continued from previous page

	0-6h		6-12h		12-18h		18-24h	
Sonic	RMSE	Bias	RMSE	Bias	RMSE	Bias	RMSE	Bias
tse06.40m	0.127	-0.0027	0.1212	-0.0666	0.3652	-0.08	0.407	-0.0414
tse06.60m	0.1294	-0.0309	0.144	-0.0847	0.3229	-0.1032	0.3889	-0.0502
tse07.10m	0.2	0.1264	0.1218	0.0435	0.4556	0.249	0.3609	0.2444
tse07.20m	0.2004	0.0894	0.1151	0.0363	0.4098	-0.0003	0.2825	0.0579
tse08.10m	0.1652	0.0834	0.1722	0.0305	0.565	0.2627	0.3572	0.2147
tse08.20m	0.1581	0.0596	0.1778	0.0453	0.4933	-0.0174	0.3369	0.0499
tse09.10m	0.2168	0.1178	0.1437	0.0327	0.6742	0.42	0.4182	0.2724
tse09.20m	0.1894	0.0611	0.1673	0.0103	0.5975	0.0646	0.345	0.0335
tse09.30m	0.185	0.075	0.1479	0.0267	0.6099	0.0209	0.3767	-0.0165
tse09.40m	0.1991	0.0922	0.1456	0.0303	0.6141	-0.0093	0.4015	-0.0203
tse09.60m	0.2121	0.1011	0.1385	0.0212	0.6191	0.0499	0.386	0.0072
tse09.80m	0.1694	0.0756	0.2122	-0.044	0.6904	0.0528	0.3527	0.0278
tse09.100m	0.1375	0.0485	0.226	-0.084	0.754	0.0797	0.3718	0.0284
tse10.10m	0.2328	0.1624	0.1301	0.0652	0.7902	0.5057	0.7413	0.5449
tse10.30m	0.2143	0.1177	0.2014	-0.0694	0.6587	-0.048	0.5233	0.1297
tse11.10m	0.1753	0.1197	0.1128	-0.0537	0.5245	0.1818	0.6248	0.3798
tse11.20m	0.1677	0.091	0.22	-0.1508	0.5132	-0.1301	0.5957	0.217
tse11.30m	0.1776	0.0898	0.2524	-0.1817	0.6585	-0.3382	0.596	0.1407
tse11.40m	0.1941	0.0943	0.2742	-0.1648	0.7004	-0.3912	0.6124	0.1059
tse11.60m	0.1578	0.0576	0.2543	-0.1467	0.675	-0.3062	0.642	0.0725
tse12.10m	0.2162	0.1528	0.0333	0.0007	0.576	0.3654	0.6526	0.5256
tse12.20m	0.1612	0.0351	0.2045	-0.1537	0.5638	-0.3009	0.5753	0.1541
tse13.10m	0.1582	0.0607	0.1569	-0.1255	0.347	-0.2618	0.2774	0.11
tse13.20m	0.1633	0.0061	0.04	-0.0276	0.3978	-0.2766	0.271	-0.0181
tse13.30m	0.1528	-0.0053	0.0329	-0.0209	0.4087	-0.2587	0.3211	-0.0354
tse13.40m	0.1426	-0.013	0.0333	-0.0222	0.3958	-0.2338	0.3449	-0.0427

