INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA MECÂNICA




Análise das condições de vento num parque eólico com terreno florestado recorrendo ao modelo CFD OpenFOAM

JOÃO MIGUEL LEITÃO GOMES PINTO
outubro de 2024

POLITÉCNICO
DO PORTO

ISEP INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO

Analysis of wind conditions over a real wind farm in heavily forested terrain using the OpenFOAM CFD model

João Miguel Leitão Gomes Pinto

Dissertation to fulfil the requirements to obtain the Master degree in Mechanical Engineering, with a specialisation in Energy

Supervisor: Carlos Silva Santos

Jury:
President:
Prof. Dra. Elza Morais Fonseca, Professora Coordenadora, Instituto Superior de Engenharia do Porto
Vowels:
Dr. Carlos Veiga Rodrigues, Vestas
Prof. Dr. Carlos Silva Santos, Professor Adjunto, Instituto Superior de Engenharia do Porto

Porto, September 2024

(White page)

Agradecimentos

Ao Prof. Doutor Carlos Silva Santos, meu orientador, pela confiança depositada em mim para a realização deste trabalho, e pelo apoio, acompanhamento, ajuda, e disponibilidade absolutamente incontestáveis. Foi uma sorte e um gosto pessoal enorme ter tido a oportunidade de podermos trabalharmos juntos, experiência com a qual aprendi imenso.

À Prof. Doutora Marina Duarte, à qual faço questão de dedicar este trabalho e deixar a minha homenagem, por toda a ajuda que sempre me deu durante o decorrer da minha Licenciatura em Engenharia Mecânica, tendo sido a responsável pela minha colaboração com o Prof. Doutor Carlos Silva Santos. Sem a sua iniciativa e vontade em fazer os alunos evoluir, este trabalho provavelmente nunca teria acontecido.

Ao projeto de computação avançada 2023.10427.CPCA.A0 da FCT (Fundação para a Ciência e Tecnologia), pelas horas de recursos computacionais que foram disponibilizados, e também ao Sr. João Pina pelo suporte técnico prestado.

À Daniela, o amor da minha vida, por ser a minha alegria, a minha força, e a minha motivação em tudo aquilo que faço, e por todos os dias estar presente na minha vida.

Aos meus pais e aos meus sogros, pelo suporte diário, e por toda a ajuda que sempre me deram para que eu conseguisse dedicar-me plenamente nos estudos.

À ARCEN Engenharia S.A., nomeadamente aos meus colegas e amigos, e em especial ao Eng.º Francisco Cruz, pela salvaguarda, liberdade, proteção, e apoio que sempre me deu quando precisei.

Ao ISEP, por tudo o que me ensinou e pelos amigos e professores que me deu a conhecer, que muito contribuíram para o meu crescimento.

Resumo

O presente trabalho teve como objetivo efetuar uma análise do recurso eólico e das respetivas condições de vento na área de implantação do parque eólico de Mörknässkogen, localizado na Finlândia Ocidental, que se caracteriza por estar inserido numa região de baixa complexidade topográfica devido ao terreno aproximadamente plano, e com forte presença de floresta sob a forma de pinheiros e abetos de elevada altura de canópia. Essa análise consistiu em simulações de escoamentos atmosféricos realizados com o modelo CFD OpenFOAM, em conjunto com dados de vento obtidos através de uma campanha de medição que foi realizada entre 24/05/2014 e 07/04/2016 com o mastro meteorológico i690. Os resultados provenientes desses dois processos foram processados pelo algoritmo *Synthesis* para avaliar a precisão do modelo matemático, e posteriormente transportar as séries temporais medidas no mastro i690 para as coordenadas das quatro turbinas instaladas, com o objetivo de estimar a produção energética anual do parque e identificar parâmetros prejudiciais ao seu funcionamento. O estudo iniciou-se com a produção de 3 conjuntos de simulações sem modelo de floresta, que revelaram resultados pouco satisfatórios, em que os melhores resultados foram obtidos no SET_{102}, com um erro RMS de 13.06% para a velocidade e 3.21% para a intensidade turbulenta. Numa segunda fase, foram produzidos 30 conjuntos de simulações adicionais, nos quais foi aplicado um modelo de floresta que é introduzido através do modelo de turbulência `kEpsilonLopesdaCosta`. Os primeiros 25 conjuntos de simulações, numerados de A1 a B5, foram configurados para efetuar um processo de calibração dos parâmetros da canópia, enquanto que os últimos 5 conjuntos utilizaram o novo tipo de malha `canopyHD` que possui uma maior resolução da malha vertical na zona por baixo da canópia. Como resultado desse estudo, e comparando com os resultados do SET_{102}, foi selecionado o SET_{214} como sendo o melhor conjunto de simulações, com um erro RMS de 1.57% (-11.49%) para a previsão cruzada da velocidade, e 1.54% (-1.67%) para a intensidade turbulenta. O SET_{214} foi utilizado para estimar a produção anual energética do parque, que foi estimada em 79584.7 MWh/ano contabilizando as quatro turbinas de 4.6 MW com a altura da nacelle de 166 m ANS, o que corresponde a 4325 horas em plena carga, e um fator de capacidade médio teórico de 49.38%. Em termos de avaliação das condições adversas, verificou-se uma velocidade à altura da nacelle ligeiramente superior ao máximo estabelecido pela norma *IEC 61400-1:2019 (4th Edition)*, e um *Shear-factor* muito superior ao limite máximo em todas as direções. Por último, foram desenvolvidas algumas ferramentas para a contabilização dos efeitos térmicos e da força de Coriolis, nomeadamente ao nível da compilação do modelo de turbulência modificado `mykEpsilonLopesdaCosta` para contemplar o campo `GbyNu`, da definição dos termos fonte a serem especificados no ficheiro `fvOptions`, e da análise MERRA2 realizada na região do parque eólico de Mörknässkogen.

PALAVRAS-CHAVE: CFD, OpenFOAM, Escoamentos atmosféricos, Avaliação do recurso eólico, Modelo de floresta

Abstract

The main purpose of this work was to perform an analysis of the wind resource assessment and respective adverse wind conditions in the installation area of Mörknässkogen wind farm, located in Western Finland, which is characterized by being located in a region of low topographical complexity due to its overall flat terrain, with a strong presence of forest in the form of very tall evergreen pine and fir trees. This analysis was performed using simulations of atmospheric flows with the OpenFOAM CFD model, combined with wind data obtained from a measurement campaign that took place between 24/05/2014 and 07/04/2016 using the meteorological mast i690. Results from these two procedures were processed through the *Synthesis* algorithm to evaluate the mathematical model accuracy and to transfer the time series measured at mast i690 to the coordinates of each wind turbine. This allowed for the estimation of the wind farm's annual energy production and to identify harmful parameters that could affect the operation of wind turbines and their performance. The study began with the production of 3 sets of simulations without a forest model, that revealed unsatisfactory results, in which the best results were obtained in SET_{102}, with an RMS error of 13.06% for the velocicty prediction and 3.21% for the turbulent intensity prediction. In a second phase, 30 additional sets of simulations were produced, in which a forest model was introduced through the `kEpsilonLopesdaCosta` turbulence model. The first 25 sets, numbered from A1 to B5, were configured to perform a calibration process of the canopy parameters, while the last 5 sets used the new `canopyHD` mesh type that allows for a significant increase in vertical mesh resolution between the canopy height and the ground surface. As a result of this study, and comparing with the results from SET_{102}, SET_{214} was selected as the overall best set of simulations, with an RMS error of 1.57% (−11.49%) for the velocicty prediction, and 1.54% (−1.67%) for the turbulent intensity prediction. Therefore, SET_{214} was used to calculate the wind farm's annual energy production, which was estimated at 79,584.7 MWh/year for the four 4.6 MW wind turbines with a hub height of 166 m AGL, corresponding to 4325 full load hours and a theoretical average capacity factor of 49.38%. In terms of site verification for adverse wind conditions, it was found that there was a wind speed at hub height slightly higher than the maximum established by the *IEC 61400-1:2019 (4th Edition)* standard, and a overall wind shear for all directions much higher than the maximum limit. Finally, some tools were developed to account for thermal effects and the Coriolis force, namely in terms of compiling the modified turbulence model `mykEpsilonLopesdaCosta` to account for the `GbyNu` field, the definition of the source terms to be specified in the `fvOptions` file, and the MERRA2 analysis performed for the region around Mörknässkogen wind farm.

KEYWORDS: CFD, OpenFOAM, Atmospheric flows, Wind resource assessment, Forest model

Index

List of Figures

List of Tables

Acronyms and Symbols

Acronyms

ABL	Atmospheric Boundary Layer
AEP	Annual Energy Production
AGL	Above Ground Level
ANS	Acima do Nível do Solo
ASL	Above Sea Level
CBL	Convective Boundary Layer
CFD	Computational Fluid Dynamics
CL	Cloud Layer
CM	Control Mass
CSV	Comma Separated Values
CV	Control Volume
DNS	Direct Numerical Simulation
ELR	Environmental Lapse Rate
FA	Free Atmosphere
FCT	Fundação para a Ciência e Tecnologia
FI	Flow Inclination
GAMG	Generalized Geometric-Algebraic Multi-Grid
GEDI	Global Ecosystem Dynamics Investigation
ISEP	Instituto Superior de Engenharia do Porto
LAD	Leaf Area Density
LAI	Leaf Area Index
LES	Large-Eddy Simulation
MERRA2	Modern-Era Retrospective analysis for Research and Applications
ML	Mixed Layer
NASA	National Aeronautics and Space Administration
NBL	Nocturnal Boundary Layer
NEWA	New European Wind Atlas
NWP	Numerical Weather Prediction models
LES	Large-Eddy Simulation
OpenFOAM	Open Field Operation and Manipulation
PBiCG	Preconditioned Bi-Conjugate Gradient
RaNS	Reynolds-averaged Navier-Stokes
RL	Residual Layer
RMS	Root Mean Square
SBL	Stable Boundary Layer
SCL	SubCloud Layer
STL	STereoLithography
TKE	Turbulence Kinetic Energy

TRaNS	Transient Reynolds-averaged Navier-Stokes
URaNS	Unsteady Reynolds-averaged Navier-Stokes
UTM	Universal Transverse Mercator
VLES	Very Large Eddy Simulations
VTK	Visualization ToolKit
WRF-ARW	Weather Research and Forecasting with Advanced Research
WAsP	Wind Atlas Analysis and Application Program

Roman Characters

a	Leaf area density	$m^2\ m^{-3}$
a_g	Acceleration applied due a pressure gradient	$m\ s^{-2}$
A	Speed scale factor for Weibull distribution	$m\ s^{-1}$
A_l	SIMPLE algorithm coefficients	
A_P	SIMPLE algorithm coefficients	
b	*IEC 61400-1:2019 (4th Edition)* constant	
b_{cano}	Canopy width	m
$\mathbf{b}$	Forces acting per unit of mass	$N\ kg^{-1}$
B	Buoyancy production	$W\ kg^{-1}$
c_p	Specific heat at constant pressure	$J\ (kg\ K)^{-1}$
C_0, C_1	Porosity model constants	
C_d	Constant drag coefficient	
C_μ, $C_{\varepsilon 1}$, $C_{\varepsilon 2}$	$k-\varepsilon$ turbulence model constants	
$C_{\varepsilon 4}$	Canopy model constant	
$C_{\varepsilon 5}$	Canopy model constant	
C_z	Canopy model parameter	$m^2\ m^{-3}$
d	Displacement height	m
d_m	Total time of the measurement campaign	days
D	Distance between trees	m
D_{ij}	Strain rate tensor in tensorial notation	s^{-1}
$\mathcal{D}$	Strain rate tensor	s^{-1}
$\mathbf{f}$	Applied forces vector	N
f	Function used in the k-ε-$\overline{v^2}$-f turbulence model	
flh	Full loading hours	h
f_c	Coriolis frequency	Hz
F_i	Drag force	N
g	Magnitude of the gravity acceleration vector	$m\ s^{-2}$
$\mathbf{g}$	Gravity acceleration vector	$m\ s^{-2}$
h_{cano}	Canopy height	m
$\hat{i}, \hat{j}, \hat{k}$	Unit vectors in the x, y and z directions	
I	Turbulent intensity	%
I^*	Non-dimensional turbulent intensity	
It	Average number of iterations of the 12 simulations	%
I_{amb}	Ambient turbulent intensity	%
I_{eff}	Effective turbulent intensity	%
I_{limit}	Maximum turbulent intensity	%
I_{ref}	Reference turbulent intensity at hub-height	%
$\mathcal{I}$	Unit tensor	
L	Monin-Obukhov length	m

$L_{\max}$	Maximum turbulent length scales	m
k	Turbulence kinetic energy per unit mass;	J kg^{-1}
	Shape factor for Weibull distribution	
k^*	Non-dimensional turbulence kinetic energy	
m	Mass	kg
n	Turbulence distribution/decay scaling law exponent	
nz_{cano}	Number of control volume below the canopy	
$\mathbf{n}$	Unit vector normal to the surface	
ni	Number of control volumes in i direction	
N	Number of total control volumes	
N_{bv}	Brunt-Väisälä frequency	Hz
p	Local pressure	Pa
p_s	Pressure at the surface	Pa
p_t	Pressure at the top	Pa
p_0	Reference pressure	Pa
P	Point	
P_k	Mechanical production of turbulence	W kg^{-1}
$\mathcal{P}_w$	Wind flow power density	W m^{-2}
q_t''	Turbulent heat flux	W m^{-2}
Q	SIMPLE algorithm source term	Pa m^{-1}
r	Mixing ratio air-water	
$\mathbf{r}$	Position vector	m
R	Specific gas constant for air	J mol^{-1}
Ri_g	Gradient Richardson number	
Ri_G	Modified gradient Richardson number	
Re	Reynolds number	
S	Surface	m^2
S_i	Source term for porous region	Pa m^{-1}
S_{CV}	Surface surrounding the control volume	m^2
S_k	k equation canopy term	Pa m^{-1}
S_ε	ε equation canopy term	Pa m^{-1}
SF	Wind shear factor	
t	Time	s
T_{ij}	Stress tensor in tensorial notation	Pa
T	Temperature;	℃, K
	Turbulence period;	s
	Rotation period	s
T_0	Cold wall temperature	℃
T_v	Virtual temperature	℃
$\mathcal{T}$	Stress tensor	N
$\mathbf{v}_S$	Surface S_{CV} velocity vector	m s^{-1}
V	Volume	m^3
V_{CM}	Volume occupied by the control mass	m^3
V_{CV}	Volume occupied by the control volume	m^3
$\mathbf{u}$,$\mathbf{v}$	Velocity vector	m s^{-1}
u, v, w	Components x, y, z of the velocity vector	m s^{-1}
u_i	i component of velocity field	m s^{-1}
u_j	j component of velocity field	m s^{-1}
u_*	Friction velocity	m s^{-1}
U	Velocity field	m s^{-1}

U_h	Horizontal velocity	m s^{-1}
U_h^*	Non-dimensional horizontal velocity	
U_{hub}	Wind speed at hub height	m s^{-1}
U_i	i component of the ensemble averaged $\mathbf{u}$	m s^{-1}
$\mathbf{U}$	Ensemble averaged velocity vector	m s^{-1}
$\dot{W}_r$	Wind turbine rated power	W
$\dot{W}_{\text{nom}}$	Wind turbine nominal power	W
x, y, z	Components of an orthogonal coordinate system	m
x_i	x component of coordinate i	m
x_j	x component of coordinate j	m
z_0	Aerodynamic roughness length	m
z_p	Distance between the ground face and the center of the adjacent cell	m
y^+	Wall-normal distance in wall units	

Greek characters

α	Wind shear stress	Pa
α_θ	Equilibrium coefficient	
β	Coefficient of volumetric expansion	K^{-1}
β_d	Canopy model constant	
β_p	Canopy model constant	
δ_{ij}	Kronecker tensor	
η	Constant pressure level	
ϵ	Error	%
ε	Turbulent kinetic energy dissipation rate	W kg^{-1}
ϕ	Conserved intensive property, generic variable	
ϕ^*	Conserved intensive property per unit of mass	
ϕ_m	Dimensionless wind shear	
Φ	Conserved extensive property	
φ	Latitude	°
γ	Flow inclination	°
Γ	Adiabatic lapse rate	℃ m^{-1}
Γ_d	Adiabatic lapse rate for dry air	℃ m^{-1}
κ	Von Kármán constant	
λ	Canopy frontal area index	
μ	Dynamic viscosity	N s m^{-2}
μ_t	Turbulent dynamic viscosity	N s m^{-2}
ν_l	Fluid kinematic viscosity	m^2 s^{-1}
ν_t	Turbulent kinematic viscosity	m^2 s^{-1}
ν_{t_w}	Turbulent kinematic viscosity near wall	m^2 s^{-1}
ν_w	Fluid kinematic viscosity near wall	m^2 s^{-1}
ω	Turbulence frequency	Hz
Ω	Angular velocity	rad/s
ψ_m	Function for momentum and energy	
ρ	Density	kg m^{-3}
ρ_0	Reference density	kg m^{-3}
σ	Standard deviation	
σ_1	Turbulence standard deviation	
σ_{ij}	Reynolds stresses	Pa

σ_k, σ_ε	$k-\varepsilon$ turbulence model constants	
σ_θ	Turbulent Prandtl number	
τ	Shear stress	Pa
τ_{ij}	Viscous component of the stress tensor	Pa
τ_{bv}	Brunt-Väisälä period	s
θ	Potential temperature	℃
θ_v	Virtual potential temperature	℃
ζ	Dimensionless height	

Mathematical operators

$\overline{(\,)}$	Average
$\widetilde{(\,)}$	Calculated quantity
$\widehat{(\,)}$	Transported quantity
$(\,)'$	Perturbation
$\langle\,\rangle$	Filter
$\lvert\ \rvert$	Magnitude/modulus of a vector
d/d	Total derivative
∂/∂	Partial derivative
Δ	Difference
∇	Gradient
$\int$	Definite integral
$\sum$	Summation

Chapter 1

Introduction

1.1 Motivation

The aim of this work was to analyse adverse wind conditions in the installation area of a wind farm, using simulations of atmospheric flows with OpenFOAM [1] CFD (Computational Fluid Dynamics) model, and tools for pre-processing and post-processing developed in Python.

It builds on the work developed in the final project of the Bachelor's Degree in Mechanical Engineering at ISEP (*Instituto Superior de Engenharia do Porto*), also under the guidance of Eng.º Carlos Silva Santos, in conjunction with the need to develop techniques that allow a better characterization and prediction of the wind resource in sites where wind turbines are intended to be installed, focusing on particular cases of greatest interest to the wind energy industry.

These cases that were subject of analysis refer to installation areas with presence of forest, typically with high canopy height, and strong solar radiation during the day, whose temperature gradient near the ground induces thermals that cause changes in the atmospheric stability, which typically varies from an unstable regime to a neutral or stable regime across the diurnal cycle.

Thus, the first goal was to implement a forest model with neutrally stratified simulations in OpenFOAM using the `kEpsilonLopesdaCosta` turbulence model. This is a variant of the $k - \varepsilon$ turbulence model that allows the representation of trees and their behaviour using a porosity model based on `powerLaw` algorithm applied below the canopy, that introduces dissipative terms into the fundamental momentum equations. Using this tool, the aim is to obtain velocity and turbulent kinetic energy profiles that are more representative of the real flow near the ground, and that are closer to the values obtained from local measurements. To the best of our knowledge and based on exhaustive research on the subject, no tutorial or application guidelines of `kEpsilonLopesdaCosta` turbulence model was found, so the entire process consisted of internal investigation of the source code and testing until it was successfully implemented.

The second goal was to add the contribution of thermal effects and Coriolis force in CFD simulations of non-neutrally stratified atmospheric flows using the solver *buoyantBoussinesqSimpleFoam* for steady-state simulations, to reproduce their effect on the atmospheric stability throughout the diurnal cycle. Its variation was

evaluated based on pre-established atmospheric stability classes that depend on the Monin-Obukhov length value. These results were compared with data obtained from NASA's MERRA2 (Modern-Era Retrospective analysis for Research and Applications, Version 2), for the same time interval as the mast measurement campaign, to evaluate the results obtained in the simulations.

In an advanced stage of development of this work, the amount of work required to develop all necessary tools to perform the simulations related to this topic and for the presentation and discussion of the respective results was considered too great for the available timeframe, and therefore the procedures were only partially developed, being a task that is still in progress.

To validate the developed procedures, the final goal is to apply them to the analysis of an existing wind farm, to replicate their respective adverse wind conditions using computational mathematical models. Finally, the accuracy of the wind resource assessment was evaluated by comparison with data collected from the meteorological mast measurement campaign. Furthermore, these data are subsequently translated to the coordinates of the wind turbines through a post-processing algorithm called Synthesis, making it possible to estimate the wind farm's energy production, and to identify harmful parameters for the turbines.

1.2 Thesis content

This thesis is divided into seven Chapters, including the present one, and seven appendixes.

In Chapter 2 a bibliographical review regarding the theoretical scientific background that was followed to develop this thesis is presented, focusing on atmospheric boundary layer physics, as well as the numerical models that are currently used to evaluate the wind resource, followed by an introduction to computational fluid mechanics and the respective tools used in OpenFOAM.

In Chapter 3 the Mörknässkogen wind farm is briefly described, which was the case study used for the implementation and validation of the developed procedures. The site is presented in terms of the installed turbines and the mast, and the local wind regime based on the data obtained during the measurement campaign.

In Chapter 4 the procedures and final results from the initial simulations that were performed considering uniform terrain roughness and variable terrain roughness based on a roughness map, without forest modelling are presented.

In Chapter 5 all tools and developed procedures used to set the forest model based on `kEpsilonLopesdaCosta` turbulence model, and the results of the respective simulations compared to those performed in Chapter 4 are introduced. Finally, the annual energy production of the wind farm was calculated with the best set of simulations.

In Chapter 6 the tools developed for the simulation of non-neutrally stratified ABL flows are presented, as well as an extensive analysis that was performed on the OpenFOAM tutorials `atmFlatTerrain` and `atmForestStability`, and the respective article that was published on these case studies.

Chapter 7 presents the final conclusions and suggestions for future work.

Since this thesis used both existing tools and developed new codes or significantly modified others, and extensive list was added in Appendix A for the benefit of the examiners.

In terms of notation, this thesis follows the style and content presented in [2], considered one of the most recommended works for the study of these topics. It was also decided to use the point as a decimal separator, since the chosen language is English and most available programs, articles and codes follow this numerical representation.

Chapter 2

State of the Art

2.1 Atmospheric Boundary Layer

2.1.1 Structure

The atmospheric boundary layer, identified by most authors with the acronym ABL, represents the region of the troposphere between the Earth's surface and 500-3000 m height, whose depth varies significantly in space and time. Frictional drag, evaporation, heat transfer, pollutant emission, and terrain orography are the main parameters that influence its structure, where, in the absence of which, winds would be purely geostrophic [3]. By definition, all of these phenomena cause the ABL to respond to changes in its characteristics over a period of approximately one hour or less. Above the ABL, the remaining troposphere is called FA (Free Atmosphere), which extends to about 11 km altitude, corresponding to the height of the tropopause at mid-latitudes[1].

In meteorological terms, the wind that exists in the ABL is caused by atmospheric phenomena associated with pressure gradients, which are due to temperature differences caused by the irregular heating of the Earth's surface and also to the Coriolis force [5]. Winds in which these forces are in balance at a given geopotential level are called geostrophic winds [4]. Typically, these winds occur in the free atmosphere, where frictional forces are very low in magnitude. In the ABL, these conditions are not verified, as the wind suffers disturbances caused by the changes in the Earth's surface mentioned above.

Along the Earth's surface, particularly in areas frequently associated with high atmospheric pressure, the ABL has a well-defined structure that evolves with the diurnal cycle [3]. This structure, which is represented in Figure 2.1, is usually classified into three main components:

1) A CBL (Convective Boundary Layer) or ML (Mixed Layer), normally associated with the daytime, where the strong solar incidence increases the thermal gradient close to the ground, causing vertical thermals in the area. This phenomenon promotes a boundary layer of intense mixing, with turbulence of convective origin, which in terms of stability is classified as being of unstable stratification. The wind is subgeostrophic[2], with

[1] Note that, in subtropical latitudes the height of the troposphere can reach ≈17 km, with temperatures around −80°C [4].

[2] Winds whose average speed is less than the average speed of geostrophic winds.

approximately constant speed and direction along the central portion of the CBL. In the presence of clouds, the CBL can be subdivided into a CL (Cloud Layer) and a SCL (SubCloud Layer).

2) An RL (Residual Layer), which forms about 30 minutes before sunset and extends until the formation of the new CBL, which occurs a few hours after sunrise. During this period, thermals stop forming, thus promoting the decay of turbulence in the previous CBL. The result is a neutrally stratified RL in terms of stability, with approximately equal turbulent intensity in all directions.

3) A NBL (Nocturnal Boundary Layer) or SBL (Stable Boundary Layer), normally associated with the night period[3], which represents the lower portion of the RL in contact with the ground. In terms of stability, it is classified as stably stratified, in which the stability of the air tends to dissipate turbulence. Furthermore, it appears that the night wind tends to be weaker close to the ground. However, for typically short periods and at an altitude of approximately 200 m, the wind can become supergeostrophic[4], accelerating to speeds between $10 - 30$ m/s, in a phenomenon called "night jets". This type of local event tends to increase the shear stress of the flow, generating turbulence as a result. On the other hand, at an altitude of about 400 m, the velocity decreases and approaches the geostrophic value.

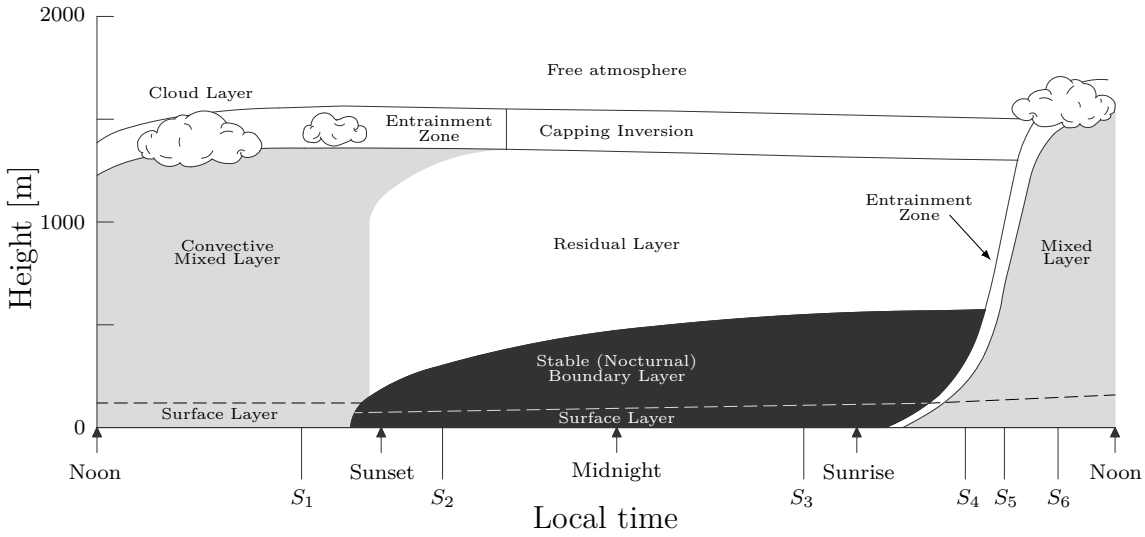


Figure 2.1: Structure of the ABL throughout the diurnal cycle: A turbulent mixed layer (CBL); a less turbulent residual layer (RL) that contains the previous CBL air; a stable nocturnal layer of sporadic turbulence (NBL). Adapted from [3].

[3]Under certain special conditions, the NBL can also form during the day, as long as the underlying surface is colder than the surrounding air. These situations can occur during the advection of hot air over a colder surface, such as in heat waves or in areas close to the coast

[4]Winds whose average speed is greater than the average speed of geostrophic winds.

2.1.2 Turbulence in the ABL

Turbulence is a highly complex physical phenomenon, consisting of the superposition of countless swirls of different scales called eddies, that interact with each other in a nonlinear and almost random way, creating chaotic motions [4].

In the ABL, turbulence refers to fluctuations in wind speed on a relatively fast time scale, typically less than 10 minutes, and can be generated mechanically, thermally, and inertially [3]. In this context, most authors indicate that these main sources of turbulence are caused by the following:

1) Mechanical turbulence, also known as forced convection, can be generated if there is shear stress in the mean wind caused by frictional drag, which typically occurs on the ground surface. Additionally, it can also be generated as the wind swirls behind obstacles such as trees, buildings, hills, and even wind turbines, which cause a wake effect downstream.

2) Thermal or convective turbulence, also known as free convection, consists of plumes or thermals of warm air that rise and cold air that sinks due to buoyancy forces, with the main origin coming from solar radiation. Another possibility is radiative cooling of the top of the cloud layer, although this is a less frequent phenomenon. Furthermore, small eddies can also be generated along the edges of larger eddies, a process called the turbulent cascade, where some of the inertial energy of the larger eddies is lost to the smaller eddies. This phenomenon is a special form of shear turbulence called inertial turbulence, where the shear is generated by larger eddies. The superposition of all scales of eddy motion can be quantified via an energy spectrum, which indicates how much of the total turbulence kinetic energy k is associated with each eddy scale.

Since most of these phenomena occur very close to the ground, it is at the surface that the turbulent intensity reaches its maximum value, decreasing with height until a zero value in the free atmosphere. To evaluate the characteristics of turbulent intensity, the concept of turbulent kinetic energy k (or TKE) is used, and it is directly related to the momentum, heat, and moisture transport through the boundary layer. Its value is given by,

$$\frac{\mathrm{TKE}}{m} = k = \frac{1}{2}\left(\overline{u'^2} + \overline{v'^2} + \overline{w'^2}\right), \tag{2.1}$$

where m is the mass, k represents the instantaneous turbulent kinetic energy per unit mass, and $u^{'}$, $v^{'}$, and $w^{'}$, are, respectively, the fluctuations of the orthogonal components of the velocity vector in the directions x, y, and z around their mean value. In other words, k is the sum of the variances of each component of the velocity vector divided by two.

Near the surface, when k reaches its maximum value, its dissipation rate ε also reaches its maximum value under the same conditions, producing a close balance between viscous dissipation and shear stress [3]. Based on this balance, it is possible to write the k balance equation, in which the production terms are balanced with

the destruction terms, in order to estimate whether the boundary layer will become more turbulent, or the turbulence will decrease. This balance equation is given by:

$$\underbrace{\frac{\partial \overline{k}}{\partial t}}_{\text{I}} + \underbrace{\overline{U}_j \frac{\partial \overline{k}}{\partial x_j}}_{\text{II}} = \underbrace{\delta_{i3} \frac{g}{\overline{\theta}_v} \left(\overline{u_i{}' \theta_v{}'}\right)}_{\text{III}} - \underbrace{\overline{u_i' u_j'} \frac{\partial \overline{U}_i}{\partial x_j}}_{\text{IV}} - \underbrace{\frac{\partial \left(\overline{u_j' k}\right)}{\partial x_j}}_{\text{V}} - \underbrace{\frac{1}{\overline{\rho}} \frac{\partial \left(\overline{u_i' p'}\right)}{\partial x_i}}_{\text{VI}} - \underbrace{\varepsilon}_{\text{VII}}, \quad (2.2)$$

where U is the velocity field, δ_{ij} the Kronecker tensor (which takes the value of $\delta_{ij} = 1$ when $i = j$, or $\delta_{ij} = 0$ otherwise), g is the acceleration of gravity, θ_v the virtual potential temperature, ρ the density, and p is the local pressure. The terms indicated in the equation 2.2 represent:

I. Term that represents local storage or tendency of k.

II. Term that describes the advection of k by the mean wind speed field.

III. Dynamic term for buoyant production or consumption. It is a production or loss term depending on whether the heat flux $\overline{u_i' \theta_v'}$ is positive (typically during daytime over land) or negative (at night over land).

IV. Term for mechanical or shear stress production/loss. The momentum flux $\overline{u_i' u_j'}$ generally has the opposite sign from the mean wind shear stress, because the wind momentum is usually lost downward to the ground. Thus, term IV results in a positive contribution to the turbulent kinetic energy k when multiplied by a negative sign.

V. Term that represents the turbulent transport of k, which describes how k is moved around by the turbulent eddies u_j'.

VI. Pressure correlation term, which describes the redistribution of k in response to pressure perturbations. It is often associated with air oscillation through buoyancy or gravity waves.

VII. Term that represents the viscous dissipation of k, which is reflected in its conversion into heat.

However, equation 2.2 can be simplified if a coordinate system aligned with the mean wind is chosen, and assuming horizontal homogeneity in the flow, resulting in the special form:

$$\underbrace{\frac{\partial \overline{k}}{\partial t}}_{\text{I}} = \underbrace{\frac{g}{\overline{\theta}_v} \left(\overline{w' \theta_v{}'}\right)}_{\text{III}} - \underbrace{\overline{u' w'} \frac{\partial \overline{U}}{\partial z}}_{\text{IV}} - \underbrace{\frac{\partial \left(\overline{w' k}\right)}{\partial z}}_{\text{V}} - \underbrace{\frac{1}{\overline{\rho}} \frac{\partial \left(\overline{w' p'}\right)}{\partial z}}_{\text{VI}} - \underbrace{\varepsilon}_{\text{VII}}. \quad (2.3)$$

Thus, turbulence in the ABL is a dissipative phenomenon in nature, and term VII (energy dissipation) is present whenever $k \neq 0$. Physically, this means that turbulence will tend to decrease and disappear over time, unless it can be generated locally or transported by turbulent or pressure processes.

Therefore, the turbulent kinetic energy k is not a constant or conservative quantity, and the ABL can be turbulent only if there are specific physical processes generating the turbulence.

2.1.3 Stratification and stability

Atmospheric stability can be viewed as the relative tendency for an air parcel to move vertically. According to the traditional definition, it can be evaluated based on the rate of temperature change with height z following a parcel of air that is being raised or lowered adiabatically in the atmosphere, considering that an air parcel that undergoes only adiabatic transformations and that the atmosphere is in hydrostatic equilibrium. This physical quantity is called adiabatic lapse rate Γ which, in the case of dry air, is given by:

$$\Gamma_d = \frac{g}{c_p} = -\left.\frac{\partial T}{\partial z}\right|_d , \tag{2.4}$$

where c_p is the specific heat at constant pressure, and d denotes dry air. In the atmosphere, Γ_d has an approximate value of 9.8℃ km^{-1}, although the actual lapse rate Γ measured by radiosondes averages values around 6 to 7℃ km^{-1} in the troposphere. In nearly all cases, the value of Γ may be different from Γ_d due to local variations in the temperature profile between different atmospheric layers. Some authors also make reference to the ELR[5] (Environmental Lapse Rate) [6], although it is not used to evaluate atmospheric stability.

In order to compare parcels of air existing at different altitudes, i.e. at different levels of atmospheric pressure, the potential temperature θ is frequently used because it is a conserved quantity for atmospheric processes in adiabatic conditions, that is defined by the Poisson equation [4]:

$$\theta = T\left(\frac{p_0}{p}\right)^{R/c_p} , \tag{2.5}$$

where p_0 is a reference pressure, and R the specific gas constant for air [7]. Alternatively, the concept of virtual potential temperature θ_v can also be used, which accounts for the water present in the air based on the mixing ratio r, and which for dry air can be defined by [3]:

$$\theta_v = \theta\,(1 + 0.61r)\,. \tag{2.6}$$

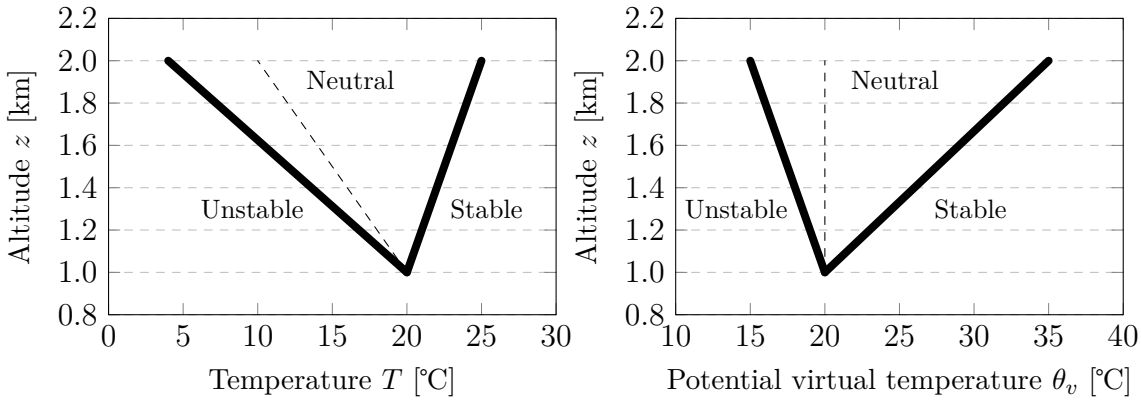
Therefore, and as represented in Figure 2.2, atmospheric stability can be classified either in terms of the variation of the virtual potential temperature θ_v, or in terms of the dry adiabatic lapse rate Γ_d, as follows:

- If $\Gamma_d > \Gamma$ or $\partial\theta_v/\partial z > 0$, the atmosphere is stably stratified. In this case, if an air parcel rises adiabatically to a higher atmospheric layer, its temperature will be lower than the surrounding air and less buoyant, which will lead to compression of the air parcel. As a consequence, there will be an increase in the density ρ, which will cause it to fall back to its original position. On the other hand, for a parcel of air descending adiabatically to a lower atmospheric

[5]The dry adiabatic lapse rate Γ_d and the ELR represent different quantities. The first one is the rate at which a dry parcel will cool if it is moved upward through the atmosphere, and also the rate at which it will warm if it moves down towards the ground (when it encounters increased atmospheric pressure and becomes compressed). The ELR, on the other hand, is a measure of the actual temperature structure existing above a given location as sensed by thermometers at fixed heights on a mast, or attached to a balloon or an aircraft.

layer, the final result is the same, but following the reverse thermodynamic process. In both situations, a positive difference between the values of Γ_d and Γ implies the appearance of restoring forces that force the air parcel to return to its original position of hydrostatic equilibrium [4].

- If $\Gamma_d = \Gamma$ or $\partial\theta_v/\partial z = 0$, the atmosphere is neutrally stratified. In this case, if a parcel of air rises adiabatically to a higher atmospheric layer, its temperature will be equal to the surrounding air, causing the parcel of air to remain stationary at that level. Neutral stratification conditions in the ABL occur in situations of strong wind and/or fog, where these weather phenomena restrict heat transfer near the ground, thus minimizing the development of any temperature stratification [6].
- If $\Gamma_d < \Gamma$ or $\partial\theta_v/\partial z < 0$, the atmosphere is unstably stratified. In this case, if a parcel of air rises adiabatically to a higher atmospheric layer, its temperature will be higher than the surrounding air, causing the parcel of air to continue rising until it reaches an area with the same temperature [8]. Unstable conditions in the ABL typically occur on days of strong sun exposure, as the high thermal gradient near the ground causes thermals of warm air that rise into the atmosphere because of buoyancy forces [6].



(a) (b)

Figure 2.2: Evaluation of atmospheric stability regimes in terms of temperature T or virtual potential temperature θ_v. (a) Stability from temperature T. (b) Stability from virtual potential temperature θ_v. Adapted from [8]

There are also other approaches to evaluate atmospheric stability, namely through the Brunt-Väisälä frequency N_{bv}, or buoyancy frequency, that measures the static stability of the atmosphere by analysing the oscillation of a disturbed air parcel around its initial altitude, whose period is $\tau_{\text{bv}} = 2\pi/N_{\text{bv}}$. In the course of the oscillation, kinetic energy is transformed into potential energy, giving rise to gravitational waves resulting from buoyancy restoration. Mathematically, the Brunt-Väisälä frequency N_{bv} is obtained from the expression [8]:

$$N_{\text{bv}}^2 = g\frac{\partial(\ln\theta_v)}{\partial z} = \frac{g}{T_v}(\Gamma - \Gamma_v), \tag{2.7}$$

where T_v is the virtual temperature. According to this criterion, the atmosphere is stably stratified if $N_{\text{bv}}^2 > 0$, unstably stratified if $N_{\text{bv}}^2 < 0$, and neutrally stratified if $N_{\text{bv}}^2 = 0$.

However, certain authors suggest that only knowledge of the adiabatic lapse rate Γ is insufficient to determine the atmospheric stability, requiring knowledge of the entire virtual potential temperature profile θ_v, or measurement of buoyancy flux [3].

In planetary terms, and in good weather conditions, the ABL tends to be unstable during the day and stable at night. Exceptions occur on high-altitude snow surfaces during the winter, where the boundary layer is stable for long periods of time, and on tropical ocean surfaces, where it can be unstable for equally long periods [6].

2.1.4 Monin-Obukhov similarity theory

In the presence of strong wind shear above the surface, the traditional bulk aerodynamic formulae are not useful to parameterise kinematic fluxes. A better method is to use the Monin-Obukhov similarity theory [9], which uses an approach of combining variables into dimensionless groups, as a whole, and then fit them as a function of some parameter, with an empirical equation. Hence, this method of obtaining an empirical equation for the dimensionless group is called similarity theory, and the relationship between the empirical equation and the dimensionless group is a similarity relationship [8]. One similarity relationship is that for the dimensionless wind shear ϕ_m, given by:

$$\frac{\phi_m}{\kappa} = \frac{z}{u_*}\frac{\partial|\overline{U_h}|}{\partial z}, \tag{2.8}$$

where $\partial|\overline{U_h}|/\partial z$ is the wind shear that is measured directly and u_* the friction wind speed who relates air density with shear stress caused by surface drag by $\tau = \rho u_*^2$. The right side of this equation, as a whole, is dimensionless. The parameter ϕ_m/κ is determined as a function of z/L by substituting measurements of $\partial|\overline{U_h}|/\partial z$ and u_* into equation 2.8 for different values of z/L and fitting curves to the resulting data. The parameter L is the Monin-Obukhov length, z/L is a dimensionless height, and κ is the von Kármán constant that is found by substituting measurements of $\partial|\overline{U_h}|/\partial z$ under neutral conditions, when $\phi_m = 1$.

By making the constant flux (with height) approximation, one can use surface values of heat and momentum flux to define turbulence scales and non-dimensionalise the turbulent kinetic energy k equation 2.2, by multiplying the whole equation by $(-\kappa z/u_*^3)$, assuming that all turbulent fluxes are equal to the values on the respective surfaces, and focusing on terms III, IV, and VII:

$$\cdots = \underbrace{-\frac{\kappa z g\left(\overline{w'\theta_v'}\right)_s}{\overline{\theta_v} u_*^3}}_{\text{III}} + \underbrace{\frac{\kappa z\left(\overline{u_i'u_j'}\right)_s}{u_*^3}\frac{\partial \overline{U_i}}{\partial x_j}}_{\text{IV}} + \cdots - \underbrace{\frac{\kappa z \varepsilon|_s}{u_*^3}}_{\text{VII}}. \tag{2.9}$$

Term III is usually assigned the symbol ζ and is further defined by:

$$\zeta = \frac{z}{L} = -\frac{\kappa z g\left(\overline{w'\theta_v'}\right)_s}{\overline{\theta_v} u_*^3}. \tag{2.10}$$

Thus, the Monin-Obukhov length L is given by:

$$L = -\frac{\overline{\theta_v} u_*^3}{\kappa g \left(\overline{w'\theta_v'}\right)_s}. \tag{2.11}$$

One physical interpretation of L is that it is proportional to the height above the surface at which buoyant factors first dominate over mechanical (shear) production of turbulence. For convective situations, buoyant and shear production terms are approximately equal at $z = -L/2$. Figure 2.3 shows the typical range of variations of the Monin-Obukhov length in fair weather conditions over land.

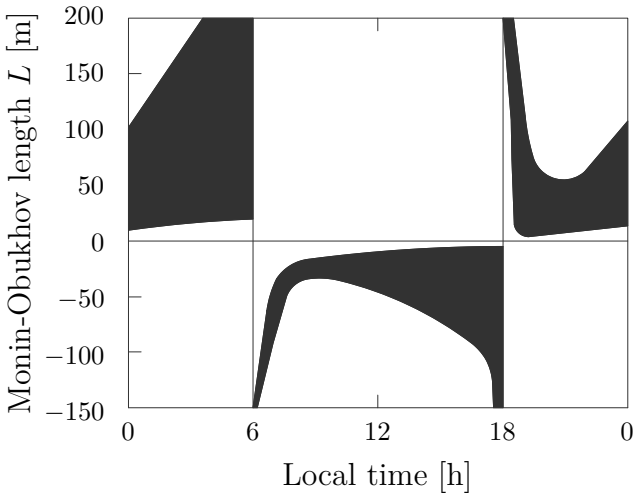


Figure 2.3: Typical ranges of Monin-Obukhov length L evolution over a diurnal cycle. Adapted from [3]

In order to evaluate atmospheric stability, Monin-Obukhov length L can be useful in the surface layer to establish stability classes. In terms of the dimensionless height ζ, atmospheric stability can be classified as stably stratified if $\zeta > 0$, unstably stratified if $\zeta < 0$, and neutrally stratified if $\zeta = 0$.

For a more detailed classification, Table 2.1 shows the stability classes proposed by different authors, with formulations both for the Monin-Obukhov length L [10] and dimensionless height ζ [11].

Table 2.1: Formulations used to classify atmospheric stability

Stability class	L	$\zeta = z/L$
Very stable (VS)	$10 \leq L \leq 50$	$0.6 \leq \zeta \leq 1$
Stable (S)	$50 \leq L \leq 200$	$0.2 \leq \zeta \leq 0.6$
Near neutral/stable (NS)	$200 \leq L \leq 500$	$0.02 \leq \zeta \leq 0.2$
Neutral (N)	$\lvert L\rvert \geq 500$	$-0.02 \leq \zeta \leq 0.02$
Near neutral/unstable (NU)	$-500 \leq L \leq -200$	$-0.2 \leq \zeta \leq -0.02$
Unstable (U)	$-200 \leq L \leq -100$	$-0.6 \leq \zeta \leq -0.2$
Very unstable (VU)	$-100 \leq L \leq -50$	$-1 \leq \zeta \leq -0.6$

2.1.5 Roughness length and displacement height

Similarity theory can also be used to derive profiles of vertical wind speed, potential virtual temperature, and specific humidity in the surface layer. Integrating equation 2.8 between z_0 and z, the wind speed profile as a function of height is obtained:

$$u(z) = \frac{u_*}{\kappa}\left[\ln\left(\frac{z}{z_0}\right) - \psi_m\right], \tag{2.12}$$

where ψ_m is the influence function for momentum and energy, given by:

$$\psi_m = \int_{z_0}^{z} (1 - \phi_m)\,\frac{dz}{z}. \tag{2.13}$$

Integrating equation 2.13 with values of ϕ_m, the influence function equation ψ_m is subdivided into other functions for different atmospheric stability regimes [8]. The influence function for momentum accounts for the difference between a logarithmic wind speed profile and an actual profile under stable and unstable conditions. The influence function for energy is analogous to that for momentum. Under neutral conditions ($\phi_m = 1$), equation 2.12 reduces to a standard logarithmic wind profile for a neutrally stratified surface layer,

$$u(z) = \frac{u_*}{\kappa}\ln\left(\frac{z}{z_0}\right). \tag{2.14}$$

The parameter z_0 presented in equation 2.14 is the aerodynamic roughness length, that is defined as the height where the wind speed becomes zero. Although this roughness length is not equal to the height of the individual roughness elements on the ground, there is a one-to-one correspondence between those roughness elements and the aerodynamic roughness length z_0[6].

For surfaces with high values of z_0, which typically occurs in areas of forest canopies or high vegetation, the individual roughness elements can be packed very closely together, making the top of those elements act like a displaced surface. In such scenarios, some authors suggest the introduction of another parameter called the displacement height d, which can be visualised as the vertical distance that z_0 is displaced. For a wide range of crops and trees, the displacement height value usually lies within 70 to 80% of the canopy height h_{cano} [8], while some authors suggest the approximation:

$$d \approx \frac{2}{3}h_{\text{cano}}. \tag{2.15}$$

When a displacement height exists, z_0 is defined as the height above the displacement height, and the mean wind speed extrapolates to zero at the height $(d + z_0)$, modifying equation 2.14 into:

$$u(z) = \frac{u_*}{\kappa}\ln\left(\frac{z-d}{z_0}\right). \tag{2.16}$$

[6] In other words, once the aerodynamic roughness length z_0 is determined for a particular surface, it does not change with wind speed, stability, or stress. It can change if the roughness elements on the surface change, such as caused by changes in the height and coverage of vegetation, trees, erection of fences, construction of houses, deforestation or lumbering [3]

Thus, the logarithmic wind speed profile is redefined according to this new formulation, as shown is Figure 2.4. However, d and z_0 also depend on the density of the drag elements, canopy height, vertical variation of foliage density and wind speed, because most vegetation is flexible and assumes a more streamlined profile for high wind speeds. Typical values of z_0 and d are listed in Table 2.2.

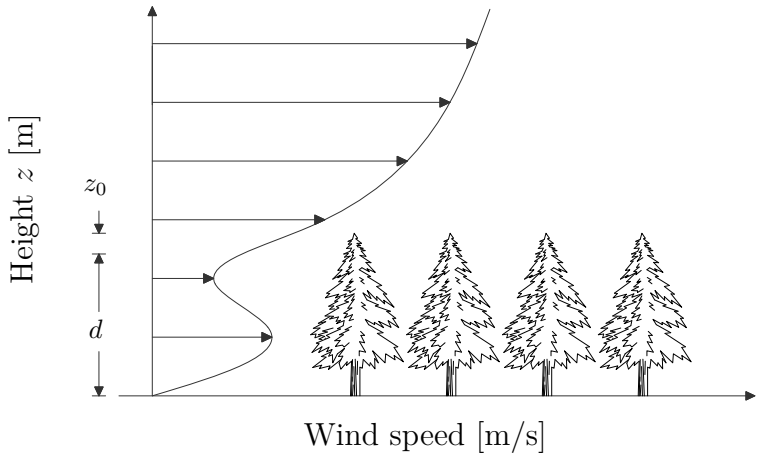


Figure 2.4: Flow over forest canopy showing wind speed as a function of height z. The thick canopy layer acts like a surface vertically displaced d, above the true surface. Adapted from [3]

Table 2.2: Typical aerodynamic roughness lengths z_0 and displacement heights d for several surfaces types

Surface type	z_0 [m]	d [m]
Rough sea	0.000015-0.0015	-
Smooth sea and ice	0.00001	-
Snow	0.00005-0.0001	-
Desert and sand	0.0003	1.00
Short grass	0.003-0.01	< 0.075
Long grass	0.04-0.1	0.19-0.75
Agricultural crops	0.04-0.2	0.27-1.3
Coniferous forest	0.28-3.9	6.3-25.3
Evergreen forest	4.8	26.3
Evergreen trees	2.4	12.8
Short vegetation	0.12	0.75

2.1.6 Canopy models

As briefly mentioned in the previous Section, the presence of forested areas significantly interferes with the wind speed profile, making it essential to model it computationally to obtain better results in the cross-prediction of wind resource. In RaNS momentum equations (discussed further in Section 2.3.3.2), the canopy

presence is introduced as a drag force, that reproduces the momentum dissipation caused by the trees and its foliage, given by:

$$F_i = -C_d a(z) \rho |\mathbf{U}| U_i, \tag{2.17}$$

where C_d is a constant drag coefficient, $a(z)$ is the local foliage density which can be set to vary with z, $|\mathbf{U}|$ is the magnitude of the velocity vector and U_i the component in x direction.

Every forest involved in computer simulations needs to be characterised by quantitative parameters. One of the most simplistic ways is to use the average height of the trees, but this is not very accurate as it does not describe the foliage density. One way to represent this foliage density is by the frontal area index:

$$\lambda = \frac{b_{\text{cano}} h_{\text{cano}}}{D^2}, \tag{2.18}$$

where b_c is the tree width, and D represents the distance between trees. A more advanced model that is used to characterise the canopy is the LAD (Leaf Area Density) which defines the foliage density as a function of height z, as shown in Figure 2.5. LAD is a convenient function to describe the density along the height of a tree and can be calculated through the total area of one side of the foliage divided by its total volume.