Continued on next page

Table D.4 – continued from previous page

	0-6h		6-12h		12-18h		18-24h	
Sonic	RMSE	Bias	RMSE	Bias	RMSE	Bias	RMSE	Bias
tse13.60m	0.1454	-0.038	0.0311	-0.0231	0.3865	-0.2135	0.3938	-0.0589
tse13.80m	0.1406	-0.0335	0.0293	-0.0237	0.3681	-0.1932	0.3899	-0.0878
tse13.100m	0.1329	-0.0327	0.0269	-0.0211	0.3626	-0.1793	0.383	-0.0999
rsw01.10m	0.1682	0.0923	0.1886	-0.0508	0.3593	-0.1887	0.3452	0.1812
rsw01.20m	0.1931	0.0356	0.1413	-0.0565	0.3557	-0.2151	0.318	0.078
rsw02.10m	0.2081	0.1015	0.5862	-0.4224	0.5695	-0.3135	0.4265	0.3355
rsw02.20m	0.1715	0.0476	0.0929	-0.0638	0.4208	-0.242	0.2639	0.0713
rsw03.10m	0.1393	0.0533	0.0956	-0.0639	0.3364	-0.1712	0.3106	0.115
rsw03.20m	0.1693	0.0226	0.0943	-0.0679	0.3395	-0.1934	0.3007	0.0368
rsw03.30m	0.1513	0.0215	0.1069	-0.0802	0.3273	-0.187	0.3167	0.0063
rsw03.40m	0.1151	0.0196	0.092	-0.0685	0.3614	-0.2017	0.3368	-0.0339
rsw03.60m	0.1157	0.0209	0.0641	-0.0448	0.3714	-0.1863	0.3504	-0.0697
rsw04.10m	0.1435	0.0582	0.117	-0.079	0.3597	-0.1237	0.3543	0.1297
rsw05.10m	0.1599	0.0575	0.1322	-0.0918	0.4158	-0.1992	0.3269	0.0574
rsw06.10m	0.126	0.0313	0.1166	-0.078	0.4055	-0.1287	0.2943	0.0176
rsw06.20m	0.1143	-0.0176	0.1068	-0.0706	0.4605	-0.2658	0.299	-0.0431
rsw06.30m	0.1291	-0.0168	0.0929	-0.06	0.4627	-0.3018	0.3222	-0.075
rsw06.40m	0.1197	-0.0192	0.0705	-0.0498	0.4565	-0.3099	0.336	-0.1174
rsw06.60m	0.1144	-0.0269	0.0487	-0.0349	0.3877	-0.2517	0.3487	-0.1377
rsw07.10m	0.2087	-0.1146	0.1759	-0.1131	0.3663	-0.1531	0.3485	0.1389
rsw07.20m	0.1071	-0.0552	0.1448	-0.1009	0.3876	-0.1684	0.3286	0.0256
rsw08.10m	0.1583	-0.0868	0.1774	-0.1268	0.3431	-0.1636	0.368	0.058
rsw08.20m	0.095	-0.0441	0.1408	-0.1125	0.3252	-0.1684	0.3459	-0.0334
rne01.10m	0.2072	0.1183	0.0948	-0.0682	0.3457	-0.1988	0.2891	0.1029
rne02.10m	0.185	0.1043	0.1052	-0.0797	0.3464	-0.1871	0.3113	0.2037
rne02.20m	0.1381	0.0513	0.0418	-0.0299	0.4079	-0.2202	0.3036	0.0517

Continued on next page

Table D.4 – continued from previous page

	0-6h		6-12h		12-18h		18-24h	
Sonic	RMSE	Bias	RMSE	Bias	RMSE	Bias	RMSE	Bias
rne03.10m	0.1871	0.0972	0.0448	-0.0244	0.2752	-0.1177	0.251	0.1382
rne04.10m	–	–	–	–	–	–	–	–
rne06.10m	0.1814	0.0461	0.0405	-0.0286	0.2859	-0.1046	0.3531	0.1647
rne06.20m	0.2035	0.0024	0.0443	-0.03	0.3342	-0.1884	0.296	0.0616
rne07.10m	0.2044	0.0803	0.0508	-0.0314	0.4301	-0.2297	0.2874	0.1319
rne07.20m	–	–	–	–	–	–	–	–
v01.2m	0.1185	0.0883	0.1124	0.0521	0.429	0.2575	0.3745	0.2967
v01.10m	0.1962	0.142	0.1412	0.0826	0.6216	0.31	0.4555	0.2987
v03.2m	0.2335	0.1629	0.1437	0.0697	0.5313	0.3713	0.2901	0.2211
v03.10m	0.1874	0.1308	0.1182	0.0548	0.6625	0.2995	0.3737	0.172
v04.2m	0.1959	0.1296	0.1542	0.0858	0.6157	0.4622	0.3835	0.3333
v04.10m	0.2053	0.1276	0.1558	0.0738	0.7895	0.2398	0.3814	0.144
v05.2m	0.1785	0.1153	0.1467	0.0887	0.5881	0.4885	0.4428	0.3749
v05.10m	0.2125	0.1318	0.1349	0.0617	0.7849	0.5141	0.4699	0.2398
v05.20m	0.2402	0.1138	0.1452	0.0626	0.7081	0.2019	0.4665	-0.0027
v06.2m	0.2235	0.1487	0.202	0.1318	0.5385	0.448	0.3947	0.3339
v06.10m	0.2392	0.1178	0.1306	0.0724	0.4927	0.129	0.4255	0.0977
v06.20m	0.2829	0.0993	0.1254	0.0604	0.4918	0.0028	0.4463	0.0142
v07.2m	0.2756	0.1954	0.213	0.1358	0.6584	0.4599	0.3358	0.2475
v07.4m	0.2593	0.1742	0.1994	0.1142	0.7007	0.3999	0.3416	0.188
v07.6m	0.2494	0.1583	0.1963	0.096	0.7702	0.3927	0.3572	0.1654
v07.8m	0.2406	0.147	0.1918	0.0799	0.7395	0.3981	0.3641	0.1757
v07.10m	0.2363	0.141	0.1918	0.0706	0.7288	0.2985	0.3556	0.1088
v07.12m	0.2281	0.1321	0.1907	0.0558	0.711	0.2145	0.3646	0.0599
v07.20m	0.218	0.1224	0.1851	0.0446	0.642	0.0503	0.3909	0.0323
EXTREME_SW	–	–	–	–	–	–	–	–

Continued on next page

Table D.4 – continued from previous page

	0-6h		6-12h		12-18h		18-24h	
Sonic	RMSE	Bias	RMSE	Bias	RMSE	Bias	RMSE	Bias
EXTREME_NE	–	–	–	–	–	–	–	–