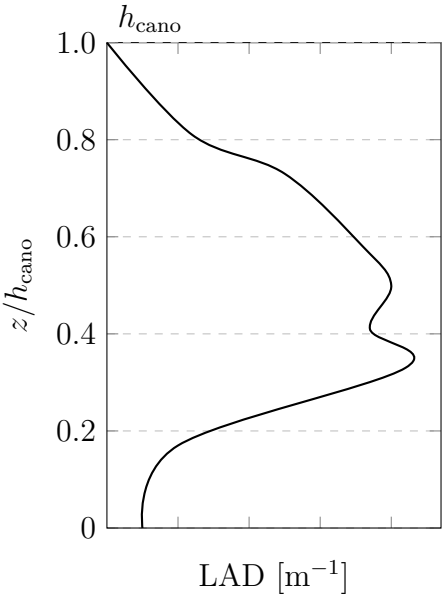


Figure 2.5: Example of the LAD profile of a forest canopy. Adapted from [12]

However, a more useful variable is the integral parameter LAI (Leaf Area Index) which is defined by the integral of $a(z)$ over the tree height h_{cano}, given by:

$$\text{LAI} = \int_0^{h_{\text{cano}}} a(z)\, dz \approx \overline{a(z)} h_{\text{cano}}. \tag{2.19}$$

In terms of turbulence generation, the forest canopy also has its own mechanisms. The forest acts on the flow by creating a plane mixing layer near the top of the

canopy, which slows down the air moving through the vegetation. This phenomenon, along with the complex behaviour of air around the trees, is expected to generate high levels of turbulence. However, the canopy also acts as a turbulence dissipator, particularly for larger length scales. In this sense, the canopy can be considered a strong dissipator for larger turbulent scales, as the larger turbulent eddies are broken down into smaller eddies within the foliage, that have a faster dissipation rate [13].

This mechanism of production and dissipation of turbulence created by canopy foliage is modelled by source/sink terms S_k and S_ε that are added to k and ε equations. Several authors propose different source/sink terms, although the most commonly used are:

$$S_k = \rho C_z \left(\beta_p |\mathbf{u}|^3 - \beta_d |\mathbf{u}| k\right), \tag{2.20}$$

$$S_\varepsilon = \rho C_z \left(C_{\varepsilon 4} \beta_p \frac{\varepsilon}{k} |\mathbf{u}|^3 - C_{\varepsilon 5} \beta_d |\mathbf{u}| \varepsilon\right), \tag{2.21}$$

where $C_z = C_d a(z)$ and $|\mathbf{u}|$ is the magnitude of the velocity vector. The other parameters $C_{\varepsilon 4}$, $C_{\varepsilon 5}$, β_d and β_p are canopy model constants. A range of values have been suggested by various authors in published literature for the required modelling constants, and those that produced the best results in canopy model tests are presented in Table 2.3. However, in this work the constants recommended in Lopes da Costa (2007) [13] were chosen, since they are those used by the `kEpsilonLopesdaCosta` turbulence model in OpenFOAM.

Table 2.3: Sets of constants used in different canopy models

Author	β_p	β_d	$C_{\varepsilon 4}$	$C_{\varepsilon 5}$
Svensson and Häggkvist (1990) [14]	1.0	0.0	1.95	0.0
Green (1992) [15]	1.0	4.0	1.5	1.5
Liu et al. (1996) [16]	1.0	4.0	1.5	0.6
Sanz (2003) [17] / Katul et al. (2004) [18]	1.0	5.1	0.9	0.9
Lopes da Costa (2007) [13]	0.17	3.37	0.9	0.9

According to the work developed in Lopes da Costa (2007) [13], the constants obtained in the study (see Table 2.3), used under the $k-\varepsilon$ turbulence model, yielded the best agreement according to the LES (Large-Eddy Simulation) benchmark of the same flow when compared to the other sets. It has also established a new procedure for obtaining relationships between the canopy model constants β_p and β_d, using the k-ε-$\overline{v^2}$-f turbulence model equations.

2.1.7 Adverse wind conditions

The atmospheric flows in the ABL are constantly subjected to adverse wind conditions that occur due to special properties of the underlying surface, or unfavourable atmospheric conditions. These phenomena are usually related to the presence of complex topography, forest [19], cities, or atmospheric instability.

For this reason, the correct prediction and characterisation of the local wind regime is extremely important when installing wind turbines, both in terms of their estimated AEP (Annual Energy Production), and in terms of identifying potential

harmful quantities to the turbines, which may eventually compromise the integrity of the equipment.

2.1.7.1 Wind shear factor

One of the most used indicators in this context is the wind shear factor, identified by α or SF (Shear Factor), which represents a measure of the wind shear stress, typically associated with a rapid variation of wind speed with height, that is calculated with the expression,

$$\alpha_{1-2} = \frac{\ln\left(U_{z_2}/U_{z_1}\right)}{\ln\left(z_2/z_1\right)}, \tag{2.22}$$

where α_{1-2} represents the wind shear stress between heights z_1 and z_2, and U_{z_1} and U_{z_2} the respective velocity at those locations. This quantity is particularly important at hub height and is calculated using the computed wind velocities at the top and bottom of the rotor swept area.

In qualitative terms, a high shear stress value can be considerably harmful for wind turbines, as it causes excessive stress on the structure. Similarly, negative wind shear is also dangerous as it can push the blades against the tower in their downward motion. For this reason, values of $0 < \alpha < 0.2$ are recommended for this type of application according to *IEC 61400-1:2019 (4th Edition)* [20].

2.1.7.2 Flow inclination

Another important quantity is the flow inclination γ, obtained directly from the CFD simulations, without any averaging procedure to avoid negative and positive values cancelling each other out. Typically its value is calculated at hub height, and this point is used as a representative inclination for the whole rotor, with the following expression,

$$\gamma = \tan^{-1}\left(\frac{w}{U_h}\right), \tag{2.23}$$

where $U_h = \sqrt{u^2 + v^2}$ is the horizontal velocity of the flow. Ideally, this quantity is intended to have a value very close to zero, a condition that occurs in horizontal flows. If this condition is not met, that is, if the wind has a relatively steep flow slope, it can cause unwanted bending moments in the turbine blades and even in the tower structure. According to *IEC 61400-1:2019 (4th Edition)* [20], values of $|\gamma| < 8°$ are desired although some turbine manufactures tolerate values up to $|\gamma| < 12°$.

2.1.7.3 Turbulent intensity

Finally, another indicator that is also universally used is the turbulent intensity I, which describes the fluctuation of velocity around an average value. This property is intended to have its value as low as possible, in order to ensure a more stable and less disturbed flow. Its value can be determined with the equation,

$$I = \frac{\sigma}{\overline{U}} \approx \frac{\sqrt{\frac{2k}{3}}}{\overline{U}_h}. \tag{2.24}$$

The second equation $\sqrt{2k/3}/U_h$ is only valid if isotropic conditions are assumed. According to *IEC 61400-1:2019 (4th Edition)* [20], the maximum values of turbulence intensity I_{limit} can be calculated using the turbulence standard deviation σ_1, with the expression:

$$I_{\text{limit}} = \frac{\sigma_1}{U_{\text{hub}}} = \frac{I_{\text{ref}}\,(0.75U_{\text{hub}} + b)}{U_{\text{hub}}}, \tag{2.25}$$

where $b = 5.6$ m/s, I_{ref} is the expected value of hub-height turbulence intensity at a 10 minutes average wind speed of 15 m/s, and U_{hub} is the wind speed at hub height.

Table 2.4 presents the maximum turbulent intensity values established for different categories of wind turbines. Its distinction is related to its application depending on different turbulence levels, from the most resilient subclass A+ which can endure $I_{\text{ref}} = 0.18$, to subclass C with $I_{\text{ref}} = 0.12$. Other subclasses include subclass A ($I_{\text{ref}} = 0.16$) and subclass B ($I_{\text{ref}} = 0.14$).

Table 2.4: Maximum turbulent intensity values for for different categories of wind turbines

	Category			
	A+	*A*	*B*	*C*
U_{hub} [m/s]	I_{limit} [%]	I_{limit} [%]	I_{limit} [%]	I_{limit} [%]
3	47.10	41.87	36.63	31.40
5	33.66	29.92	26.18	22.44
7	27.90	24.80	21.70	18.60
9	24.70	21.96	19.21	16.47
11	22.66	20.15	17.63	15.11
13	21.25	18.89	16.53	14.17
15	**20.22**	**17.97**	**15.73**	**13.48**
17	19.43	17.27	15.11	12.95
19	18.81	16.72	14.63	12.54
21	18.30	16.27	14.23	12.20
23	17.88	15.90	13.91	11.92
25	17.53	15.58	13.64	11.69
27	17.23	15.32	13.40	11.49
29	16.98	15.09	13.20	11.32
31	16.75	14.89	13.03	11.17
33	16.55	14.72	12.88	11.04

Later in this work, particularly in Chapter 3, it is mentioned that the wind turbines installed at the Mörknässkogen wind farm, which was the case study in this thesis, refer to a class III A wind turbine model, therefore the maximum turbulent intensity considered was $I_{\text{limit}} = 17.97\%$ for a wind speed of 15 m/s at hub height.

2.2 Numerical models used for wind resource assessment

In the context of the wind energy industry, the decision to install a wind farm in a given location inevitably depends on the assessment of the region's long-term wind resource. These studies are carried out with measurement campaigns with meteorological masts positioned in strategic locations, which operate for a sufficiently long period to obtain accurate wind data representative of the local wind regime[7], combined with numerical CFD simulations or linear models. Results are then introduced into post-processing algorithms that produce results that characterise the wind resource in greater detail in the area where the wind turbines are intended to be installed. In this context, the most commonly used numerical models in the wind energy are described below.

2.2.1 Mesoscale models

These are CFD programs that resolve atmospheric flows in regions of hundreds or thousands of kilometers, in low-resolution calculation meshes, with resolutions in the order of 1-9 km. These are the models most used by meteorological institutes around the world, and that is why they are usually called NWP (Numerical Weather Prediction) models. They generally start from an initial condition and evolve over time, incorporating boundary conditions from planetary models, allowing the description of specific regional features of the meteorological state, such as cloud formation and precipitation distribution[8].

However, the low resolution of the calculation meshes is undesirable in situations of high orographic complexity because it does not capture the topographic profile in detail. To capture these effects, microscale models are used to simulate atmospheric flows at lower scales, using substantially more refined meshes.

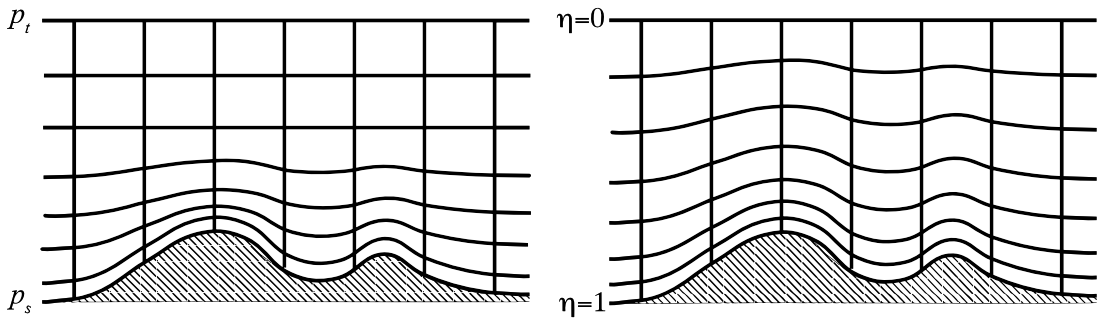
In terms of mesoscale models available in the wind industry, the most used is WRF-ARW (Weather Research and Forecasting with Advanced Research) [21], which is an open source model that integrates an Eulerian solver that accounts for the effects of solar radiation, convection phenomena, precipitation, ground surface heating, and the diurnal cycle [22]. Furthermore, it is capable of modelling compressibility and the effects of atmospheric instability. According to the theory proposed by Laprise [23], hydrostatic pressure is considered as an independent variable, which is why the model uses a coordinate system based on constant pressure levels [24], called η, and which are given by:

$$\eta = \frac{p - p_t}{p_s - p_t}, \tag{2.26}$$

where p_s and p_t represent, respectively, the pressure at the surface and at the top of the respective domain. The strategy of this coordinate system is illustrated in Figure 2.6.

[7] An additional step whereby local measurements are correlated with long-term data (MERRA2, ERA5, etc.) is often used to better estimate the local wind regime.

[8] Although these models were developed for weather forecasting for short time horizons, in the wind industry they are used to simulate the local wind regime in the past (hindcasting).



(a) (b)

Figure 2.6: Coordinate system used by WRF-ARW: (a) Traditional "*sigma*" coordinate system used in hydrostatic atmospheric models. (b) Hybrid coordinate system. Adapted from [24].

Regarding turbulence modelling, WRF-ARW uses the VLES technique (Very Large Eddy Simulations), which can be improved by introducing specific parametric models for the planetary boundary layer.

WRF has been the object of intensive study by several authors, namely in the development of WRF-ARW applications in planetary grids [25], and in meteorological simulations in the Arctic [26], where extreme climatic conditions served to place the model proof. The model has also been the subject of several sensitivity analysis studies to evaluate the quality of simulations when compared with measurement data[9].

2.2.2 Microscale models

Microscale models essentially consist of CFD codes that solve flows on high-resolution calculation grids. This way, they are able to capture the topographic profile of the terrain in greater detail, and characterise the local wind regime with greater accuracy. In the wind industry, besides OpenFOAM, the most used codes are:

- **WINDIE** - This is a CFD code developed by a team of researchers from the ISEP, that was born from the work developed by F. A. Castro [29]. It consists of a non-linear model that solves the Reynolds-averaged Navier-Stokes equations on terrain-following meshes, specially designed to capture complex physical phenomena such as flow separation, turbulence induced by complex topography, thermal effects, large flow deviations and shear, as well as other flow features such as those induced by neighbouring forested areas. It also incorporates a forest canopy model which has been validated in scientific literature, and it has the ability to perform

[9]Some studies used WRF-ARW to simulate the wind regime in the Iberian peninsula [27], and another study conducted in Portugal, which used WRF-ARW to analyse the wind resource in a strategic location with a high density of available energy [28]. Both studies revealed good correlations between simulation results and measured experimental data, although they reported that predictions are not very accurate near the ground, especially in complex terrain.

time-dependent simulations and can be coupled with mesoscale simulations[10].

- **Meteodyn** - It is a commercial CFD model, more focused on evaluating wind resources in a consultancy context. It also enables forest modelling, the contribution of thermal effects, and also the analysis of urban microclimates, particularly in terms of flow interactions with different types of infrastructure [30], although it uses simplified turbulence modelling.

- **WindSim** - It is a commercial CFD model widely used around the world, particularly by meteorologists, researchers, wind turbine manufacturers, project managers, government agencies, and academic institutions. You have the possibility of launching simulations using cloud computing with *Windsim Accelerator*, whose computational resources allows for significantly shorten processing times. In addition, it also has *Windsim Express*, which is a more basic and quick-to-use tool [31].

2.2.3 Linear models

The most commonly used linear model is WAsP (Wind Atlas Analysis and Application Program) [32], which makes use of the Jackson-Hunt linear theory [33] to calculate the local wind atlas[11], based on orography and roughness maps, and on wind data from measurement campaigns.

Then, assuming that the wind atlas is uniform across the entire area, WAsP does the reverse process and reintroduces the effect of terrain and roughness at the specific location of the turbines. Lastly, it calculates the Weibull distribution, average speed, and power density at each turbine.

Furthermore, WAsP also allows the application of the PARK model [35] to model the wake effect caused by wind turbines located upstream of others. Overall, it is concluded that linear models are quick and easy-to-use tools that produce good results in terms of preliminary wind resource assessment of relatively flat sites.

However, these models are not suitable in situations of high orographic complexity, they do not foresee recirculation and turbulence zones, and they do not provide information on the vertical component of the wind velocity.

[10] This technique is known as coupling, and consists of using results from mesoscale models to provide boundary conditions for microscale models in more specific locations. Microscale-mesoscale coupling has proven to be a good solution in several studies as it can improve the accuracy of final simulations. For this reason, it is a subject that has been the subject of study by several authors, who essentially seek to optimise coupling techniques and the interface with the respective microscale and mesoscale models.

[11] One of the most relevant examples is the new European atlas NEWA (New European Wind Atlas) [34], that already makes use of mesoscale (meteorological) models with results every 3 km, which are then used in the WAsP linear model to map the wind at a resolution of 50 m.

2.3 Mathematical models

2.3.1 Introduction

In this study of atmospheric flows, the simulations were approached using the mathematical model of incompressible flows, which is based on the continuity equation and the momentum conservation equations, also known as the Navier-Stokes equations, and which are the foundation of computational fluid mechanics. This area makes use of numerical methods, such as the finite volumes or finite elements method, to deal with the simulation of flows, heat transfer, and other physical phenomena such as chemical reactions, combustion, and electroacoustics.

Its applicability results from the need to solve and study this type of problems, since the Navier-Stokes equations do not have an analytical solution for complex geometries, as is the case in almost all real flows. The solution of these models consist of discretization techniques that approximate differential equations by systems of algebraic equations, which can be solved computationally, and the final result can only be reached as long as all the boundary conditions of the problem are known [36]. The concept consists of dividing a given complex geometry into small, simpler geometries, giving rise to a calculation mesh that is solved through an iterative process, until an equilibrium is reached in all control volumes and respective boundaries (convergence of the solution) [37, 38].

2.3.2 Fundamental laws

The laws of physics that underlie the solution to a problem involving computational fluid mechanics are based on the principles of conservation of mass and momentum, which are reflected in the Navier-Stokes equations, that describe the flow in time and space. These conservation laws can be derived by considering a given amount of matter or control mass (CM), and its extensive properties, such as mass, momentum and energy. For mass, the conservation equation is given by:

$$\frac{dm}{dt} = 0. \tag{2.27}$$

On the other hand, momentum can undergo changes due to the action of applied forces, and its conservation equation is given by Newton's second law:

$$\frac{d(m\mathbf{v})}{dt} = \sum \mathbf{f}, \tag{2.28}$$

where t represents the time, $\mathbf{v}$ the velocity vector, and $\mathbf{f}$ vector of the forces applied to the respective mass[12].

2.3.2.1 Reynolds transport theorem

The analysis method based on a defined mass is an approach that is typically used to study the dynamics of solid bodies, where the CM is easily identified. However, in the case of fluids it is more difficult to track a portion of mass along a flow. For this reason, it is more convenient to deal with the flow within a certain

[12]Symbols in bold, such as $\mathbf{v}$ or $\mathbf{f}$, represent vectors with three orthogonal components.

spatial region, called the control volume (CV). To this end, the analysis method can be converted mathematically using the Reynolds transport theorem. As a result of applying the theorem, extensive properties are transformed into intensive ones, which are independent of the amount of matter. For this, an intensive property that is conserved ϕ is considered, which can take the values of [2]:

$$\phi = \begin{cases} 1 & \text{for conservation of mass} \\ \mathbf{v} & \text{for conservation of momentum} \\ \phi^* & \text{for conservation of a scalar} \end{cases}, \tag{2.29}$$

where ϕ^* represents the value of a given conserved property per unit of mass. Thus, the corresponding extensive property Φ can be expressed according to the equation:

$$\Phi = \int_{V_{\mathrm{CM}}} \rho\phi \, dV, \tag{2.30}$$

where V_{CM} represents the volume V occupied by the control mass. Using the definition of the equation 2.30, the left side of the equations 2.27 and 2.28 for a given control volume can be rewritten in the form:

$$\frac{d}{dt}\int_{V_{\mathrm{CM}}} \rho\phi \, dV = \frac{d}{dt}\int_{V_{\mathrm{CV}}} \rho\phi \, dV + \int_{S_{\mathrm{CV}}} \rho\phi(\mathbf{v} - \mathbf{v}_S)\cdot\mathbf{n} \, dS, \tag{2.31}$$

where V_{CV} represents the volume occupied by the control volume, S_{CV} the surface S surrounding the control volume, $\mathbf{n}$ a unit vector normal to the surface S_{CV} with a direction away from the control volume, and $\mathbf{v}_S$ the velocity of the surface S_{CV}. For a control volume fixed in space, it turns out that $\mathbf{v}_S = 0$, and the first derivative on the right-hand side becomes a partial derivative. This equation states that the rate of change of a property in the control mass is the rate of change of the same property within the control volume plus its transport by advection the across the boundary.

2.3.2.2 Conservation of mass

The integral form of the conservation of mass equation, also known as the continuity equation, results from equation 2.31 with $\phi = 1$, given by:

$$\frac{\partial}{\partial t}\int_V \rho \, dV + \int_S \rho\mathbf{v}\cdot\mathbf{n} \, dS = 0, \tag{2.32}$$

with $V \equiv V_{CV}$ and $S \equiv S_{CV}$ for conciseness. Applying the Gaussian divergence theorem to the convective term, it is possible to transform the surface integral into a volume integral, therefore equation 2.32 can be rewritten in vector format:

$$\frac{\partial\rho}{\partial t} + \nabla\cdot(\rho\mathbf{v}) = 0. \tag{2.33}$$

In most problems, including atmospheric flows, the fluid density is assumed to be constant[13], changing the mass conservation equation to:

$$\nabla\cdot\mathbf{v} = 0, \tag{2.34}$$

where the term $\nabla\cdot\mathbf{v}$ represents the divergence of the velocity field.

[13]This assumption is not only valid in liquids, which are typically considered incompressible, but also in gases if the Mach number is less than 0.3.

2.3.2.3 Conservation of momentum

The integral form of conservation of momentum equation can also be rewritten applying the Reynolds transport theorem, making $\phi = \mathbf{v}$ in equation 2.31:

$$\frac{\partial}{\partial t}\int_V \rho\mathbf{v}\, dV + \int_S \rho\mathbf{v}\mathbf{v}\cdot\mathbf{n}\, dS = \sum \mathbf{f}. \tag{2.35}$$

The right side of the equation represents the forces that typically act on control volumes, that can be pressure forces, normal or shear stresses, surface tensions, gravity, centripetal and Coriolis forces, and electromagnetic forces [39]. For a Newtonian fluid, the stress tensor $\mathcal{T}$, which represents the molecular rate of momentum transport, can be written as:

$$\mathcal{T} = -\left(p + \frac{2}{3}\mu\nabla\cdot\mathbf{v}\right)\mathcal{I} + 2\mu\mathcal{D}, \tag{2.36}$$

where μ is the dynamic viscosity of the fluid, $\mathcal{I}$ the unit tensor and $\mathcal{D}$ the strain rate tensor, given by:

$$\mathcal{D} = \frac{1}{2}\left[\nabla\mathbf{v} + (\nabla\mathbf{v})^T\right]. \tag{2.37}$$

Even so, the equations 2.36 and 2.37 can be rewritten in tensorial notation, also known as Einstein notation, in the following form:

$$T_{ij} = -\left(p + \frac{2}{3}\mu\frac{\partial u_j}{\partial x_j}\right)\delta_{ij} + 2\mu D_{ij}, \tag{2.38}$$

$$D_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right). \tag{2.39}$$

For incompressible flows, the second term in parentheses in the equation 2.38 is equal to zero by the conservation of mass equation. The following notation is often used in the literature to describe the viscous component of the stress tensor τ_{ij}, which is given by:

$$\tau_{ij} = 2\mu D_{ij} - \frac{2}{3}\mu\delta_{ij}\nabla\cdot\mathbf{v}. \tag{2.40}$$

Considering the forces acting per unit of mass, represented by $\mathbf{b}$, the integral form of the equation 2.35 is given by:

$$\frac{\partial}{\partial t}\int_V \rho\mathbf{v}\, dV + \int_S \rho\mathbf{v}\mathbf{v}\cdot\mathbf{n}\, dS = +\int_S \mathcal{T}\cdot\mathbf{n}\, dS + \int_V \rho\mathbf{b}\, dV. \tag{2.41}$$

Applying the Gauss divergence theorem, the surface integral is transformed into a volume integral, leaving equation 2.41 in the form:

$$\frac{\partial(\rho\mathbf{v})}{\partial t} + \nabla\cdot(\rho\mathbf{v}\mathbf{v}) = \nabla\cdot\mathcal{T} + \rho\mathbf{b}. \tag{2.42}$$

Typically, equation 2.42 is rewritten as a function of the i-Cartesian component, whose coordinate system is oriented as a function of a certain fixed flow direction.

This strategy allows one to put the equation in *strong conservation* form[14], which is given by:

$$\frac{\partial(\rho u_i)}{\partial t} + \nabla \cdot (\rho u_i \mathbf{v}) = \nabla \cdot \mathbf{t}_i + \rho b_i, \tag{2.43}$$

where u_i and $\mathbf{t}_i$ represent, respectively, the velocity component u and the stress tensor $\mathcal{T}$ in the x_i direction of the Cartesian reference frame. Replacing the viscous stress component of equation 2.40 rewritten in Cartesian indices notation in the equation 2.43, and admitting that gravity is the only external force that acts on the control volume, we obtain:

$$\frac{\partial(\rho u_i)}{\partial t} + \frac{\partial(\rho u_j u_i)}{\partial x_j} = \frac{\partial \tau_{ij}}{\partial x_j} - \frac{\partial p}{\partial x_i} + \rho g_i, \tag{2.44}$$

where g_i is the component of gravitational acceleration g in the direction of the Cartesian coordinate x_i.

As only the pressure gradient appears in the equation 2.44, the absolute value of the pressure is not important, except in compressible flows, including some atmospheric flows, oceanic flows, and flows with a free surface exposed to the atmosphere. For flows with small and smooth changes in fluid density, the term ρg_i can be divided into two components: $\rho_0 g_i + (\rho - \rho_0) g_i$, where ρ_0 represents a reference density. The first component can be included with the pressure, and if the change in density is retained only in the gravitational term, the Boussinesq [2] approximation is typically used.

2.3.2.4 Methods to solve the Navier-Stokes equations

As mentioned previously, only a few analytical solutions are known for the Navier-Stokes equations, typically for flows in very simple geometries. In contrast, and in the presence of more complex flows, the mathematical treatment of these equations becomes extremely complex as a consequence of non-linearity, as it admits multiple solutions to the same problem (non-unique solution), and due to the instability associated with turbulence. For these and other reasons, the solution of the Navier-Stokes equations is part of *The Millennium Prize Problems* [40], which represent a set of mathematical problems that remain unsolved.

However, and despite their analytical complexity, the Navier-Stokes equations can be computationally modelled in a refined calculation grid using CFD models, to obtain approximate and realistic solutions for a variety of viscous flows [39].

There are several methods for solving the Navier-Stokes equations. In this work, the tool used to run the simulations in OpenFOAM is based on the SIMPLE

[14]The momentum conservation equations are said to be in *strong conservation* form if all terms have the divergence form of a vector or tensor. In terms of numerical simulations, when equations in the form of *strong conservation* are used with the finite volume method, conservation of momentum in the calculation is automatically guaranteed. This feature is important because it helps ensure that the numerical method does not diverge during solving [2].

algorithm, which makes use of an implicit pressure correction method. The method presupposes a system of equations with the following form:

$$A_P^{u_i} u_{i,P}^{n+1} + \sum_l A_l^{u_i} u_{i,l}^{n+1} = Q_{u_i}^{n+1} - \left(\frac{\delta p^{n+1}}{\delta x_i} \right)_P , \tag{2.45}$$

where A_P and A_l are coefficients at the calculation point P and and its neighboring nodes used in calculating the derivatives in the convective and diffusive terms, and term Q is a source term that may include external forces, linear terms, and any terms that may have an explicit treatment [2]. The mass conservation equation is transformed into an equation for pressure correction, which is solved in an iterative process with the momentum conservation equation, until a non-divergent velocity field is obtained which, together with the corrected pressure, obeys equation 2.44. Replacing index $n+1$ with index m, which represents an iteration, u_i^m starts to represent an estimate of u_i^{n+1}, and the equations are in the form:

$$A_P^{u_i} u_{i,P}^{m^*} + \sum_l A_l^{u_i} u_{i,l}^{m^*} = Q_{u_i}^{m-1} - \left(\frac{\delta p^{m-1}}{\delta x_i} \right)_P . \tag{2.46}$$

2.3.3 Turbulence modelling

2.3.3.1 Introduction

Turbulent flows are highly unstable, and have fast three-dimensional variations in the instantaneous velocity field, that contain intense vorticity, which makes their mathematical modelling difficult.

Through these processes, the interaction between eddies causes agitation, thus promoting the mixing of the various portions of fluid. At this point, and through the action of molecular viscosity, the kinetic energy of the various flows is converted into internal energy of the fluid, thus reducing velocity gradients. For this reason, mixing is considered as a turbulent dissipative process. However, turbulent flows differ in terms of size, strength, and time interval between occurrences, making their direct simulation very complex.

In this sense, certain studies [41] make the distinction between *turbulence simulation* and *turbulence model*. In a *turbulence simulation*, the equations are solved for a time-dependent velocity field, which at one point represents the velocity field $U(x,t)$ of the turbulent flow. On the other hand, in a *turbulence model* the equations are solved for certain average quantities. Thus, the two most used *turbulence simulation* approaches are DNS (Direct Numerical Simulation) and LES (Large-Eddy Simulation), whose main characteristics are:

- In DNS, which is conceptually the technique with the simplest approach, the Navier-Stokes equations are solved to determine $U(x,t)$ in all parts of the fluid. However, as all length and time scales are resolved, DNS becomes a computationally expensive process, the cost of which increases depending on the Reynolds number by Re^3. Therefore, this technique is typically applied to simpler flows with moderate Reynolds numbers.

- In LES, the Navier-Stokes equations are solved for a filtered velocity field $\langle U \rangle(x,t)$, which is representative of larger-scale turbulent motions. The solved equations include a sub-grid scale model to account for the influence of smaller-scale motions, which are not directly represented.

In terms of the *turbulence model*, the Navier-Stokes equations are solved using statistical average quantities to obtain an average velocity field $\overline{U}(x,t)$. This technique is known as RaNS (Reynolds-averaged Navier-Stokes), and is an especially useful strategy in problems with high Reynolds numbers. However, in these time average procedures, the non-linearity of the Navier-Stokes equations gives rise to additional terms that also require a modelling process, as is the case of Reynolds stresses, represented by σ_{ij}, and which are calculated with the equation:

$$\sigma_{ij} = -\rho\overline{u_i' u_j'} = -\frac{2}{3}\rho k \delta_{ij} + \mu_t \left(\frac{\partial \overline{u}_i}{\partial x_j} + \frac{\partial \overline{u}_j}{\partial x_i} \right), \tag{2.47}$$

where μ_t is the turbulent viscosity.

2.3.3.2 RaNS technique

The RaNS technique describes the entire turbulence spectrum based on a turbulence model, in which Reynolds stresses can be modelled according to two typical approaches [41]:

1) From a turbulent viscosity model, where the turbulent viscosity μ_t can be obtained from a mathematical relationship, or from turbulence quantities, such as the turbulent kinetic energy k or its dissipation rate ε, for which specific transport equations are solved.

2) From Reynolds stress models, where the model transport equations are solved for individual Reynolds stresses and for another turbulent quantity, such as for the dissipation rate ε, thus eliminating the need for turbulent viscosity μ_t.

In either case, the technique requires a variable decomposition process, called Reynolds decomposition [42]. In its most classic variant, the Reynolds decomposition uses time averages, where a generic variable $\phi(t)$ is decomposed into a time-averaged value $\overline{\phi}$ and its instantaneous fluctuation $\phi'(t)$, resulting in the expression:

$$\phi(x_i,t) = \overline{\phi}(x_i) + \phi'(x_i,t). \tag{2.48}$$

For stationary flows, where the respective time-averaged value $\overline{\phi}$ is approximately constant, its value is given by:

$$\overline{\phi}(x_i) = \lim_{T\to\infty} \frac{1}{T} \int_0^T \phi(x_i,t)\, dt. \tag{2.49}$$

where t represents the instant of time, and T a time period large enough to contain all relevant periods of turbulence fluctuations. In graphical terms, the strategy is represented in Figure 2.7.

There are also other possibilities of decomposition more oriented to unstable flows, as the ensemble averages known as URaNS (Unsteady Reynolds-averaged Navier-Stokes) [43] or the transient averages TRaNS (Transient Reynolds-averaged

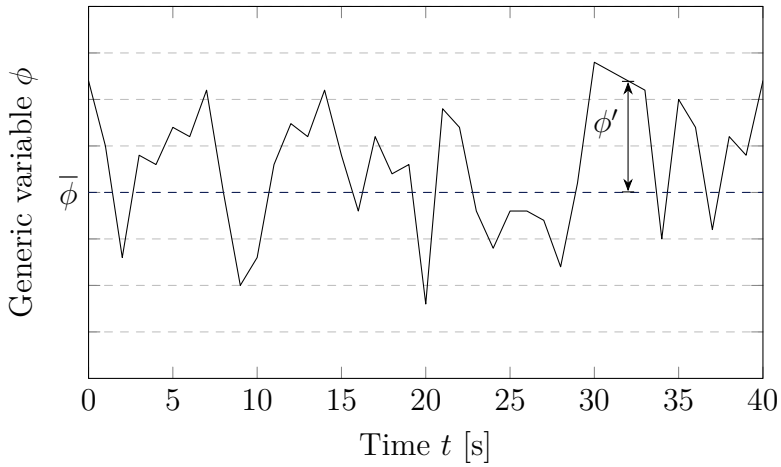


Figure 2.7: Graphical representation of Reynolds decomposition for a generic variable representing turbulence fluctuations. Adappted from [42].

Navier-Stokes) [44]. In these situations, the time-averaged value $\overline{\phi}$ is not constant, thus time-averaging cannot be used unless the time scale of the respective turbulence model is adjusted for the time period average $T < \infty$. In most of these cases, the flow is solved using the URaNS technique, whose concept for calculating the temporal average value is given by:

$$\overline{\phi}(x_i, t) = \lim_{N \to \infty} \frac{1}{N} \sum_{n=1}^{N} \phi(x_i, t), \tag{2.50}$$

where N is the number of points in the set, and must be large enough to eliminate the effects of random turbulence fluctuations.

However, certain studies [45] report that the computational resources needed to calculate flows with RaNS models are quite modest, which has made this approach the basis for calculating several types of flows in all engineering fields. This will be the approach used in this work.

2.3.3.3 The $k - \varepsilon$ turbulence model

The standard $k - \varepsilon$ turbulence model [46] belongs to the class of two-equation models [47], in which the transport equations are solved for two quantities of turbulence, namely the turbulent kinetic energy k and its dissipation rate ε. In this model, the turbulent viscosity μ_t is computationally modelled using the equation:

$$\mu_t = \rho C_\mu \frac{k^2}{\varepsilon}, \tag{2.51}$$

where C_μ represents a dimensionless constant that relates the turbulent shear stress to the turbulent kinetic energy k, with the mathematical relation:

$$k = \frac{1}{2}\left(\overline{u'^2} + \overline{v'^2} + \overline{w'^2}\right) = \frac{1}{2}\overline{u_i' u_j'}. \tag{2.52}$$

The dissipation rate of turbulent kinetic energy ε can be estimated by solving two additional transport equations [42], expressed here for an incompressible fluid:

$$\rho\frac{\partial(\overline{u}_j k)}{\partial x_j} = \frac{\partial}{\partial x_j}\left[\left(\mu + \frac{\mu_t}{\sigma_k}\right)\frac{\partial k}{\partial x_j}\right] + P_k - \rho\varepsilon, \tag{2.53}$$

$$\rho\frac{\partial(\overline{u}_j \varepsilon)}{\partial x_j} = \frac{\partial}{\partial x_j}\left[\left(\mu + \frac{\mu_t}{\sigma_\varepsilon}\right)\frac{\partial \varepsilon}{\partial x_j}\right] + \frac{C_{\varepsilon 1}\varepsilon}{k}P_k - \frac{C_{\varepsilon 2}\rho\varepsilon^2}{k}, \tag{2.54}$$

where P_k represents the mechanical production of turbulence, given by:

$$P_k = \sigma_{ij}\frac{\partial \overline{u}_i}{\partial x_j} \approx \mu_t\left(\frac{\partial \overline{u}_i}{\partial x_j} + \frac{\partial \overline{u}_j}{\partial x_i}\right)\frac{\partial \overline{u}_i}{\partial x_j}. \tag{2.55}$$

Quantities C_μ, σ_k, σ_ε, $C_{\varepsilon 1}$ and $C_{\varepsilon 2}$ represent empirical and dimensionless constants that are obtained by calibrating the $k-\varepsilon$ model. Two sets of typical values are shown in the Table 2.5, the first being suggested for Standard applications (typically for flat jets and mixing layers) [46], and the second more suitable for atmospheric flows [48].

Table 2.5: Typical constants of the model $k-\varepsilon$

Model	C_μ	σ_k	σ_ε	$C_{\varepsilon 1}$	$C_{\varepsilon 2}$
Standard	0.090	1.00	1.30	1.44	1.92
Atmospheric	0.033	1.00	1.85	1.44	1.92

2.3.3.4 Other RaNS models

Although the $k-\varepsilon$ model is the most used to model turbulence in CFD simulations, including in the study of atmospheric flows, the accuracy of the results decreases for complex flows [41]. In certain cases, the standard $k-\varepsilon$ model has some limitations, namely in wall regions or flows with high pressure gradients [45]. For this reason, there are other models that also revealed good results for certain specific characteristics of flows, such as:

- The *Realizable* $k-\varepsilon$ model [49], which produces better results for three-dimensional flows with high pressure gradients in the boundary layer. In this formulation, the constant C_μ becomes a function of the local strain rate and vorticity.
- The *RNG* $k-\varepsilon$ model [45], which makes use of mathematical tools from statistical mechanics to systematically remove the effects of small-scale turbulence from the Navier-Stokes equations, expressing its effect in terms of larger-scale movements and a modified viscosity.
- The $k-\omega$ model [50], similar to the $k-\varepsilon$ model, that also describes turbulence based on the turbulent viscosity μ_t and the turbulent kinetic energy k, but replaces the dissipation rate ε by the specific dissipation rate, $\omega \equiv \varepsilon/k$. For flows in the boundary layer, the $k-\omega$ model proved to be superior both in treating the viscous region close to walls, and in accounting for the effects of pressure gradients in the direction of the flow direction [41].

In addition to these models, there are other less conventional, such as the *Menter shear stress transport SST* $k-\omega$ model, and the one-equation Spalart-Allmaras, which are typically more used in aerospace engineering applications.

Still, many authors have dedicated their time to studying these turbulence models to find coefficients that more accurately represent the atmospheric flows, particularly for the $k-\varepsilon$ model [51, 52].

2.4 OpenFOAM

2.4.1 Introduction

OpenFOAM, acronym for Open Field Operation and Manipulation [1], is an open source CFD software package developed in C++ language, which contains several tools and utilities capable of performing multi-physics numerical simulations for various engineering problems [53].

OpenFOAM uses the finite volume method to discretize the fundamental equations of various physical processes, allowing the simulation of a wide range of fluid dynamics phenomena[15]. It has also other built-in tools for mesh generation and manipulation, parallel processing, pre-processing and post-processing utilities for data extraction and visualisation like paraFoam. In addition, it has several tutorial folders with important code comments, giving the user a starting point to adapt a typical case to its specific problem.

Nowadays, its been regularly used in academic and research contexts, due to the fact that it has a strong component of versatility and the possibility of editing models according to specific needs. The atmospheric flows field has contributed to the development of new tools in OpenFOAM, motivated by the exponential growth of the wind industry, namely by the need to create models that can simulate the local wind regime with high precision, even in cases with great influence from adverse conditions.

2.4.2 Typical case structure

A typical simulation folder in OpenFOAM contains three main subfolders:

1) `0` folder - Within this directory, you will find the files that specify the boundary conditions of the computational domain. Each physical quantity involved (such as pressure, velocity, turbulent kinetic energy, etc.) is represented by a separate file, and within each file, the conditions for each boundary of each face are specified.

2) `constant` folder - This directory includes the files that outline the geometry of the case and its computational domain, as well as certain constants to be used in the transport equations, such as modelled transport properties. One of the most important subfolders is `polyMesh` that contains all the geometric information necessary to define the domain.

[15] This include physical phenomena such as laminar and turbulent flows in compressible or incompressible regimes, turbulence modelling with RaNS models, LES and DNS, heat transfer, chemical reactions, molecular dynamics, combustion, phase change, among others.

3) `system` folder - This folder contains the files that specify the convergence criteria for solvers, numerical schemes, domain decomposition for parallel processing, and data extraction for postprocessing, among other things. It can also contain the `fvOptions` file, which is a powerful tool used to apply additional source terms, constraints, or specific modelling features to the governing equations. This file allows users to include effects like porous media resistance, body forces, heat sources, momentum sinks, or other custom modifications to the simulation without altering the core solver.

Figure 2.8 illustrates the structure and content of an OpenFOAM case folder. This particular case involves simulating an adiabatic, incompressible flow using the $k - \varepsilon$ turbulence model with parallel processing.

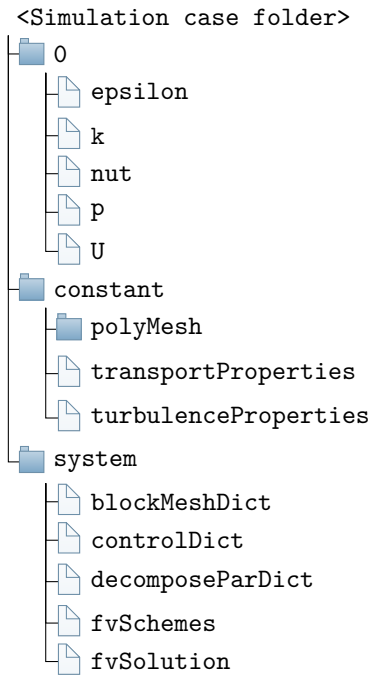


Figure 2.8: Example of an OpenFOAM typical case folder structure and its content

2.4.3 Tools for mesh generation in complex topography

When the wind resource of a given region is analysed, correct topographic mapping and its representation in the numeral mesh are essential, especially in the case of terrain with mountains or high orographic complexity. The quality of simulation results increases depending on the accuracy with which the terrain profile is captured, that must be as close to reality as possible.

In OpenFOAM, the mesh that contains the terrain profile is created by running the *blockMesh* utility, which is used to generate structured hexahedral meshes for CFD simulations. This tool creates a mesh by defining blocks, that are subdivided into cells, based on user-specified vertices, edges, and faces, which are defined in `blockMeshDict` file. In terms of applicability, *blockMesh* is particularly suited to generate grids with simple or regular geometries, such as rectangular or cubic shapes.

In such cases, the mesh can be easily divided into structured hexahedrons, resulting in a high quality grid. However, for more complex geometries, such as a topographic profile, it is not the most suitable tool to adequately fit the mesh to the geometry, unless there is a pre-processing routine that writes a specific mesh structure in `blockMeshDict` file.

For numerical mesh generation of a complex terrain, OpenFOAM has two traditional tools that are typically used for this type of problem, namely *snappyHexMesh* and *extrudeMesh*, whose characteristics are:

(i) *snappyHexMesh* is a tool that allows one to sculpt a previously generated mesh block according to the information of the Cartesian points xyz from an STL (STereoLithography) file. Its main features are the ability to generate hexagonal meshes capable of dealing with complex geometries, and adaptive mesh refinement, making it possible to change the density of control volumes in regions of greater interest, where resolution is more critical. However, studies have shown that the final result of *snappyHexMesh* is highly dependent on the available computational resources and the user's knowledge to adjust the parameters configured in the `snappyHexMeshDict` file for each topography [54].

(ii) *extrudeMesh* is a tool that allows one to vertically extrude a surface previously generated from an STL file. Studies have shown that the appearance of mesh close to the ground is significantly better when compared to *snappyHexMesh*. However, the total number of cells obtained in the final mesh is double that expected, due to the intrinsic triangular structure of the STL format, which increases the need for high computational resources.

Therefore, it was decided to use *blockMesh* together with tools and techniques developed in [54], that consist of code capable of reading the VTK (Visualization ToolKit) file that contains the topographic map information, and translating its xyz data into OpenFOAM `blockMeshDict` format. Prior to this procedure, xyz terrain data is manipulated with *gsrf3* (*gsurf* successor variation), an ISEP *in-house* code for UTM (Universal Transverse Mercator) coordinate transformation, that allows for coordinate grid rotation, grading of nodal spacing in a geometric expansion basis and flattening of domain borders for calculations smoothing [54].

Implementing these techniques, OpenFOAM folder structure now has additional folders that contain these tools, specially design for the simulation of atmospheric flows over complex terrain. Its new content is hereafter described, with the respective tree view shown in Figure 2.9.

- `<Simulation case folder>/0`

  In addition to the native OpenFOAM files that define model boundary conditions, this folder contains `z` and `z0` files which are used to prompt paraFoam to load two additional scalar fields. These fields allow for the visual inspection of (i) height ASL (Above Sea Level) of ground patch and (ii) modelled ground roughness boundary conditions. With the developed procedures, `k`, `epsilon` and `omega` files are now generated by the new code *write_bCs*, and `nut`, `z` and `z0` generated by the new code *write_z0*.

- `<Simulation case folder>/0/dat`

  The `.dat` files located in this folder contain the data called by homonym files in folder `<Simulation case folder>/0` via an `#include` instruction in the respective patch definition.

- `<Simulation case folder>/constant`

  With the developed procedures, `turbulenceProperties` file is now generated by the new code *write_turbulenceProperties*.

- `<Simulation case folder>/system`

  With the developed procedures, `blockMeshDict` file is now generated by the new code *write_blockMeshDict*[16].

- `<Simulation case folder>/_postproc`

  Contains postprocessing tools, although it was not used in this work.

- `<Simulation case folder>/_preproc`

  This folder contains the new preprocessing tools and procedures. Besides the subfolders hereafter presented, it stores the executables *write_bCs*, *write_blockMeshDict*, *write_turbulenceProperties* and *write_z0*. It also contains the configuration file `_preProcDict`, that is responsible for setting preprocessing parameters.

  - `<Simulation case folder>/_preproc/_gsurf_WINDIE`
    Contains the FORTRAN source code of *gsrf3* and *groug*, another ISEP *in-house* code similar to *gsrf3* used for UTM coordinate transformation of roughness mapping instead of height.

  - `<Simulation case folder>/_preproc/plot`
    Stores raw `.dat` files generated by *write_bCs* allowing for the plot of boundary conditions profiles at *inlet* patch.

  - `<Simulation case folder>/_preproc/src`
    Contains the FORTRAN source code of the developed preprocessing tools. For each source code, the folder also contain a `Makefile` for code compilation via a *make* instruction in the terminal, and a *make clean* instruction to clean all files.

  - `<Simulation case folder>/_preproc/utils`
    This directory contains two GNU Octave scripts: (i) `utils.m`, which is used to compute mesh parameters such as the expansion factor R, and (ii) `terraintoolcsv2raw.m`, a utility for converting topographic xyz data in CSV (Comma Separated Values) format into a raw ASCII file.

  - `<Simulation case folder>/_preproc/vtk`
    Contains the VTK file with topographic data to be read by *write_blockMeshDict*, generated by *gsrf3*.

[16] According to [54], *write_blockMeshDict* code takes less than 1 second to run and write `blockMeshDict` file, and clock time of *blockMesh* utility was approximately 1/20 when compared to *snappyHexMesh* and almost equal to *extrudeMesh* clock time.

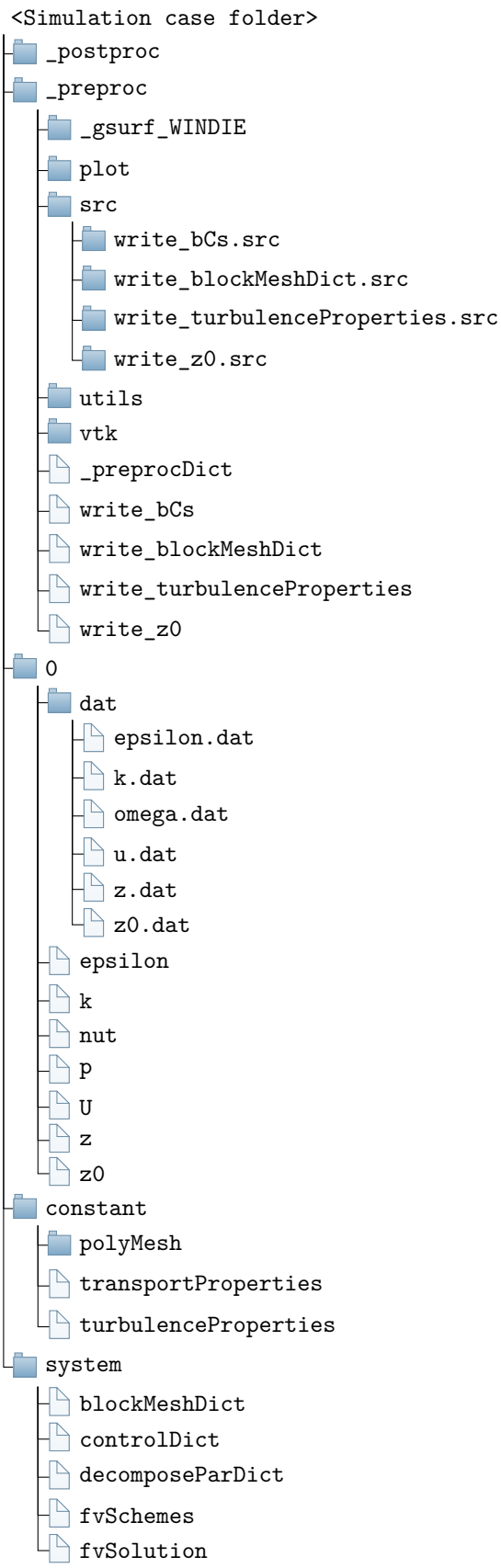


Figure 2.9: Example of an OpenFOAM case folder structure and its content with new tools and procedures specially designed for the simulation of atmospheric flows over complex terrain [54]

Chapter 3

Case study: Mörknässkogen wind farm

3.1 Introduction

In this Chapter we present a description of the wind farm that was subject of analysis, which is located in Mörknässkogen, Finland. The wind farm consists of four identical 4.6 MW turbines, with a hub height of 166 m AGL and a rotor diameter of 160.0 m, which were strategically installed based on previous CFD simulations and data obtained from meteorological mast i690 installed in the region, whose measurement campaign took place between 24/05/2014 and 07/04/2016. This data was analysed to produce wind roses of frequencies, turbulent intensity, average velocity, energy contribution and Weibull curves representing the local wind regime, to identify the predominant wind direction and adverse conditions. In terms of topography, land coverage was analysed to assess the orographic complexity and terrain roughness, particularly in forested areas.

3.2 Geographical location

Mörknässkogen wind farm is located in western Finland, approximately 50 km northeast of the city of Vaasa, in the municipality of Vöyri (Vörå), in the region of Ostrobothnia. It is located 6 km east of the Gulf of Bothnia, at the village coast of Oravais, as shown in Figure 3.1.

Figure 3.1: Geographical location of Mörknässkogen wind farm

Meteorological mast i690 used to estimate wind conditions is located approximately 3.5 km to the northwest of the wind farm. Both mast and turbines coordinates are presented in Table 3.1 and also illustrated in Figure 3.2, and they are considered to be located in gently undulating terrain, with terrain height varying from 30 to 40 m ASL.

Regarding wind flows, the main source of complexity comes therefore, not from the orographic profile, but from terrain cover in the form of very tall evergreen pine and fir trees, whose canopy height can reach 30 m AGL. These trees have a large presence in the region, not only in the wind farm itself but also in the entire surroundings, thereby contributing to a high forest density that is a significant factor when predicting the wind resource.

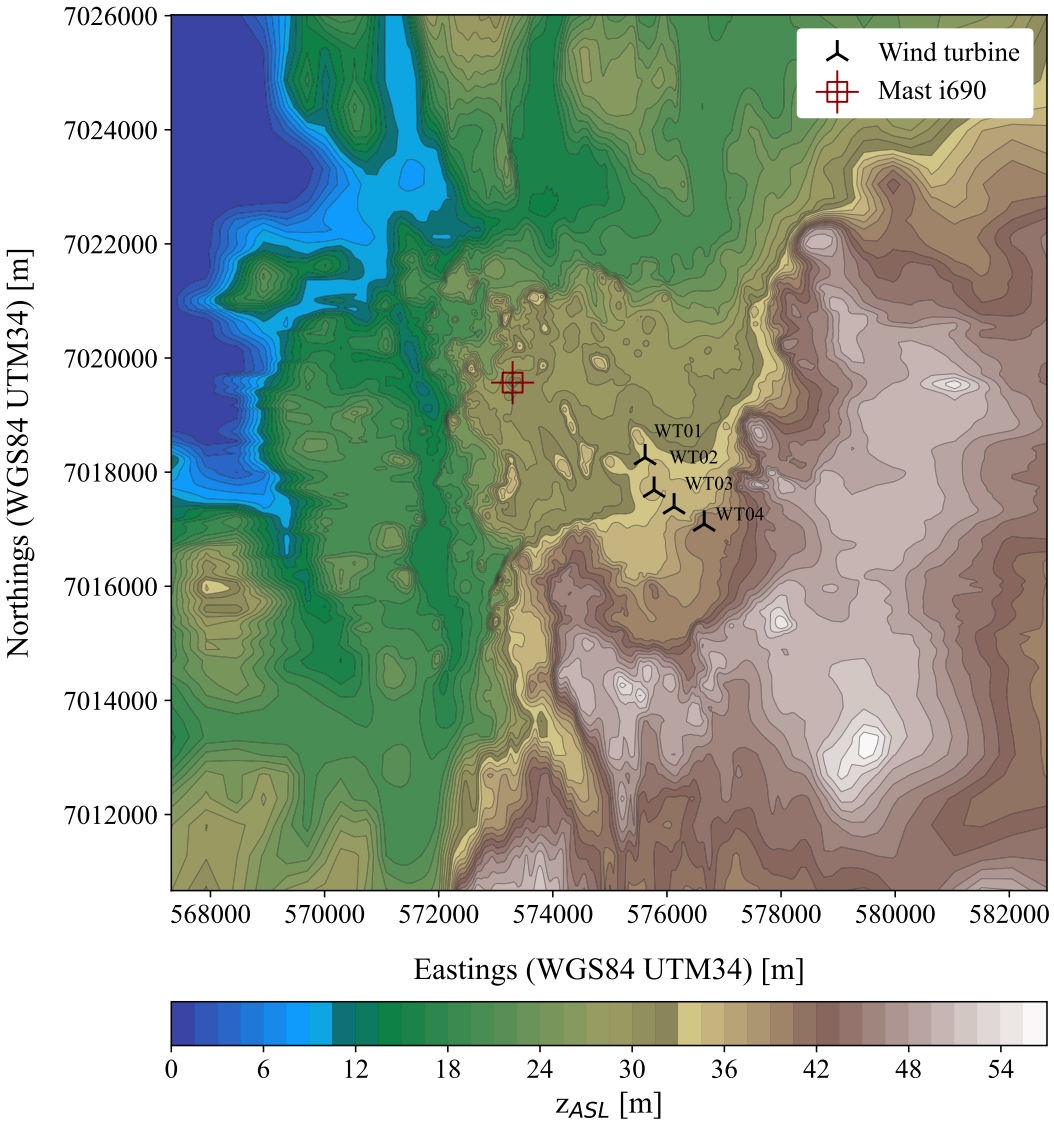


Figure 3.2: Geographical location of mast and turbines

In terms of land coverage, all terrain points are surrounded or very close to forested areas, including the mast. This terrain configuration is expected to interfere with the simulation results due to significant changes in the vertical wind speed profile, specially in forested areas, also causing a noticeable impact on turbulence, flow inclination and shear factor levels as a result.

Table 3.1: Geographical location of mast and turbines. Northings and Eastings coordinates are presented following the WGS84, UTM34 format. h_{hub} is the hub height, d_{mast} the linear distance between mast i690 and the corresponding wind turbine, and Δz_{hub} the elevation between turbine hub height and mast i690 measurement height at 120 m AGL.

Equipment	**Northings** [m]	**Eastings** [m]	h_{hub} [m]	Δz_{hub} [km]	d_{mast}
Mast i690	573295.0	7019569.0	-	-	-
Turbine WT_{01}	575616.7	7018255.5	166.0	33.2	2.67
Turbine WT_{02}	575774.7	7017685.0	166.0	34.7	3.11
Turbine WT_{03}	576125.2	7017393.5	166.0	35.0	3.57
Turbine WT_{04}	576652.6	7017091.0	166.0	40.0	4.17

For this reason, in sets of simulations that do not have a forest model, the cross-predictions conducted in the mast that are used to validate the mathematical model may present high deviations compared to those expected for this type of studies. This problem will be a topic of discussion later in this thesis.

3.3 Land use and surface roughness

Land use and surface roughness can be classified into three main types, considering the topographic survey that was performed on site, and the roughness map provided for this work:

1) Water with very low aerodynamic roughness for scattered small lakes and the Gulf of Bothnia.

2) Agricultural fields and small crops, with low surface roughness varying between typical values of $z_0 = 0.05$ m to $z_0 = 0.1$ m.

3) Forested areas covering large tracts of land, assigned with roughness values of $z_0 = 0.7$ m.

Outside of this general classification, small shrubs or low-height trees can also be identified along the land cover, which can be included in the aerodynamic roughness range between $z_0 = 0.1$ m and $z_0 = 0.7$ m depending on their impact on the ABL flow. Residential areas are small, sparse and made up of low-rise buildings and were not subject of explicit characterisation in the surface roughness maps.

The roughness map is shown in Figure 3.3, where the delimitation of the computational domain defined for the CFD simulations performed in Chapter 4 and Chapter 5 is also shown, which is enlarged in Figure 3.4. The roughness map analysis reinforces the strong presence of forest in the region, where it can be seen that both the mast i690 and the wind turbines are installed in a forest-surrounded site, which contributes to an increase in turbulence and wind shear at lower altitudes.

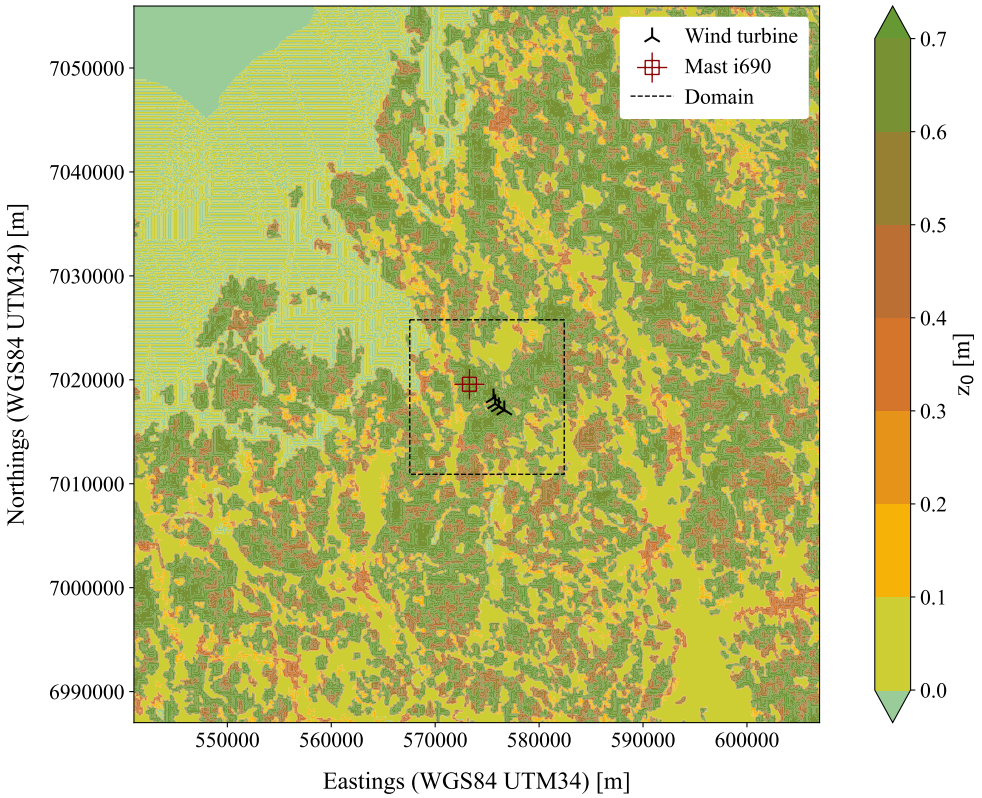


Figure 3.3: Forest canopy and terrain roughness map of the region around Mörknässkogen wind farm.

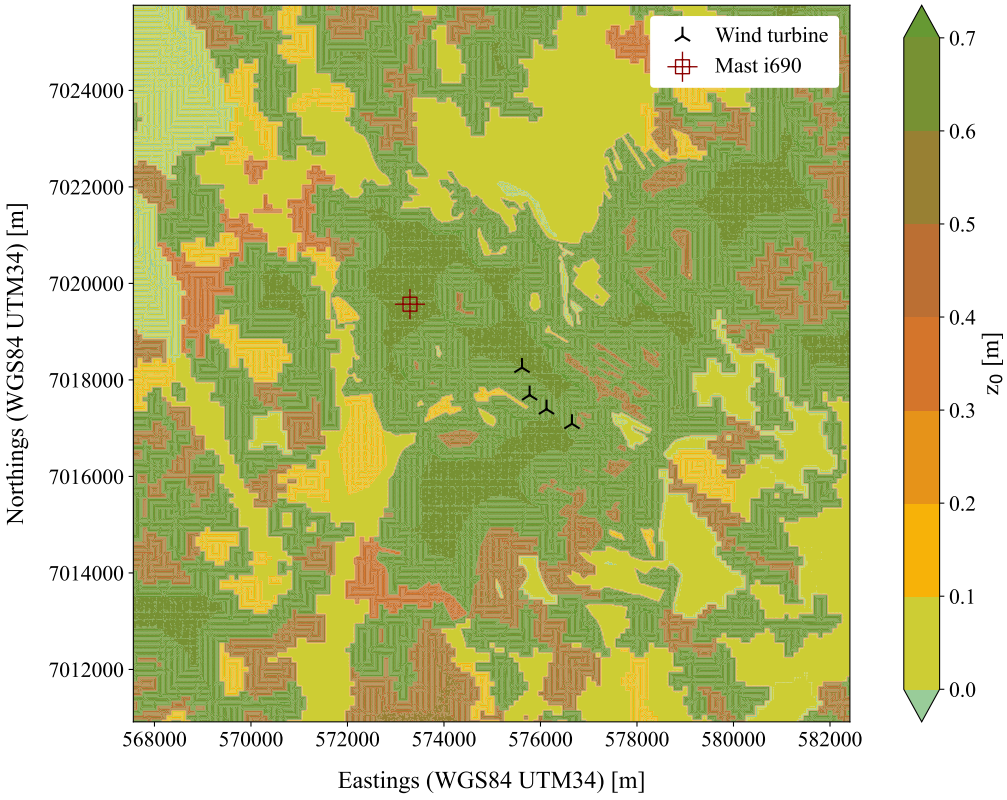


Figure 3.4: Forest canopy and terrain roughness map of Mörknässkogen wind farm within the computational domain used in the CFD simulations

3.4 Wind data and assessment

3.4.1 Data structure

As mentioned in the introduction Section, the measurement campaign that provided wind data for the wind resource assessment took place between 24/05/2014 and 07/04/2016, which was about 2 years of local measurements. During that time period, the control system produced a total of 105264 records, obtained with sampling rates of 1 Hz transformed into 10 minute averages as is standard practice in the wind industry.

Mast i690 that was installed on site consisted of a 116 m lattice tower, equipped with anemometers installed on side booms with opposite orientation at heights 120, 82 and 48 m, all Thies Clima First Class Advanced sensors. Wind direction was measured by two Thies Clima First Class wind vanes mounted at heights 119.3 and 47.3 m. There were also present two Schneider/Ekker TEC1008 temperature sensors and a barometer Schneider/Ekker EK-APS21-1 at heights 115.5, 20 and 1 m, respectively. The data logger was a Schneider Wexile 4200.

All recorded measurements were compiled and recorded in three text files, one for each height (120, 82, and 48 m AGL), that were provided for the development of this work. These files, later called `DataLog_120m_SIM.txt`, `DataLog_82m_SIM.txt`, and `DataLog_48m_SIM.txt` respectively, have a structure as shown in Figure 3.5.

```
i690 - Oravais
May 2014 - April 2016
105264 records
Est,Dir,Vavg,Vmax,Vmin,Vstd,YYYYMMDDhhmm,
i690,999,99.99,99.99,99.99,99.99,201405010000,
⋮
i690,240.4,7.29,8.26,6.3,0.372,201509192240,
i690,237,7.84,8.78,6.89,0.387,201509192250,
i690,235.5,8.25,9.08,6.76,0.397,201509192300,
⋮
i690,999,99.99,99.99,99.99,99.99,201604302350,
```

Figure 3.5: Example of the wind data structure inside `Datalog.txt` file. Data per line for 10-minute intervals : mast ID, average wind direction, average wind speed, maximum wind speed, minimum wind speed, wind speed standard deviation, date and time in `YYYYMMDDhhmm` format.

However, some records cannot be validated for analysis due to sporadic errors, as happened in the first record shown above with a direction value of 999.9, and a speed value of 99.99. These record failures can occur for a variety of reasons, such as freezing (a frequent occurrence in these latitudes), failures in measuring equipment, intervals due to equipment maintenance, among others. Starting from this data file, the first task was the development of a post-processing code in Python to read and process all wind records, and subsequently processing the data to produce graphs that represent the wind regime on site. This algorithm used to read and process the wind data is listed in Appendix B.1.

3.4.2 Weibull distribution

A widely used tool in the wind industry to analyse the wind regime is the Weibull PDF. Like the Rayleigh distribution[1], is a probability distribution function that has been widely used in applications related to the wind energy industry because they are the probabilistic models that best describe the frequency distribution of wind speed. The Weibull probability distribution function is given by equation 3.1, and is characterised by 2 parameters: the dimensionless parameter k, which indicates the shape factor, and the parameter A, which indicates the speed scale factor[2].

$$p(U) = \left(\frac{k}{A}\right)\left(\frac{U}{A}\right)^{k-1} \exp\left[-\left(\frac{U}{A}\right)^{k}\right]. \tag{3.1}$$

Weibull curves representing the local wind regime were produced for the data sets corresponding to the three measurement heights, which are shown in Figure 3.6.

3.4.3 Wind data overall reading and processing results

Results obtained from the recorded wind data, processed through the algorithm developed in Python (Appendix B.1) for the three measurement heights (120, 82, and 48 m AGL), are presented in the Figure 3.6 for sectorwise frequency analysis, Figure 3.7 for sectorwise mean wind speed and energy contribution analysis, and Figure 3.8 for sectorwise average ambient turbulent intensity I_{amb} analysis and the dependence of the wind speed on it. All records were filtered and processed by sectors, with 30° intervals, thus dividing the wind rose into 12 main directions.

Overall results showed a predominant wind direction of 210° (SSW), both in terms of wind frequency and available energy in that same direction, whose wind power density $\mathcal{P}_w$ was calculated with the expression:

$$\mathcal{P}_w = \frac{1}{2}\rho U^3, \tag{3.2}$$

considering the air density corresponding to the average temperature over the measurement period of $\rho = 1.239$ kg/m^3. In addition, turbulent intensity recorded in measurements at 120 m AGL (height closest to the height of the turbines) is relatively low, with an average value of 10.01%, and 9.62% in the predominant wind direction of 210° (SSW), which is a good indicator for the wind farm. The maximum turbulence intensity recorded was 12.07% for winds coming from 0° (N), which is not relevant due to the fact that the frequency of winds from this direction is low, with only 6.72% of all occurrences. In terms of data validation, only records with wind speeds higher than 5 m/s were considered[3].

[1] The Reyleigh distribution is a special case of the Weibull distribution, where the shape parameter is $k = 2$.

[2] Various methods can be used to estimate the Weibull parameters, A and k, depending on the wind data available and the precision required in the analysis. Some of these methods are, for example, the method of moments, the least squares method, the average velocity method, and graphical methods

[3] Wind turbines generally begin generating effective power at a certain minimum wind speed, called the cut-in speed, which is typically in the range of 3 to 4 m/s. However, efficient and relevant turbine operation occurs at slightly higher speeds, generally above 5 m/s. Therefore, turbulent intensity studies for speeds below this threshold are less critical, as the turbines will not be operating significantly and any loads on the turbines at these speeds are very low.

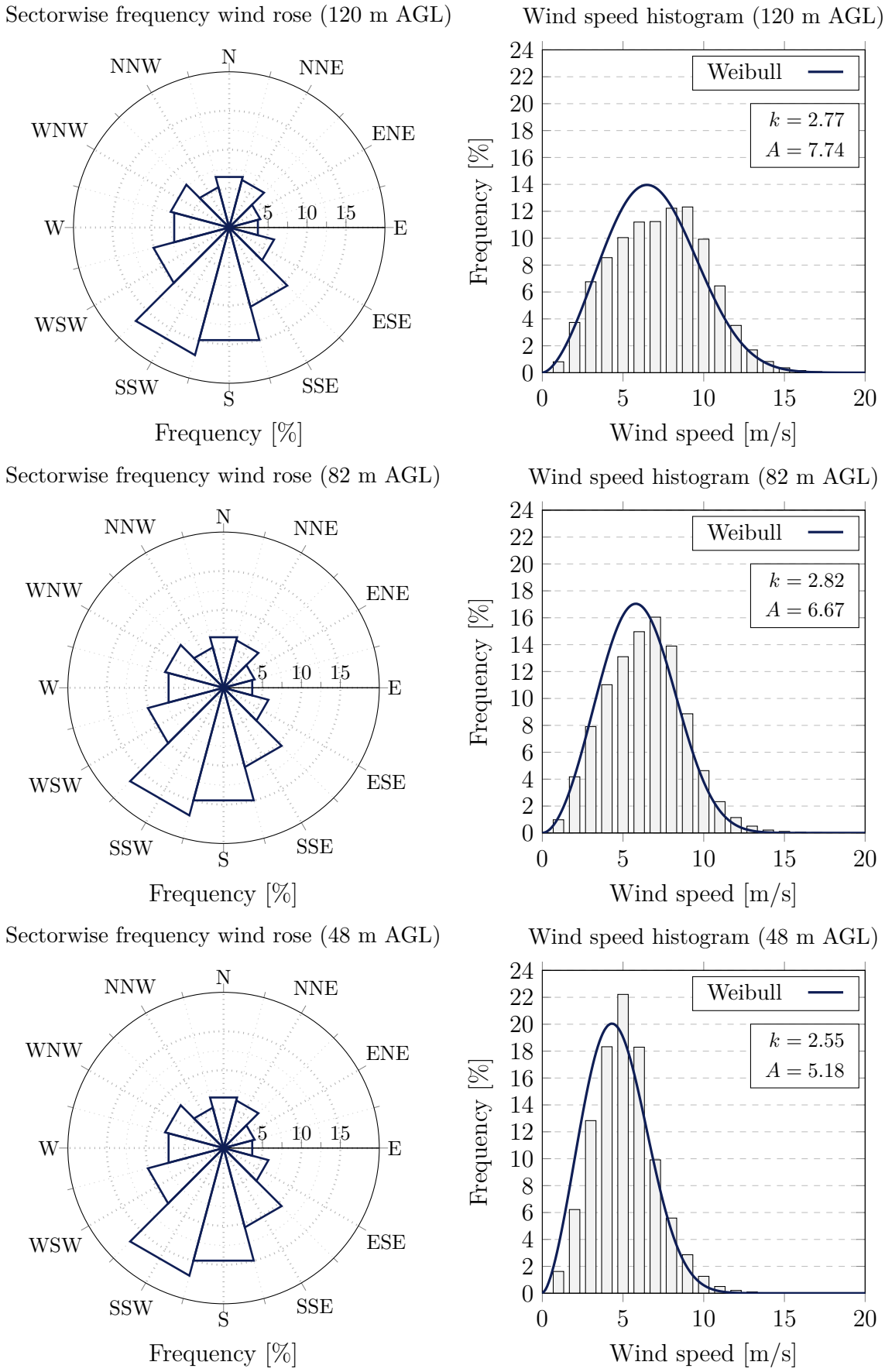
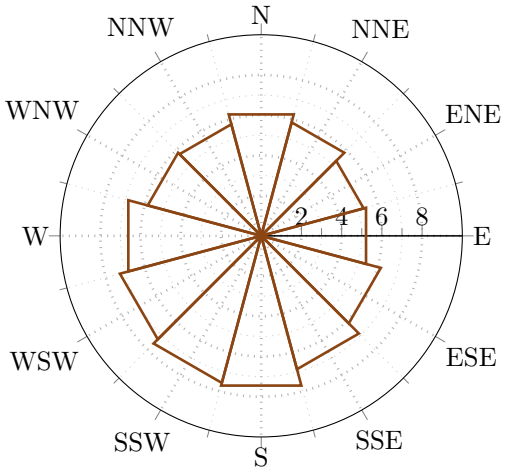


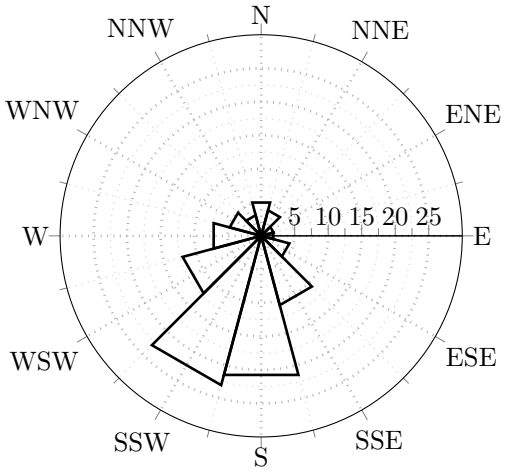
Figure 3.6: Results from wind data processing in terms of frequency of measurements and Weibull curves for heights of 120, 82 and 48 m AGL

Sectorwise mean wind speed (120 m AGL)



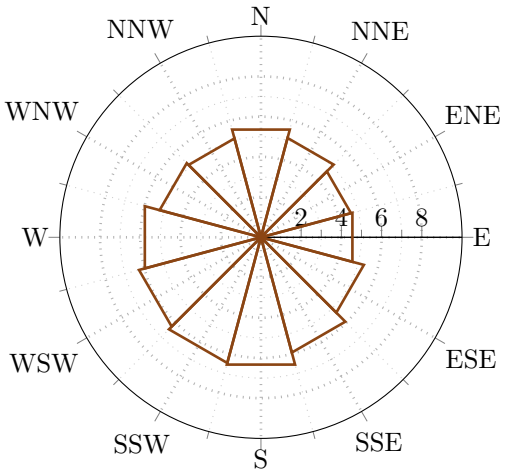
Wind speed [m/s]

Sectorwise energy contribution (120 m AGL)



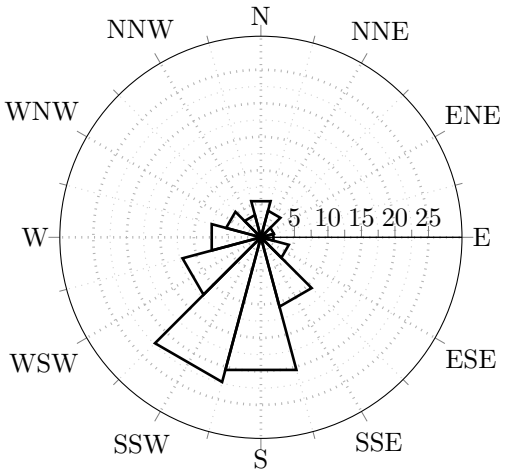
Energy contribution [%]

Sectorwise mean wind speed (82 m AGL)



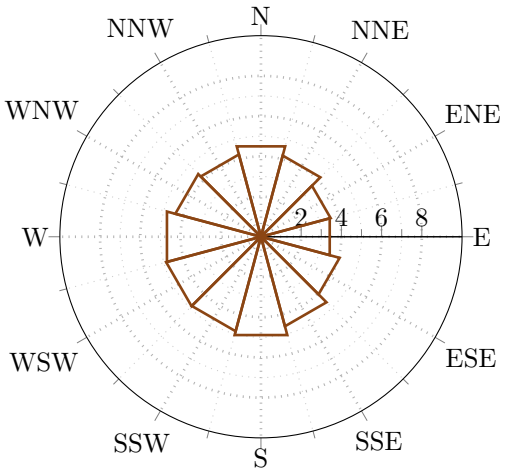
Wind speed [m/s]

Sectorwise energy contribution (82 m AGL)



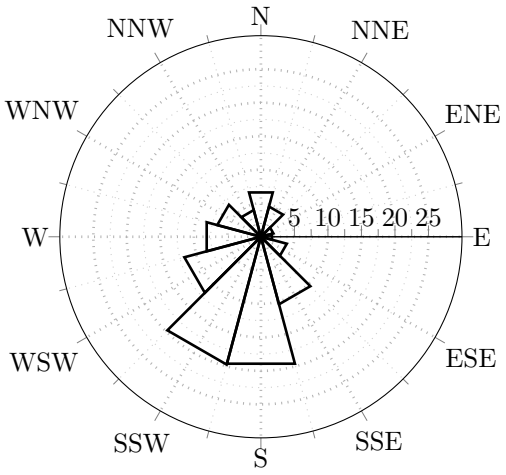
Energy contribution [%]

Sectorwise mean wind speed (48 m AGL)



Wind speed [m/s]

Sectorwise energy contribution (48 m AGL)



Energy contribution [%]

Figure 3.7: Results from wind data processing in terms of mean wind speed and energy contribution for heights of 120, 82 and 48 m AGL

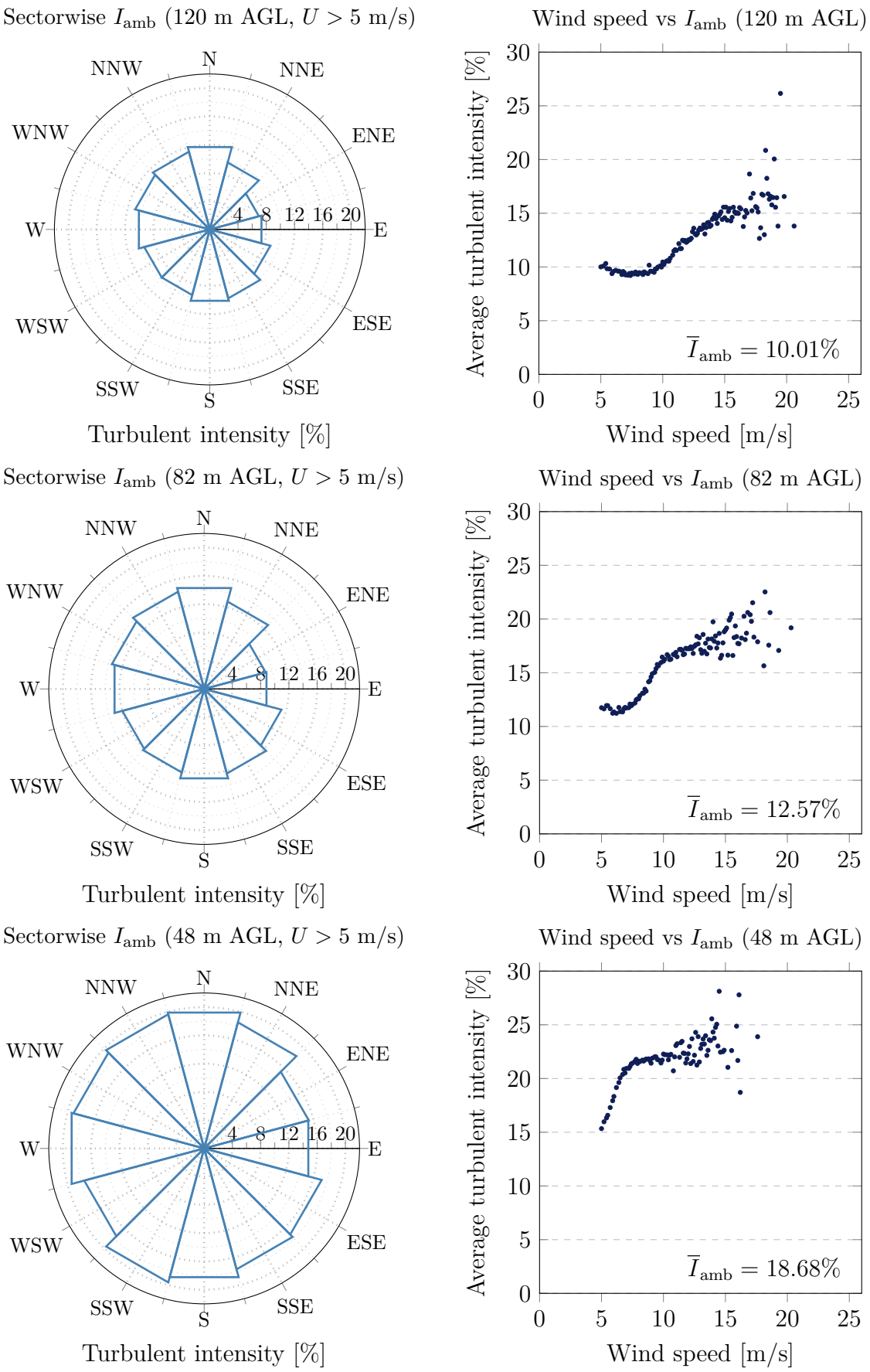


Figure 3.8: Results from wind data processing in terms of ambient turbulent intensity for heights of 120, 82 and 48 m AGL

All these results were synthesised for the three measurement heights in Table 3.2, including the Weibull parameters for each one of them.

Table 3.2: Overall results of reading and processing wind data, synthesised for the three measurement heights

z_{AGL} [m]	$\overline{U}$ [m/s]	U_{max} [m/s]	A_{Weibull} [m/s]	k_{Weibull}	$\overline{I}_{\mathrm{amb}}$ ($U > 5$ m/s) [%]	$\mathcal{P}_w$ [W/m^2]
120	6.74	20.57	7.74	2.77	10.01	297.92
82	5.84	20.28	6.67	2.82	12.57	189.08
48	4.59	17.58	5.18	2.55	18.68	93.75

Chapter 4

Simulations of neutrally stratified ABL flows using uniform and variable terrain roughness

4.1 Introduction

In this Chapter we present simulations of neutrally stratified ABL flows over the existing Mörknässkogen wind farm, using uniform terrain roughness and variable terrain roughness based on a roughness map. Both topographic and roughness maps were provided for the study developed in this work.

Initially three sets of simulations were produced, briefly described in Table 4.1, where terrain roughness was modelled (i) as uniform roughness, (ii) as variable roughness based on a roughness map, and (iii) was an attempt to duplicate the roughness value of all points on the roughness map to replicate the effect of the forested areas in the region. For each set, 12 simulations were performed corresponding to the wind direction, at intervals of 30° between 0° and 360° of the wind rose. Procedures described in this Chapter were used in the three sets. In Table 4.1, and ni refers to the number of control volumes in the i direction.

Table 4.1: Initial description of the sets of simulation performed with uniform and variable roughness based on a roughness map

Ref.	**Name**	$nx \times ny \times nz$	**Computational domain** [km^3]	**Roughness**
SET_{100}	Uniform	150×150×60	16.0×16.0×3.0	Uniform
SET_{101}	Roug	150×150×60	16.0×16.0×3.0	Roughness map
SET_{102}	Roug×2	150×150×60	16.0×16.0×3.0	2×Roughness map

In terms of general configurations, all simulations were defined with a numerical mesh of 150×150×60 control volumes, associated with a computational domain of 16000×16000×3000 m^3 in the x, y and z directions respectively. The remaining settings are described in greater detail later in this Section.

All tests and simulations were conducted in OpenFOAM v2306[1] under a Linux distribution within the INCD cloud environment, that was used without any source code modifications. All post-processing codes ran on a regular laptop with an Intel Core i5-7200 2.70 CPU, 8 GB of RAM and a 256 GB SSD disk.

4.2 Model limitations

All simulations produced in this Chapter were performed having on its basis some limitations in terms of the applied mathematical model, whose assumptions and simplifications were as follows:

- **Absence of moisture in air**

  Atmospheric air was considered to be dry, a common simplification adopted in most ABL flows according to reviewed literature, which reduces computational costs as a result of simplifying the mathematical model.

- **Incompressible and Newtonian fluid**

  Air was modelled assuming incompressibility and Newtonian behavior[2], which also simplifies the governing equations, thus reducing computational resources. This approach is particularly effective for steady-state ABL flows where density variations are minimal and viscosity remains constant. These assumptions are established by the *simpleFoam* solver that was used in all simulations.

- **Absence of Coriolis force**

  OpenFOAM's *simpleFoam* solver does not account for Coriolis force or any other rotation effects. For simulations where Coriolis force is significant, such as in large-scale meteorological or oceanographic models, its effect can be incorporated using additional terms in the momentum equations via the `fvOptions` file or using a different solver.

- **Absence of buoyancy and gravitational forces**

  All simulations were performed considering steady-state and neutrally stratified ABL flows, without solving an additional energy equation.

- **Absence of forest model**

  In this Chapter, surface roughness modelling is confined to the aerodynamic roughness length z_0 incorporated into the *ground* patch wall function. However, a canopy model was introduced in the simulations performed in Chapter 5.

[1] OpenFOAM v2306 is distributed by OpenCFD Ltd, that has been managing and developing OpenFOAM since its debut in 2004, releasing all versions prior to 8th August 2011, when OpenCFD transferred the IP rights to "OpenFOAM Foundation, inc".

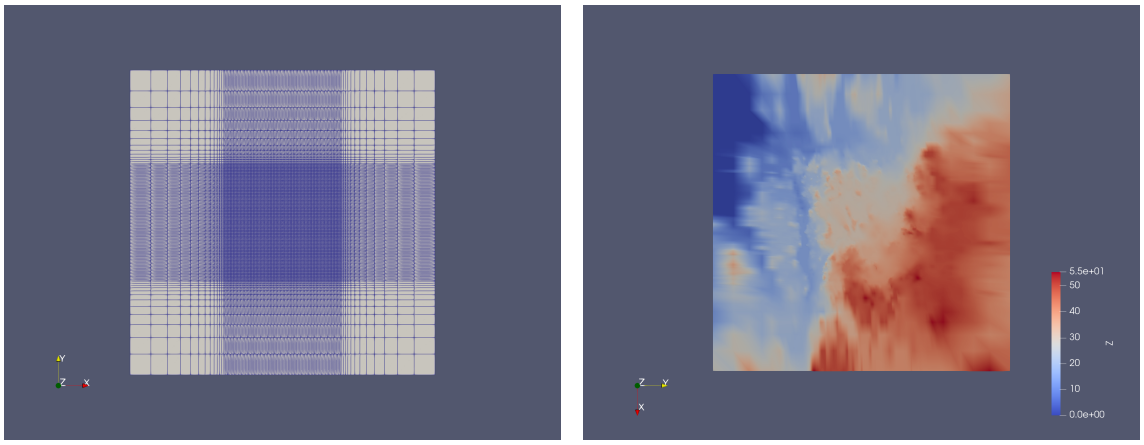
[2] Air is classified as a Newtonian fluid, meaning that its viscosity remains constant regardless of the rate of deformation or shear stress applied. In a Newtonian fluid, shear stress is directly proportional to the velocity gradient, following Newton's law of viscosity. For atmospheric air, this physical relation between shear stress τ and the rate of strain du/dy is linear and governed by $\tau = \mu(du/dy)$ being μ the constant dynamic viscosity.

4.3 Preprocessing

4.3.1 Horizontal 2D mesh generation

As briefly mentioned in Section 2.4.3, the horizontal numerical mesh was generated using the *gsrf3* and *groug* codes. In terms of procedure, these codes start by manipulating the topography and roughness maps (`topo.dat` and `roug.dat`) located in the `_gsurf_WINDIE/topography` directory (see case structure in Figure 2.9) to convert this data into an orthogonal mesh, defined for a horizontal computational domain of 16000×16000 m^2, with 150×150 number of control volumes in the x and y directions respectively.

Besides that, *gsrf3* also has an additional tool to set a higher resolution area within a defined central area, which has been set to a 6000×6000 m^2 square, with a central coordinates distance of 50 m in both x and y directions[3]. Outside this area, mesh resolution gradually decreases as a function of a geometric progression until it reaches the edges of the computational domain. The resulting generated mesh is presented in Figure 4.1, including the VTK file `topo_LOCAL.vtk` representing the numerical mesh generated by *gsrf3*, which can be visualised in *Paraview* software.



(a) (b)

Figure 4.1: Computational domain generated by *gsrf3* for a 150×150 horizontal mesh, with a central area of higher resolution. (a) Top view of the mesh. (b) Top view of the VTK file `topo_LOCAL.vtk` generated by *gsrf3*

Lastly, *gsrf3* code apply a rotation to all mesh points around its vertical axis, centered on the defined UTM34 coordinates (574986.9 N, 7018340.4 E), to align the *inlet* patch to be perpendicular to the selected wind direction. In the OpenFOAM simulations we opted for the flow input to always enter from the left side of the computational domain, so in addition to the rotation that is applied depending on the wind direction, another additional rotation of 90° counterclockwise is also applied to align the computational domain with the flow inlet direction.

Base settings for mesh generation are defined in the `preproc.cfg` and `windie.cfg` files found in the `_gsurf_WINDIE` directory, the most important of

[3]This central area of higher resolution was defined to contain the turbines and the mast, which are the points of greatest interest for the study.

which are described in greater detail in Table 4.2 and Table 4.3.

Table 4.2: Main settings for generating the horizontal numerical mesh, defined in the configuration file `preproc.cfg`

Parameter	Description	Value
`nit, njt`	Number of control volumes in x and y directions of the topography map.	`nit` = 2640 `njt` = 2759
`nir, njr`	Number of control volumes in x and y directions of the roughness map.	`nir` = 2640 `njr` = 2759
`dxn, dyn`	Length of the central area of higher resolution, in x and y directions [m].	`dxn` = 6000.0 `dyn` = 6000.0
`cofx, cofy`	Mesh resolution in the central area of higher resolution, in x and y directions [m].	`cofx` = 50 `cofy` = 50
`xmax, xmin`	Length of the computational domain in x direction relative to the center [m].	`xmax` = 8000.0 `xmin` = −8000.0
`ymax, ymin`	Length of the computational domain in y direction relative to the center [m].	`ymax` = 8000.0 `ymin` = −8000.0
`rot`	Wind direction [°].	`rot` = 0.0
`xcenter, ycenter`	Geographic UTM34 coordinates of the mesh center.	`xcenter` = 574986.9 N `ycenter` = 7018340.4 E
`npassfilter`	Terrain smoothing factor at the edges	`npassfilter` = 0.0
`alphalayer`	Thickness at the edges where the `npassfilter` smoothing is applied	`alphalayer` = 0.0

Table 4.3: Main settings for generating the horizontal numerical mesh, defined in the configuration file `windie.cfg`

Parameter	Description	Value
`nig, njg`	Number of control volumes of the numerical mesh, in the x and y directions	`nig` = 150 `njg` = 150

4.3.2 Vertical 3D mesh generation

The vertical numerical mesh is generated using the *write_blockMeshDict* code, which is part of the tools developed in [54], previously mentioned in Section 2.4.3, that reads the VTK file created by *gsrf3* and write its xyz data in the OpenFOAM `blockMeshDict` format. With this procedure, when OpenFOAM *blockMesh* utility is executed, each face from the VTK surface would originate a hexahedron with bottom face shaped to ground surface z data and a horizontal top face at the defined height of 3000 m. These hexahedrons were also configured to be divided into 60

control volumes in the z direction, and their length is gradually increased to a ratio $R = 100.0$ between the height of the control volumes furthest from and closest to the ground.

Generated mesh, presented in Figure 4.2, was also validated with the OpenFOAM *checkMesh* utility which returned no errors or warnings.

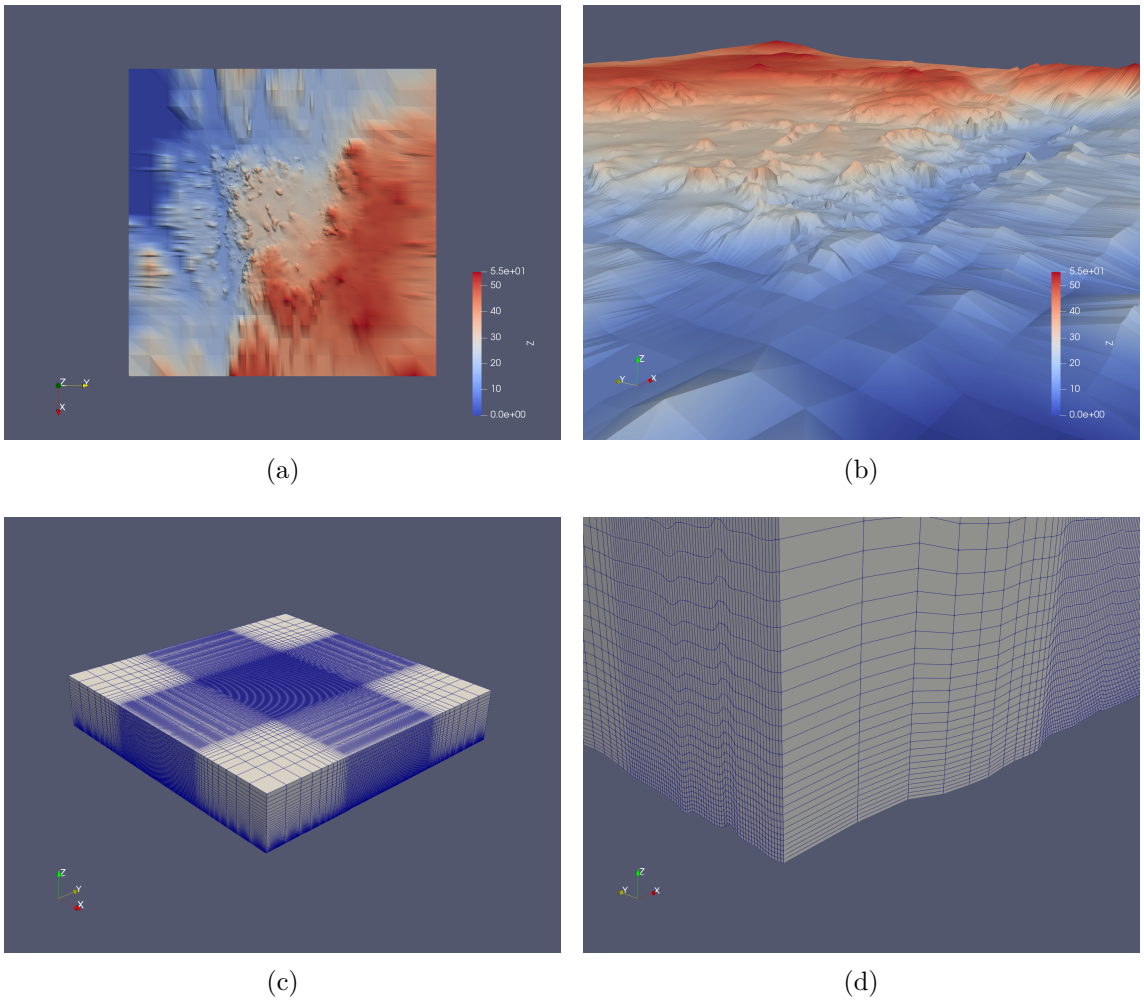


Figure 4.2: Computational domain of Mörknässkogen wind farm for a 150 × 150 × 60 numerical mesh, with a central area of higher resolution. (a) Top view of *ground* patch. (b) Zoomed isometric view of *ground* patch. (c) Isometric domain overview. (d) Cross section plane normal to y axis at domain center. In all figures except (c), *ground* patch is magnified 20 times in z direction to make the topographic variation visible, as the real terrain is relatively flat.

Base settings for vertical mesh generation are defined in the `_preprocDict` file found in the `_preproc` directory, here described in greater detail in Table 4.4.

Domain boundaries were also defined when generating the `blockMeshDict` mesh file, in which the domain was divided into 6 patches: the terrain was given the designation *ground*, to which was assigned the OpenFOAM attribute of `wall` type; *sky*, *front* and *back* boundaries were assigned the `symmetryPlane` type; and for the *inlet* and *outlet* boundaries the generic type `patch` was defined. For nodes in the

Table 4.4: Main settings for generating the vertical numerical mesh, defined in the configuration file `_preprocDict`

Parameter	Description	Value
`sky`	Calculation domain height [m].	`sky` = 3000.0
`nz`	Number of control volumes in z direction.	`nz` = 60
`R`	Geometric expansion ratio in z direction.	`R` = 100.0

ground patch, z coordinates were assigned from the VTK file data and for nodes in the *sky* patch, z coordinates were set to the defined domain height of 3000.0 m.

4.3.3 Boundary conditions

In this Section, boundary conditions were implemented considering that all simulations performed referred to idealised incompressible and neutrally stratified ABL flows, in steady state.

Given this assumption, the quantities that have been configured in the OpenFOAM folder `0` are restricted to: pressure p, velocity U, turbulent kinetic energy k, turbulent kinematic viscosity ν_t, and turbulent kinetic energy dissipation rate ε, since the $k-\varepsilon$ turbulence model[4] was used.

Since turbulent viscosity directly depends on the surface aerodynamic roughness length z_0, the configuration of ν_t boundary conditions will be described in Section 4.3.4. Among the remaining quantities, the pressure p is the only one that can be considered to have static conditions, which means that its boundary conditions can be defined independently of site data or any other quantities involved. Table 4.9 displays the boundary conditions assigned to the pressure field in the p file for each of the patches into which the domain was divided.

Table 4.5: Boundary conditions for pressure field set up in `p` file

Patch/Field	Quantity	Attribute	Value	First guess
ground, *inlet*	p	`zeroGradient`	-	-
sky, *front*, *back*		`symmetryPlane`	-	-
outlet		`fixedValue`	`uniform 0`	-
internal		-	-	`uniform 0`

For the quantities U, k and ε, whose boundary conditions depend on the studied site data, respective files `U`, `k` and `epsilon` were written in OpenFOAM folder `0` using the tool *write_BCs* developed in [54], based on user defined parameters. These base settings required to calculate the boundary conditions, and similar to the vertical mesh generation, are also defined in the `_preprocDict` file found in the `_preproc` directory, and the respective parameters are listed in Table 4.6. According to the

[4]For the $k-\omega$ turbulence model, the dissipation frequency of turbulent kinetic energy ω must also be included in the OpenFOAM folder `0`.

assumptions of this procedure, the influence of site data on domain boundaries is confined to the *inlet* and *ground* patches.

Table 4.6: Main settings for boundary conditions calculations, defined in the configuration file `_preprocDict`

Parameter	**Description**	**Value**
`zrs`	Reference height AGL [m].	`zrs` = 10.0
`ustar`	Friction velocity u_* [m/s].	`ustar` = 0.607
`z0i`	Aerodynamic roughness length z_0 [m].	`z0i` = 0.03
`delta`	Boundary layer height δ [m].	`delta` = 1000.0
`kv`	Von Kárman constant κ.	`kv` = 0.41
`cmu`	$k-\varepsilon$ turbulence model coefficient C_μ.	`cmu` = 0.033

For *sky*, *front* and *back* patches were assigned the `symmetryPlane` attribute to adhere to the no-flux condition, which is physically implemented by configuring it in the `U` file, making velocity tangential to these boundaries.

Following the recommendations indicated in [45], boundary conditions for *outlet* patch implied a zero gradient condition for the scalars k and ε, and also a mass conservation condition for U, which in OpenFOAM can be configured with the `inletOutlet` attribute.

Inlet boundary conditions for quantities U, k and ε are defined according to the inlet profile equations proposed by [55], given by:

$$U = \begin{cases} u_*/\kappa \ln(1 + z/z_0) & , z < \delta \\ u_*/\kappa \ln(1 + \delta/z_0) & , z \geq \delta \end{cases}, \tag{4.1}$$

$$k = \begin{cases} u_*^2/C_\mu^{1/2} \ln(1 - z/\delta) & , z < 0,99\delta \\ u_*^2/(100 C_\mu^{1/2}) & , z \geq 0,99\delta \end{cases}, \tag{4.2}$$

$$\varepsilon = \begin{cases} C_\mu^{3/4} k^{3/2}/(\kappa z) & , z < 0,95\delta \\ C_\mu^{3/4} k^{3/2}/(0,95\delta\kappa) & , z \geq 0,95\delta \end{cases}. \tag{4.3}$$

These profiles were calculated with $u_* = 0.607$ m/s, $C_\mu = 0.033$, $z_0 = 0.03$ m and $\kappa = 0.41$, which are typical calibrated values used in regular neutrally stratified ABL flows under similar conditions [55] and a prescribed ABL height $\delta = 1000.0$ m[5].

At *ground* patch, U was set to zero to establish a no-slip condition. To reduce the computational effort and resources that would be required to integrate the turbulence model equations up through the entire wall, OpenFOAM wall functions were used for both turbulent quantities k and ε, as follows:

[5]Friction velocity u_* can be calibrated for each wind direction based on wind direction we opted not to do this to better assess the use of the canopy model later.

- k

  For turbulent kinetic energy, the OpenFOAM `kqRWallFunction` was used, which applies a zero gradient boundary condition for k at the wall, meaning that k remains constant in the near-wall region, which simplifies the modelling of the near-wall turbulence. This assumption has also been widely found in the reviewed literature on the simulation of ABL flows over complex terrain. Its value was calculated with the equation proposed by [56]:

$$k = \frac{u_*^2}{C_\mu^{1/2}}, \tag{4.4}$$

  where the value $k_{\text{wall}} = 2.028$ was obtained from the previously defined coefficients of $u_* = 0.607$ m/s and $C_\mu = 0.033$.

- ε

  For turbulent kinetic energy dissipation rate, the OpenFOAM wall function used was `epsilonWallFunction`, that follows the standard near-wall treatment described in [46]. In this approach, ε is calculated with the expression:

$$\varepsilon = \frac{u_*^3}{\kappa z_p}, \tag{4.5}$$

  where z_p is the distance between the ground face and the center of the adjacent cell. According to the defined constants $u_* = 0.607$ m/s, $\kappa = 0.41$ and $z_p = 0.03$ m (described in Section 4.3.4), the value of $\varepsilon_{\text{wall}} = 18.183$ was obtained[6].

According to this formulation, coded boundary conditions in *write_bCs* are synthesised in Table 4.7 for all six patches from the computational domain.

4.3.4 Roughness modelling

Terrain aerodynamic roughness length z_0 was modelled using the tool *write_z0* developed in [54], which allows mapping of the roughness values in two ways: (i) with uniform roughness, assigning the predefined value of z_0 to all mesh points; (ii) with roughness defined from a variable roughness map.

In the simulations performed in this Chapter, previously described in Table 4.1, uniform roughness was used for SET_{100}, and variable roughness based on a roughness map was used for SET_{101} and SET_{102}. For SET_{102}, roughness values were mathematically manipulated to double their default values at all mesh points. This selection of the roughness modelling type and the aerodynamic roughness length z_0 are both defined in the configuration file `_preprocDict`. These configurations are described in Table 4.8.

[6]However, bibliographic research revealed that the most used formulation for the ε wall function in the simulations of ABL flows over complex terrain, that also considers aerodynamic roughness length z_0, proposes the expression $\varepsilon = u_*^3/[\kappa(z + z_0)]$ [52, 56], although it was not used in this work because it does not follow the native formulation used in OpenFOAM

Table 4.7: Boundary conditions for U, k and ε coded in *write_bCs*

Patch/Field	Quantity	Attribute	Value
U			
ground	`fixedValue`	`uniform (0 0 0)`	-
sky, front, back	`symmetryPlane`	-	-
inlet	`fixedValue`	Equation 4.1	-
outlet	`inletOutlet`	`$internalField`	-
internal	-	-	`uniform 0`
k			
ground	`kqRWallFunction`	`uniform 2.028`	Equation 4.4
sky, front, back	`symmetryPlane`	-	-
inlet	`fixedValue`	Equation 4.2	-
outlet	`zeroGradient`	-	-
internal	-	-	Equation 4.4
ε			
ground	`epsilonWallFunction`	`uniform 18.183`	Equation 4.5
sky, front, back	`symmetryPlane`	-	-
inlet	`fixedValue`	Equation 4.3	-
outlet	`zeroGradient`	-	-
internal	-	-	Equation 4.5

Table 4.8: Main settings for aerodynamic roughness length modelling, defined in the configuration file `_preprocDict`

Parameter	Description	Value
`z0u`	Aerodynamic roughness length z_0 to be considered in case of choosing the Uniform roughness mode [m].	`z0u` $= 0.03$
`opt`	Intended roughness mapping mode: #1 - Uniform roughness #2 - Roughness map in `rough_LOCAL.vtk` #3 - Defined by equation in `write_z0.f90`	`opt` $= 1$ (SET_{100}) `opt` $= 2$ (SET_{101}, SET_{102})

In OpenFOAM, the ground surface terrain roughness is configured in the `nut` file inside folder `0` as boundary condition for turbulent kinematic viscosity ν_t. In the original version of the *write_z0* code developed in [54], this boundary condition is introduced using the `nutkAtmRoughWallFunction` wall function programmed directly in the source code, which has been discontinued in the latest versions of OpenFOAM. To overcome this, *write_z0* code was modified to replace this boundary conditions with its latest version, called `atmNutkWallFunction`, that provides a wall constraint on the turbulent viscosity ν_{t_w} calculated with the following expression:

$$\nu_{t_w} = \nu_w \left[\frac{y^+ \kappa}{\ln \left[\max \left(E', 1 + 10^{-4} \right) \right]} - 1 \right], \tag{4.6}$$

where ν_w is the kinematic viscosity of fluid near wall, $y^+ = (u_* y)/\nu_w$ the wall-normal distance in wall units, $u_* = C_\mu^{1/4}\sqrt{k}$ and $E' = (y + z_0)/z_0$ a distance parameter.

According to this slightly different formulation, boundary conditions coded in *write_z0* are described in Table 4.9 for all six patches from the computational domain, and parameters C_μ, κ, E' and z_0 are defined for *ground* patch in file `nut` that is written by *write_z0*.

Table 4.9: Boundary conditions for ν_t coded in *write_z0*

Patch/Field	Quantity	Attribute	Value	First guess
ground	ν_t	`atmNutkWallFunction`	-	`uniform 0`
sky, *front*, *back*		`symmetryPlane`	-	-
inlet, *outlet*		`calculated`	-	`uniform 0`
internal		-	-	`uniform 0`

4.3.5 Turbulence properties

The turbulence model was parameterised using the tool *write_turbulenceProperties* developed in [54], which was designed to automate the process of selecting the turbulence model to be used by the OpenFOAM solver, which is written in the file `turbulenceProperties` in the `constant` folder.

All simulations performed in this Chapter were defined with RaNS atmospheric $k - \varepsilon$ turbulence model, with the empirical constants of the model fitted for ABL flows (see Table 2.5). This selection is defined in the configuration file `_preprocDict`, whose respective `model` parameter is described in Table 4.10.

Table 4.10: Settings for turbulence model defined in `_preprocDict` file.

Parameter	Description	Value
`model`	Intended turbulence model: # kEpsilon: 11-Calibrated, 12-Standard, 13-Atmospheric # RNGkEpsilon: 21-Original, 22-Modified, 23-Atmospheric # RealizableKE: 3 # kOmega: 4 # kOmegaSST: 5	`model` = 13

4.4 Processing

All simulations were performed with the *simpleFoam* solver, typically recommended for this type of incompressible turbulent and steady-state ABL flows over complex terrain. In this work, convergence criteria for initial residuals were defined in the `fvSolution` file in `system` folder, with values of 1×10^{-3} for pressure field p, and 1×10^{-4} for the other quantities U, k and ε. These criteria are slightly stricter than those found in OpenFOAM tutorials, with the purpose of increasing the accuracy of results.

In terms of linear solvers, two distinct methods were implemented: GAMG (Generalised Geometric-Algebraic Multi-Grid) for the pressure field and PBiCG (Preconditioned Bi-Conjugate Gradient) for the remaining fields. Both methods were configured with a tolerance between iterations equal to 1×10^{-5}, except for the velocity field U. Although at the beginning of the study this parameter was also set to 1×10^{-5}, it was later identified that this value was causing the simulations not to converge. Tests have demonstrated that small variations in the velocity field U, even of the order of 10^{-5}, produced significant changes in the pressure field p, causing its initial residuals to remain permanently in oscillation but never reaching convergence. To overcome this problem, tolerance between iterations for velocity field U was changed to 1×10^{-12}, having completely eliminated the convergence problems from this origin.

4.4.1 Developed procedure to launch OpenFOAM simulations

To optimise the process of launching simulations, a script called `Run.sh` was created, which sequentially executes all terminal commands that are necessary to perform the simulation, making the process more agile. Thus, it is only necessary to set all configuration files previously, and then run the script via terminal, which will process every operation automatically. The code is presented in Figure 4.3.

1	`foamCleanPolyMesh`	Remove resultados anteriores.
2	`cd _preproc/_gsurf_WINDIE/`	-
3	`./gsrf3`	Geração da malha topográfica 2D.
4	`./groug`	Geração da malha de rugosidade 2D.
5	`cd ../`	-
6	`./write_blockMeshDict`	Escreve a estrutura da malha.
7	`cd ../`	-
8	`rm -rf log.blockMesh`	-
9	`blockMesh`	Geração da malha vertical 3D.
10	`cd _preproc/`	-
11	`./write_z0`	Escreve os ficheiros de rugosidade.
12	`./write_bCs`	Escreve os ficheiros das condições de fronteira.
13	`./write_turbulenceProperties`	Escreve os ficheiros das propriedades de turbulência.
14	`cd ../`	-
15	`simpleFoam | tee log.simpleFoam`	Executa o solver *simpleFoam*.
16	`cp -r $(foamListTimes -latestTime) res_final`	Copia a última simulação para a pasta `res_final`.
17	`touch visu.foam`	Cria o ficheiro .foam para o Paraview.

(a) (b)

Figure 4.3: `Run.sh` script to launch simulations. (a) Code content of `Run.sh` script. (b) Short description of code output.

4.5 Postprocessing

4.5.1 The *Synthesis* algorithm

The wind resource assessment was evaluated with a new tool coded in Python called the *Synthesis* algorithm, by which the field measurements at the mast inside the computational domain are incorporated into the numerical solutions. This code is capable of reading the results from OpenFOAM simulations and then apply a transfer matrix to transport the time series measured on the mast to any position, which in this case are the turbines.

The procedure thus allows establishing a relationship between measured velocity U and turbulence intensity I to any point inside the grid, to estimate the wind farm's energy production and to evaluate harmful parameters such as turbulent intensity, wind shear α and flow inclination γ.

In addition, the algorithm can also be used to evaluate the mathematical model implemented through cross-predictions between different heights on the mast itself or between masts when available. These results are then compared with measured wind data, and the set that presents the smallest deviation between measured and calculated results is selected as the one that best represents the wind regime.

In terms of the *Synthesis* algorithm, and starting from a given set of 12 simulation, the code procedure can be mathematically summarised by the following processing routine for each simulated direction:

(i) Read OpenFOAM results using the Python's *fluidfoam* library, consisting of the xyz data of the three-dimensional numerical mesh to obtain all coordinates and the calculated quantities of U and k at all mesh points.

(ii) Perform a three-dimensional scan, and for each control volume, inverts the rotation applied to the mesh by *gsrf3*, transform the relative coordinates into UTM34 coordinates by adding the geographic coordinates of the center of the mesh, and calculate the velocity magnitude from all U components, wind direction, shear factor α, flow inclination γ and turbulent intensity I.

(iii) Perform a three-dimensional linear interpolation of the velocity U and turbulent intensity I to the coordinates of the four turbines.

(iv) Read the wind data records for the three mast measurement levels (120, 82 and 48 m AGL) to construct the respective time series.

(v) Transport the measurement time series to the coordinates of the four turbines using the following transport equations, one for each transported quantity. Velocity values are synthesised according to:

$$\widehat{U}_h^t = U_h^{\oplus} \underbrace{\left(\widetilde{U}_h^t / \widetilde{U}_h^{\oplus}\right)}_{\Delta S}, \tag{4.7}$$

where $\Delta S = \widetilde{U}_h^t / \widetilde{U}_h^{\oplus}$ is the non-dimensional speed-up ratio, the symbol "$\widehat{\ }$" is the synthesised value, "$\widetilde{\ }$" is the calculated CFD value obtained through three-dimensional interpolation, superscript "$^{\oplus}$" refers to the reference station (mast), superscript "^{t}" refers to target point where the synthesis is being performed, and $U_h^{\oplus}$ is the measured horizontal velocity collected at the mast. For mean turbulent intensity, values are synthesised according to:

$$\widehat{I}^t = \widetilde{I}^t + \Delta I \left(\widetilde{U}_h^{\oplus} / \widetilde{U}_h^t\right), \tag{4.8}$$

where $\Delta I = I^{\oplus} - \widetilde{I}^{\oplus}$ is the difference between the measured and calculated turbulence intensity I, which when multiplied by the inverse of the speed-up ratio ΔS results in the turbulence correction term.

4.6 Presentation and discussion of results

This Section presents the simulation results and conclusions obtained from the three different strategies used to model terrain roughness. All sets were evaluated in terms of the mathematical model validation and vertical profiles. Results for horizontal velocity U_h, turbulent kinetic energy k and turbulent intensity I were evaluated in terms of their non-dimensional ratios defined for each quantity by:

$$U_h^* = \frac{U_h}{U_{h,\text{ref}}}, \tag{4.9}$$

$$k^* = \frac{k}{k_{\text{ref}}}, \tag{4.10}$$

$$I^* = \frac{I}{I_{\text{ref}}}, \tag{4.11}$$

where $U_{h,\text{ref}}$, k_{ref} and I_{ref} are the reference values calculated or measured at the intermediate measuring height of the mast, corresponding to 82 m AGL.

Figure 4.4 shows the vertical profiles obtained for a 210° wind from OpenFOAM simulations compared with the data measured at mast coordinates, in terms of non-dimensional velocity U_h^* and non-dimensional turbulent kinetic energy k^*.

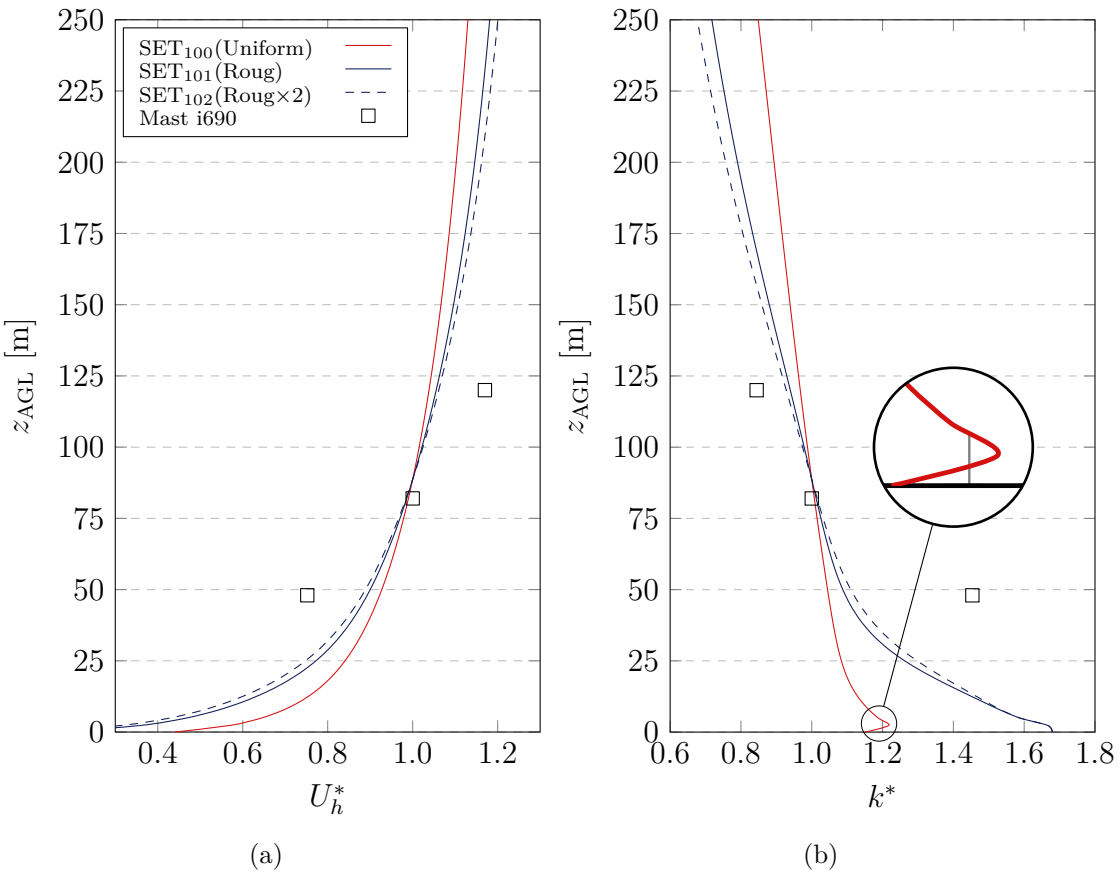


Figure 4.4: Vertical profiles at mast i690 coordinates for a 210° wind to analyse the influence of the terrain roughness modelling type. (a) Non-dimensional horizontal velocity U_h^*. (b) Non-dimensional TKE k^*.

Graphical analysis has demonstrated that SET_{101}(Roug) and SET_{102}(Roug×2), whose terrain roughness is modelled based on a roughness map, presents a curve that better fits the measured data. Non-dimensional turbulent intensity I^* vertical profiles illustrated in Figure 4.5 is also consistent with this conclusion, as the profiles present a significant improvement in terms of approximation to measured data at both 48 and 120 m AGL, suggesting a more realistic representation of wind conditions across the region.

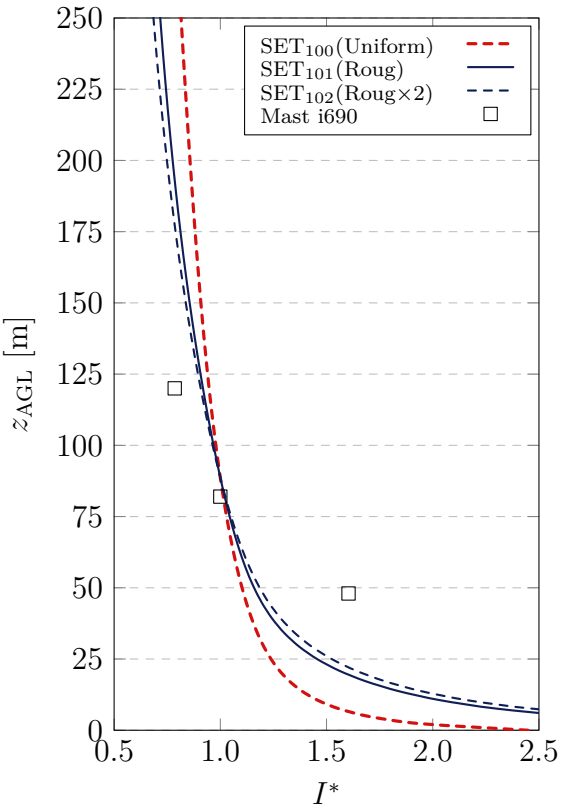


Figure 4.5: Non-dimensional turbulent intensity I^* vertical profile at mast i690 coordinates for a 210° wind to analyse the influence of roughness modelling type.

4.6.1 Mathematical model validation

The *Synthesis* algorithm was applied to the three sets of simulations with the strategy of performing wind cross-predictions between different measurement heights on the mast itself. These predictions consisted of transporting virtual time series for velocity U and turbulent intensity I (considering only wind speeds higher than 5 m/s) from one initial measurement height to the other two (for example from 120 to 82 m AGL, and from 120 to 48 m AGL), and the procedure is repeated for the other two measurement heights following the same strategy.

These results were compared with wind data measured at the same target position to analyse the deviations between the actual measured series and the transported series. The results were summarised in Table 4.11, Table 4.12 and Table 4.13, whose results indicate that the set that best represents the local wind regime is SET_{102}(Roug×2), with overall errors with an RMS error of 13.06% for velocity U and 3.21% for turbulent intensity I.

Table 4.11: Numerical model validation with *Synthesis* algorithm for simulations SET_{100}(Uniform)

From height [m]	**To height** [m]	U_{mast} [m/s]	U_{calc} [m/s]	ϵ [%]	I_{mast} [%]	I_{calc} [%]	ϵ [%]
Mast 120	Mast 82	5.838	6.446	10.41	12.575	10.639	-1.94
Mast 120	Mast 48	4.588	6.040	31.65	18.683	11.550	-7.13
Mast 82	Mast 120	6.741	6.105	-9.43	10.010	11.861	1.85
Mast 82	Mast 48	4.588	5.470	19.22	18.683	13.615	-5.07
Mast 48	Mast 120	6.741	5.120	-24.05	10.010	16.404	6.39
Mast 48	Mast 48	5.838	4.896	-16.14	12.575	17.324	4.75
RMS		-	-	20.03	-	-	4.95
BIAS		-	-	1.94	-	-	-0.19
STD		-	-	21.84	-	-	5.42

Table 4.12: Mathematical model validation with *Synthesis* algorithm for simulations SET_{101}(Roug)

From height [m]	**To height** [m]	U_{mast} [m/s]	U_{calc} [m/s]	ϵ [%]	I_{mast} [%]	I_{calc} [%]	ϵ [%]
Mast 120	Mast 82	5.838	6.263	7.28	12.575	11.434	-1.14
Mast 120	Mast 48	4.588	5.597	21.99	18.683	13.593	-5.09
Mast 82	Mast 120	6.741	6.283	-6.79	10.010	11.071	1.06
Mast 82	Mast 48	4.588	5.217	13.71	18.683	14.869	-3.81
Mast 48	Mast 120	6.741	5.526	-18.02	10.010	14.252	4.24
Mast 48	Mast 48	5.838	5.134	-12.06	12.575	15.983	3.41
RMS		-	-	14.38	-	-	3.48
BIAS		-	-	1.02	-	-	-0.22
STD		-	-	15.72	-	-	3.80

Table 4.13: Mathematical model validation with *Synthesis* algorithm for simulations SET_{102}(Roug×2)

From height [m]	**To height** [m]	U_{mast} [m/s]	U_{calc} [m/s]	ϵ [%]	I_{mast} [%]	I_{calc} [%]	ϵ [%]
Mast 120	Mast 82	5.838	6.215	6.46	12.575	11.608	-0.97
Mast 120	Mast 48	4.588	5.496	19.79	18.683	13.983	-4.70
Mast 82	Mast 120	6.741	6.332	-6.07	10.010	10.903	0.89
Mast 82	Mast 48	4.588	5.162	12.51	18.683	15.077	-3.61
Mast 48	Mast 120	6.741	5.628	-16.51	10.010	13.865	3.86
Mast 48	Mast 48	5.838	5.189	-11.12	12.575	15.763	3.19
RMS		-	-	13.06	-	-	3.21
BIAS		-	-	0.84	-	-	-0.22
STD		-	-	14.27	-	-	3.51

In this study, and taking into account the topographic profile around the Morknaskogen wind farm, it is highly likely that the cause of these deviations is due to the absence of a forest model implemented in the OpenFOAM simulations. This would explain the high deviations mainly at the height of 48 m AGL which is closest to the forest canopy, where the vertical wind profile is strongly modified. In other words, although SET_{102}(Roug×2) produced significantly better results when compared to the other two sets, the resulting deviations are considerably high for this type of study, where deviation values of less than 1% are typically considered acceptable for both quantities. It was then concluded that the attempt to duplicate all values in the roughness map proved to be insufficient in terms of approximating the results to the real behaviour of the flow, especially close to the ground, and therefore it is concluded that it is not sufficient to represent the presence of forest.

Chapter 5

Simulations of neutrally stratified ABL flows using a forest canopy model

5.1 Introduction

In this Chapter we present the simulations of neutrally stratified ABL flows over the existing Mörknässkogen wind farm, with the introduction of a forest model directly in the OpenFOAM simulations, with the aim of improving the deviations found in the validation of the configured mathematical models previously tested in Chapter 4.

Using this additional configuration, it is expected to obtain non-dimensional velocity U_h^* and non-dimensional turbulent intensity I^* profiles closer to the real ones, and ideally reduce the errors obtained in the mathematical model validation to RMS values to around 1% or less, which would result in a very good representation of the real flow structure near the ground.

The canopy model was implemented using OpenFOAM's recently added `kEpsilonLopesdaCosta` turbulence model[1]. This is a variant of the standard $k-\varepsilon$ turbulence model that allows the forest to be modelled as a porous medium, which is a region where fluid flow is affected by resistance due to the porous material. This representation was achieved by creating a porous zone in the same location as the trees, between the ground surface and the canopy top, which then introduces source terms to handle the changes in turbulence in porous regions represented by the `powerLawLopesdaCosta` porosity model. The implementation of this procedure is detailed in this Chapter, from the configuration in OpenFOAM to the complementary tools that were developed.

5.2 Implementing `kEpsilonLopesdaCosta` turbulence model

5.2.1 Porous zone definition

Model implementation begins with the definition of the porous zone or area within the computational domain where the porous media model is applied. In OpenFOAM, this zone is configured as a `porousBlockage` in the `topoSetDict` file that needs to be added to the `system` folder. The `topoSetDict` file is then used to

[1]The `kEpsilonLopesdaCosta` turbulence model was introduced in OpenFOAM version v1812, released in December 2018.

provide settings when running OpenFOAM's *topoSet* utility, which is used to create and manipulate sets and zones of cells, faces or points within the mesh, based on various geometric and field-based criteria.

Porous zone definition involved the creation of a *cellSet* that represents the porous region in the mesh, designated as `porousBlockageCellSet`, using the *boxToCell* method that selects mesh cells based on their spatial position within a box. Taking into account the internal methodology used by `kEpsilonLopesdaCosta` turbulence model, the box was defined for the entire computational domain of 16.0×16.0×3.0 km^3, since the porous media model is only applied below the surface corresponding to the top of the canopy that is defined in Section 5.2.2.

Following the syntax used by OpenFOAM, this box is defined with Cartesian coordinates, from the respective bottom corner (−8000, −8000, 0) to the top corner (8000, 8000, 3000). All cells within this box are included in the cells defined by the `porousBlockageCellSet`. Figure 5.1 shows the code included into the `topoSetDict` file.

```
actions
(
    // porousBlockage
    {
        name    porousBlockageCellSet;
        type    cellSet;
        action  new;
        source  boxToCell;
        box     (-8000 -8000 0) (8000 8000 3000);
    };
    {
        name    porousBlockage;
        type    cellZoneSet;
        action  new;
        source  setToCellZone;
        set     porousBlockageCellSet;
    };
);
```

Figure 5.1: Definition of the porous zone in the `topoSetDict` file within the OpenFOAM `system` folder

5.2.2 Generation of the canopy surface file

In OpenFOAM, the implementation of the `powerLawLopesdaCosta` porosity model requires an STL file that represents the upper surface of the canopy as input, so that the porosity region is only applied below this surface. This STL file, latter called `canopy_LOCAL.stl`, has to be placed inside a new folder called `triSurface`, which needs to be created within the `constant` folder.

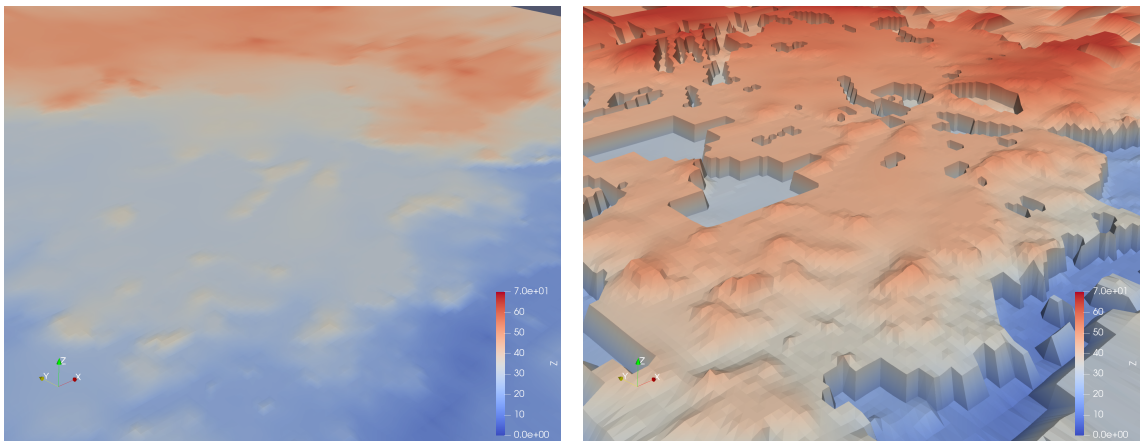
To generate `canopy_LOCAL.stl` file, the procedure involved two steps:

(i) **Create a new topography map that includes the forest height**.

In the absence of a forest map, the topography and roughness maps were manipulated to construct a new map representative of the terrain with trees, designated as `canopy_LOCAL.vtk`. This was achieved with a new developed Python script called `Generate_canopy_vtk.py` (see Appendix B.3), which performs a scan on both the topography and roughness maps (`topo_LOCAL.vtk` and `rough_LOCAL.vtk` respectively), and at the coordinates where in the roughness map the aerodynamic roughness value is equal or greater then $z_0 = 0.7$ m[2], the height of the trees is added to the initial height of the topographic map. Procedure results are shown in Figure 5.2.

(ii) **Extract the upper canopy surface from the new map generated in step (i) to an STL file**.

A simpler approach would be to use Paraview to export the STL surface directly from the new manipulated map `canopy_LOCAL.vtk`. However, another approach was chosen to optimise the process, which consisted of creating a Python script called `vtk_to_stl.py` (see Appendix B.4) that reads the vtk file for the manipulated topographic map `canopy _LOCAL.vtk` and converts the three-dimensional structure into a polygon format, whose geometry is then written to the STL file `canopy_LOCAL.stl`.



(a) (b)

Figure 5.2: Perspective of the topographic map before and after adding tree heights based on aerodynamic roughness values from the roughness map, using `Generate_canopy_vtk.py` script. (a) Before adding the trees. (b) After adding the trees. In Figure (b), VTK surface is magnified 10 times in z direction to make the variation in height more visible.

5.2.3 Setting `powerLawLopesdaCosta` porosity model

The `powerLawLopesdaCosta` porosity model in OpenFOAM is designed to simulate flows through porous media using a power-law relationship for resistance,

[2]Forested areas are identified in the roughness map with aerodynamic roughness values of $z_0 = 0.7$ m in the topographic survey that was performed in the region around Mörknässkogen wind farm.

and its empirical formulation is accounts for the non-linear relationship between the pressure drop and the velocity in porous materials [13].

As investigated in the OpenFOAM source code, `powerLawLopesdaCosta` porosity model is actually an adaptation of the original `powerLaw` porosity model, where the source term is modelled with a power law of the velocity magnitude $|\mathbf{U_i}|$, which can be expressed by:

$$S_i = -\rho C_0 |\mathbf{U_i}|^{(C_1-1)/2}, \tag{5.1}$$

where S_i is the source term associated with the porous region, C_0 and C_1 are user defined linear and exponent empirical coefficients, respectively. In the case of the `powerLawLopesdaCosta` porosity model, the formulation was slightly modified to receive canopy parameters as input, establishing a mathematical approximation relationship between the power law and the drag force that reproduces the momentum dissipation that trees and its foliage should produce in the flow, previously described in equation 2.17.

Combining these two formulations, the linear coefficient C_0 from `powerLaw` porosity model is transformed into $C_z = C_d a(z)$, that represents the relationship between the constant drag coefficient C_d and the local foliage density $a(z)$. The multiplication of the velocity vector component U_i is also introduced, equivalent to equation 2.17. In the exponential term, C_1 remains with the same nomenclature, but now accounts for the effect of eliminating the division by 2. This manipulation results in the equation used in `powerLawLopesdaCosta` porosity model, given by:

$$S_i = -\rho C_d a(z) |\mathbf{U_i}|^{(C_1-1)} U_i. \tag{5.2}$$

The configuration of `powerLawLopesdaCosta` porosity model is conducted in the `fvOptions`[3] file located in the OpenFOAM `constant` folder. Figure 5.3 shows the code included into the `fvOptions` file, and all user-defined parameters that are configured following OpenFOAM's nomenclature are as follows:

(i) `zDir`, the vertical direction within porous region.

(ii) `searchSpan`, the search range or span used to determine the normal vector from the mesh data provided.

(iii) `topSurface`, the top surface file name defining the extent of the porous region, whose file is searched by OpenFOAM in `triSurface` folder, described in Section 5.2.2.

(iv) `groundPatches`, the list of the ground patches defining the lower surface of the porous region.

[3]OpenFOAM's `fvOptions` file is a utility used to add source terms, resistance terms, and other modifications to the governing equations flexibly, without the need to change or compile the solver's source code. It allows for the application of body forces, porosity models, heat sources, damping terms, and other custom influences directly within the computational domain, making it easier to customize and implement various physical phenomena. When `fvOptions` file exists inside the `constant` folder, all the additional options configured within it are sent to the OpenFOAM solver when it is launched via the terminal.

(v) `Sigma`, the porosity surface area per unit volume, represented by the foliage area density $a(z)$.

(vi) `Cd`, the constant drag coefficient of trees C_d.

(vii) `C1`, the model exponent coefficient C_1.

```
porosity1
{
    type                    explicitPorositySource;

    explicitPorositySourceCoeffs
    {
        selectionMode       cellZone;
        cellZone            porousBlockage;
        type                powerLawLopesdaCosta;

        zDir                (0 0 1);
        searchSpan          3000;
        topSurface          "canopy_LOCAL.stl";
        groundPatches       (ground);
        Sigma               0.125;
        Cd                  0.10;
        C1                  1.44;

        coordinateSystem
        {
            origin          (0 0 0);
            rotation        none;
        };
    };
};
```

Figure 5.3: Definition of the `powerLawLopesdaCosta` porosity model in the `fvOptions` file within the OpenFOAM `constant` folder

5.2.4 Setting turbulence model for `kEpsilonLopesdaCosta`

To optimize the simulation launch process, and also make it simple for the user to select the turbulence model `kEpsilonLopesdaCosta` to be used in the simulations, changes were introduced to the source code of the executable *write_turbulenceProperties* developed in [54] to add the possibility of choosing this new turbulence model, which in the original version was not included.

At the same time, a new comment line was added to the original version of `_preprocDict` configuration file, enabling the possibility of selecting the turbulence model `kEpsilonLopesdaCosta` by assigning code 14 to the parameter `model`. Figure 5.4 shows this new configuration possibility, and illustrates a typical configuration of `_preprocDict` file in simulations with a forest model.

With this new procedure, when *write_turbulenceProperties* code configured for the `kEpsilonLopesdaCosta` model is executed, the respective properties are written to the file `turbulenceProperties` (see Figure 5.5) within the folder `constant`, which then provides the properties of the turbulence model to the OpenFOAM solver.

```
#########################################################
# CONFIGURATION OF PREPROCESSING TOOLS FOR OpenFOAM #
#########################################################

# write_blockMeshDict #
sky           = 3000.0    # Height ASL of computational domain top patch [m]
nz            = 60        # Number of cells in z direction
R             = 100.0     # R expansion factor

# write_bCs #
zrs           = 82.0      # Reference height [m]
ustar         = 0.3025    # Friction velocity [m/s]
z0i           = 0.03      # Reference aerodynamic roughness length [m]
delta         = 1000.     # Height of boundary layer [m]
kv            = 0.41      # vonKarman constant
cmu           = 0.033     # kEpsilon model coefficient (betaStar kOmega)

# write_z0 #
z0u           = 0.03      # Uniform aerodynamic roughness length [m]
opt           = 2         # 1 - Uniform roughness mode
                          # 2 - Roughness map in rough_LOCAL.vtk
                          # 3 - Roughness defined by an equation in write_z0.f90

# write_turbulenceProperties #
model         = 14        # kEpsilon:  11-Calibrated,12-Standard,13-Atmospheric
                          # RNGkEpsilon:  21-Original,22-Modified,23-Atmospheric
                          # RealizableKE: 3
                          # kOmega:  4
                          # kOmegaSST: 5
                          # kEpsilonLopesdaCosta:  14

# Canopy #
icano         = 1         # 0 = without canopy ; 1 = with canopy
hcano         = 17.5      # Canopy height [m]
nzcano        = 30        # Number of cells in z direction (below canopy)
```

Figure 5.4: Typical configuration of the `_preprocDict` file used in forest model simulations. The text in blue colour represents the code that was added to the original file throughout this work.

```
simulationType RAS;
RAS
{
RASModel        kEpsilonLopesdaCosta;
turbulence      on;
kEpsilonLopesdaCostaCoeffs
{
    Cmu                 0.0900;
    C1                  1.4400;
    C2                  1.9200;
    sigmak              1.0000;
    sigmaEps            1.3000;
};
printCoeffs     on;
};
```

Figure 5.5: Configuration of the turbulence model `kEpsilonLopesdaCosta` in the `turbulenceProperties` file, written by the *write_turbulenceProperties* executable

5.3 Calibration of canopy parameters

The canopy configuration is a complex procedure by nature, that physically depends on the shape and geometry of the trees, type of foliage and its variation in density with height $a(z)$, seasonal foliage loss, among many others variables. Besides that, in practical terms it is quite dependant of the available information regarding these parameters and their quality. Furthermore, tree characteristics can change seasonally, adding further complexity.

Considering this modelling difficulty, the challenge was the selection of the canopy parameters to be used in the simulations, in terms of canopy height h_{cano} associated with the existing evergreen pine and fir trees around the wind farm region, and the respective foliage density $a(z)$ and drag coefficient C_d that are specified in `fvOptions` (see Section 5.3).

Information related to canopy parameters was consulted through published scientific literature and local photos to set up a representative model, and other additional sources were also used to help their definition, such as satellite info from Google Earth. As a result, it was considered:

(i) A canopy height h_{cano} between 3.0 and 30.0 m AGL, although with a high prevalence of trees taller than 15.0 m AGL (see Figure 5.6), indicated in the Global Forest Watch [57] for the region around Morknaskogen wind farm, whose data is obtained from a study that mapped the global forest canopy height through integration of NASA's GEDI[4] and Landsat data [58].

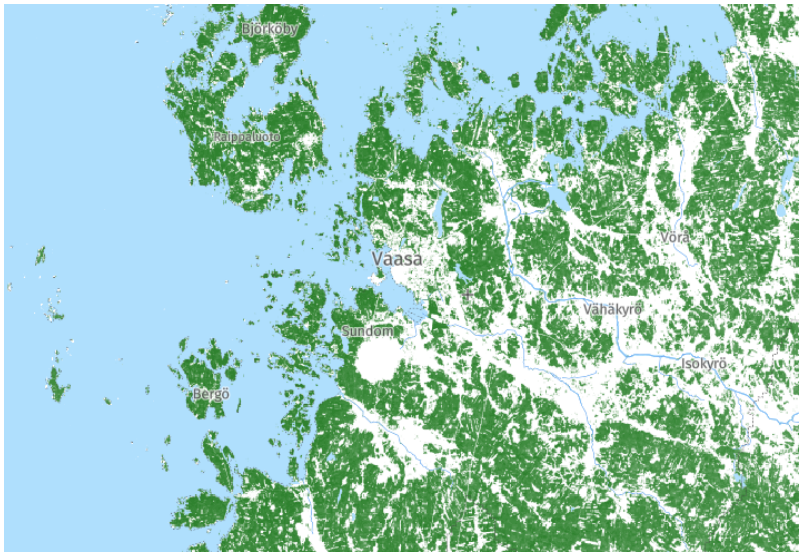


Figure 5.6: Tree cover height in the Vaasa city region, filtered to show trees between 15.0 and 30.0 m AGL tall in green colour

[4]The NASA's GEDI (Global Ecosystem Dynamics Investigation) is a spaceborne lidar instrument operating onboard the International Space Station since April 2019, that consists in a 30 m spatial resolution global forest canopy height map that was developed through the integration of the GEDI lidar forest structure measurements and Landsat analysis-ready data time-series. It provides point-based measurements of vegetation structure, including forest canopy height between 52°N and 52°S globally.

(ii) A drag coefficient C_d that could vary between 0.1 and 0.3 for most vegetation, as suggested in [18].

(iii) A mean leaf area density around $\overline{a(z)} = 0.125$ m^2/m^3, as suggested in [59] for most turbulent flows in plant canopies. Besides that, it was also the value that produced the best agreement with field data in the studies performed in Lopes da Costa (2007) [13]. Despite this, it is a parameter that varies significantly with height z, especially in forested systems, so the ideal procedure is to obtain an empirical value through a calibration process with local wind characteristics.

In order to minimise wind modelling errors, a calibration process of the canopy parameters was performed, which consisted of 25 sets of simulations with different forest characterisations, that resulted in a 5 × 5 matrix of results, where canopy height h_{cano} was varied for the 5 values between 15.0 and 25.0 m AGL, and the drag coefficient C_d between 0.1 and 0.3. All simulations were performed using a 150×150×60 mesh, and the value of $\overline{a(z)} = 0.125$ m^2/m^3 remained constant.

For each set, the *Synthesis* algorithm was applied to calculate the RMS errors related to the cross-prediction of velocity U and turbulent intensity I, to identify the canopy configuration with the best agreement with the wind data. Results of this experiment are synthesised in tables 5.1 and 5.2.

Table 5.1: RMS errors obtained for velocity U cross-prediction with the *Synthesis* algorithm for the calibration process of canopy parameters

RMS_U	C_d					
h_{cano} [m]	**0.10**	**0.15**	**0.20**	**0.25**	**0.30**	
15.0	3.26%	1.56%	1.27%	1.96%	2.73%	A
17.5	2.88%	1.24%	1.50%	2.03%	3.32%	B
20.0	1.34%	2.04%	3.54%	4.80%	5.84%	C
22.5	1.80%	4.08%	5.89%	7.34%	8.53%	D
25.0	1.85%	4.16%	6.00%	7.46%	8.67%	E
	1	2	3	4	5	

Table 5.2: RMS errors obtained for turbulent intensity I cross-prediction with the *Synthesis* algorithm for the calibration process of canopy parameters

RMS_I	C_d					
h_{cano} [m]	**0.10**	**0.15**	**0.20**	**0.25**	**0.30**	
15.0	1.60%	1.39%	1.25%	1.16%	1.08%	A
17.5	1.56%	1.33%	1.17%	1.06%	0.98%	B
20.0	1.45%	1.23%	1.08%	0.98%	0.90%	C
22.5	1.36%	1.16%	1.02%	0.92%	0.85%	D
25.0	1.41%	1.20%	1.05%	0.94%	0.86%	E
	1	2	3	4	5	

Results revealed that the best canopy configuration in terms of velocity U was obtained in SET_{B2} ($h_{\text{cano}} = 17.5$ m, $C_d = 0.15$), with RMS errors of 1.24% for velocity U, and 1.33% for turbulent intensity I.

On the other hand, turbulent intensity I produced better results in SET_{D5} ($h_{\text{cano}} = 22.5$ m, $C_d = 0.30$), which was expected due to the higher values of C_d, as a higher resistance to motion creates forces that disturb the flow, increasing speed variations and the formation of eddies, which increases overall turbulence as a result.

However, it is considered that the RMS errors obtained for turbulent intensity I are satisfactorily low in all sets (see Figure 5.2), and therefore the choice of the best set was made based on velocity U, which will also be more important for estimating the wind turbines energy production. It is estimated that this calibration process consisted of 487 hours of computational processing, although in practical situations such a high number of tests may not be necessary.

5.4 New `canopyHD` mesh type

In an advanced stage of this work, it was considered that it would be useful to introduce an improvement to the three-dimensional mesh generation process, which consisted of increasing mesh resolution in the area below the canopy in order to make this region better defined, which would theoretically allow to better capture the changes in the flow within this most critical region.

The strategy used to achieve this improvement consisted of changing *write_blockMeshDict* source code, which is responsible for creating the mesh configuration file `blockMeshDict` used by the *blockMesh* utility, to create the new version called *write_blockMeshDictCano*.

In the original version of *write_blockMeshDict*, the code generates the three-dimensional mesh based on vertically oriented hexahedron blocks that are extruded in z direction, up to the height corresponding to the `sky` parameter. This prism is then divided into nz control volumes, whose vertical spacing gradually increases in z direction at a geometric expansion ratio R.

In the new version, called the `canopyHD` mesh type, the approach was to split this vertical block into two different vertical blocks. The first block is defined between the ground and the canopy height, and is divided into a given number of nz_{cano} control volumes, evenly spaced at a distance of $\Delta z_{\text{cano}} = h_{\text{cano}}/nz_{\text{cano}}$. The second block is defined above the height of the canopy, and extends to the top of the computational domain corresponding to the height configured in the `sky` parameter, using a geometrical expansion as before[5].

Settings for this new type of mesh generation were added to the `_preprocDict` configuration file (see Figure 5.4), which in addition to the recently added new configuration to select `kEpsilonLopesdaCosta` turbulence model, now also includes three new parameters to define the application of this method:

[5]It is remarked that this new version may also be used without canopy, and the numbering of the OpenFOAM mesh was made more intuitive.

(i) `icano`, the option to select the new mesh generation type `canopyHD`. Set the value 1 for `canopyHD` mesh, or value 0 for original mesh.

(ii) `hcano`, the canopy height [m].

(iii) `nzcano`, the number of control volumes in the z direction, within the high-resolution region below the canopy.

In `blockMeshDict` file, hexahedron blocks written by *write_blockMeshDictCano* have the structure shown in Figure 5.7, whose sequence of numbers represents the indices of the vertices that define the hexahedron, which are defined in a specific order in OpenFOAM.

In Figure 5.7, the first two hexahedrons refer to the row of blocks below the canopy, in this case with $nz_{\text{cano}} = 10$ and a grading factor for the z direction equal to 1.0 (uniform cell size) used in *simpleGrading*[6] method. The last two hexahedrons refer to the row of blocks above the canopy, defined with $nz = 60$ and expansion ratio of $R = 100.0$ in the positive z direction. Result of implementing the `canopyHD` mesh type is represented in Figure 5.8.

```
blocks
(
  hex (    0     1    150    149  44402  44403  44552  44551) (1 1 10) simplegrading (1 1   1.0)
  hex (    1     2    151    150  44403  44404  44553  44552) (1 1 10) simplegrading (1 1   1.0)
  ...
  hex (46493  46494  46643  46642  24292  24293  24442  24441) (1 1 60) simplegrading (1 1 100.0)
  hex (46494  46495  46644  46643  24293  24294  24443  24442) (1 1 60) simplegrading (1 1 100.0)
  ...
);
```

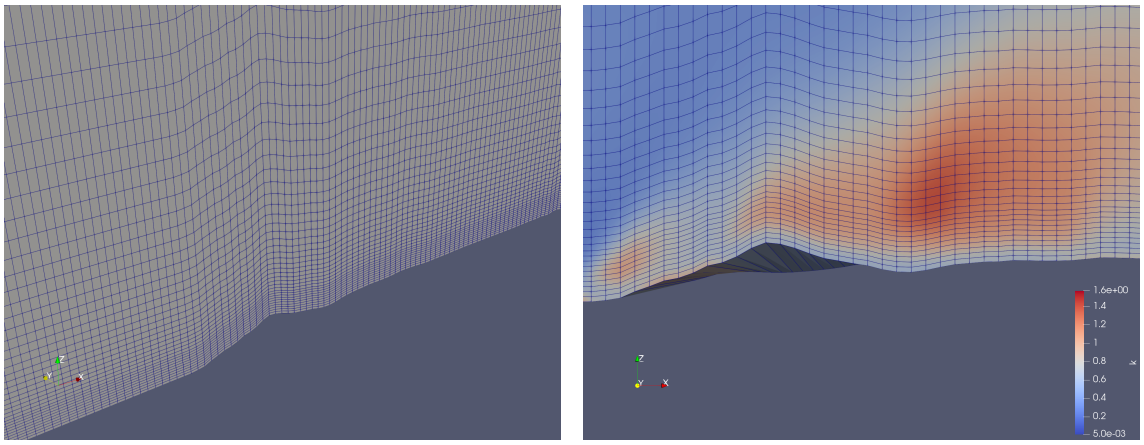
Figure 5.7: Structure of the hexahedron blocks written in file `blockMeshDict` by *write_blockMeshDictCano*, for a simulation configured for `canopyHD` mesh type.

In terms of the *Synthesis* algorithm, it was also necessary to make changes to the Python code to account for the new mesh structure when the `canopyHD` mesh type is used, since the order of the points was completely changed, and for all intents and purposes the final numerical mesh is a set of one mesh above other mesh in the z direction. To overcome this difficulty, a section of code was programmed to perform a sequence of matrix operations to transform the `canopyHD` mesh structure into the original mesh structure, so that it can be read and processed in the same way (see Appendix B.2).

5.5 Presentation and discussion of results

This Section presents the results obtained from the simulations performed with the implementation of a forest model defined by the `kEpsilonLopesdaCosta` turbulence model, and the conclusions arising from this procedure in terms of the improvements that the introduction of this model produced when compared to the simulations previously performed without a forest model in Chapter 4. In addition, the results

[6]The *simpleGrading* method in OpenFOAM is used to define the distribution of cells in a mesh block. It specifies how the size of cells changes linearly or exponentially along each coordinate direction, using three grading factors, one for each direction (x, y, z), where a factor of 1 indicates uniform cell size, and a value greater or less than 1 indicates progressive expansion or contraction of cells, respectively, from one side of the block to the other.



(a) (b)

Figure 5.8: Perspective of the highest resolution area below the canopy. (a) Isometric view of the mesh. (b) Front view with results example for k. In both figures, numerical mesh is magnified 10 times in z direction.

were also subjected to a sensitivity analysis concerning the canopy height, canopy parameter $C_z = C_d\overline{a(z)}$, type of vertical mesh generation and grid dimensions.

As mentioned in Section 5.3, the best results were obtained for SET_{B2} when the calibration of canopy parameters was performed, corresponding to a 150×150×60 mesh of default generation type, $h_{\text{cano}} = 17.5$ m, $C_d = 0.15$ and $\overline{a(z)} = 0.125$ m^2/m^3.

Starting from SET_{B2}, three additional sets were produced to test the application of the new `canopyHD` mesh type for three levels of vertical mesh refinement in the area of higher resolution below the canopy, with nz_{cano} values of 10, 20 and 30 control volumes within that region. Having fixed the canopy height at $h_{\text{cano}} = 17.5$ m AGL in these simulations, this corresponds to a vertical spacing between mesh cells Δz_{cano} of 1.750, 0.875 and 0.583 m, respectively. These sets were assigned SET_{210}, SET_{211} and SET_{212}.

To conclude, the previous set with highest mesh resolution, corresponding to SET_{212} with $nz_{\text{cano}} = 30$ ($\Delta z_{\text{cano}} = 0.583$ m), was selected to produce two extra sets with 180 × 180 horizontal mesh configuration for SET_{213} and 250 × 250 for SET_{214}, with the aim to perform a sensitivity analysis to the horizontal mesh resolution.

Results obtained from all performed sets are described in Table 5.3, which contains information regarding all configurations used in terms of mesh generation and canopy parameters, and the RMS errors obtained through the *Synthesis* algorithm for velocity U and turbulent intensity I cross-predictions between mast measurement heights. It also shows the average number of iteration $\overline{It_i}$ that was required for the simulations to converge.

After analysing all obtained results, it was concluded that the best values in terms of mathematical model validation were achieved in SET_{210}, with RMS errors of 1.13% for velocity and 1.31% for turbulent intensity. However, and despite the

Table 5.3: Summary of all 396 performed simulations for Mörknässkogen wind farm. Mean leaf area density set to $\overline{a(z)} = 0.125$ m^2/m^3 for all simulations. SET_{B2} coloured in light gray was the one that presented the best results in the process of calibrating canopy parameters. SET_{210} coloured in light green was the best set overall. Δx represents the spacing between mesh cells within the highest resolution central area. Legend of additional symbols shown at the end of the table. $\overline{It}$ is the average number of iterations of the 12 simulations.

Ref.	$nx \times ny \times nz$	nz_{cano}	Δx [m]	h_{cano} [m]	C_d	RMS_U [%]	RMS_I [%]	$\overline{It}$
# SET_{100}	150×150×60	-	50.0	-	-	20.03	4.95	1011
# SET_{101}	150×150×60	-	50.0	-	-	14.38	3.48	979
# SET_{102}	150×150×60	-	50.0	-	-	13.06	3.21	976
SET_{A1}	150×150×60	-	50.0	15.0	0.10	3.26	1.60	902
SET_{A2}	150×150×60	-	50.0	15.0	0.15	1.56	1.39	891
SET_{A3}	150×150×60	-	50.0	15.0	0.20	1.27	1.25	884
SET_{A4}	150×150×60	-	50.0	15.0	0.25	1.96	1.16	879
SET_{A5}	150×150×60	-	50.0	15.0	0.30	2.73	1.08	875
SET_{B1}	150×150×60	-	50.0	17.5	0.10	2.88	1.56	896
SET_{B2}	150×150×60	-	50.0	17.5	0.15	1.24	1.33	886
SET_{B3}	150×150×60	-	50.0	17.5	0.20	1.50	1.17	880
SET_{B4}	150×150×60	-	50.0	17.5	0.25	2.03	1.06	875
SET_{B5}	150×150×60	-	50.0	17.5	0.30	3.32	0.98	871
SET_{C1}	150×150×60	-	50.0	20.0	0.10	1.34	1.45	887
SET_{C2}	150×150×60	-	50.0	20.0	0.15	2.04	1.23	876
SET_{C3}	150×150×60	-	50.0	20.0	0.20	3.54	1.08	870
SET_{C4}	150×150×60	-	50.0	20.0	0.25	4.80	0.98	866
SET_{C5}	150×150×60	-	50.0	20.0	0.30	5.84	0.90	863
SET_{D1}	150×150×60	-	50.0	22.5	0.10	1.80	1.36	877
SET_{D2}	150×150×60	-	50.0	22.5	0.15	4.08	1.16	868
SET_{D3}	150×150×60	-	50.0	22.5	0.20	5.89	1.02	863
SET_{D4}	150×150×60	-	50.0	22.5	0.25	7.34	0.92	859
SET_{D5}	150×150×60	-	50.0	22.5	0.30	8.53	0.85	856
SET_{E1}	150×150×60	-	50.0	25.0	0.10	1.85	1.41	875
SET_{E2}	150×150×60	-	50.0	25.0	0.15	4.16	1.20	867
SET_{E3}	150×150×60	-	50.0	25.0	0.20	6.00	1.05	862
SET_{E4}	150×150×60	-	50.0	25.0	0.25	7.46	0.94	858
SET_{E5}	150×150×60	-	50.0	25.0	0.30	8.67	0.86	856
△ SET_{210}	150×150×70	10	50.0	17.5	0.15	1.13	1.31	1287
△ SET_{211}	150×150×80	20	50.0	17.5	0.15	1.21	1.33	1842
△ SET_{212}	150×150×90	30	50.0	17.5	0.15	1.16	1.45	2791
△ SET_{213}	180×180×90	30	40.0	17.5	0.15	1.24	1.60	2762
△ SET_{214}	250×250×90	30	30.0	17.5	0.15	1.57	1.54	2705

△ Simulations with `canopyHD` mesh type.

Simulations without canopy model.

fact that this was the best set, the final set chosen to perform the wind resource assessment was SET_{214} since it was the one with the highest overall mesh resolution, with an increase of +257.14% in the number of mesh cells, which thus allows to capture terrain topography more accurately. Besides that, the RMS errors obtained in SET_{214} of 1.57% for velocity and 1.54% for turbulent intensity only have non-significant differences when compared to SET_{210}, resulting in a small loss of accuracy of −0.44% for velocity and −0.23% for turbulent intensity. Results of the numerical model validation performed with SET_{214} are described in detail in Table 5.4.

Table 5.4: Mathematical model validation with *Synthesis* algorithm for SET_{214} ($h_{cano} = 17.5$ m, $C_d = 0.15$, `canopyHD` mesh type)

From height [m]	**To height** [m]	U_{mast} [m/s]	U_{calc} [m/s]	ϵ [%]	I_{mast} [%]	I_{calc} [%]	ϵ [%]
Mast 120	Mast 82	5.838	5.711	-2.18	12.575	12.945	0.37
Mast 120	Mast 48	4.588	4.531	-1.24	18.683	16.813	-1.87
Mast 82	Mast 120	6.741	6.892	2.24	10.010	9.721	-0.29
Mast 82	Mast 48	4.588	4.630	0.92	18.683	16.343	-2.34
Mast 48	Mast 120	6.741	6.834	1.38	10.010	11.300	1.29
Mast 48	Mast 48	5.838	5.787	-0.87	12.575	14.420	1.85
RMS		-	-	**1.57**	-	-	**1.54**
RMS_{120}		-	-	**1.77**	-	-	**1.35**
BIAS		-	-	0.04	-	-	-0.17
STD		-	-	1.72	-	-	1.68

Vertical cross-prediction errors are low in all combinations performed between the mast measurement heights, even considering the lowest level at 48 m AGL, where larger errors were expected because it is quite close to the top of the existing trees.

In Table 5.4 was added the value of the RMS error corresponding only to the top anemometers pair at 120 m AGL, designated as RMS_{120}, since it is more representative of the expected errors in the extrapolations between 120 m AGL and the turbine hub heights of 166 m AGL. However, for the case of vertical cross-predictions of velocity, the RMS_{120} error is higher than RMS at +0.20%, which indicates that the inclusion of 48 m AGL measurements still yields lower overall errors, contrary to what would be expected.

As an example, Figures 5.9 and 5.10 show the velocity and TKE maps obtained for the most frequent wind direction of 210°, at hub height of 166 m AGL. Velocity and TKE maps for all wind directions can be consulted in Appendix E and Appendix F, respectively. Figure 5.10 shows that installed turbines are located in a high turbulence area when compared to the surroundings, although its magnitude may not be concerning. That analysis will be conducted in more detail in Section 5.5.7.

In Figure 5.12 is also represented the map of TKE at canopy height, corresponding to 17.5 m AGL in the case of SET_{214}, where the influence of the trees on local turbulence production is notorious due to the velocity gradient associated with these regions. Maps for all directions can be consulted in Appendix G.

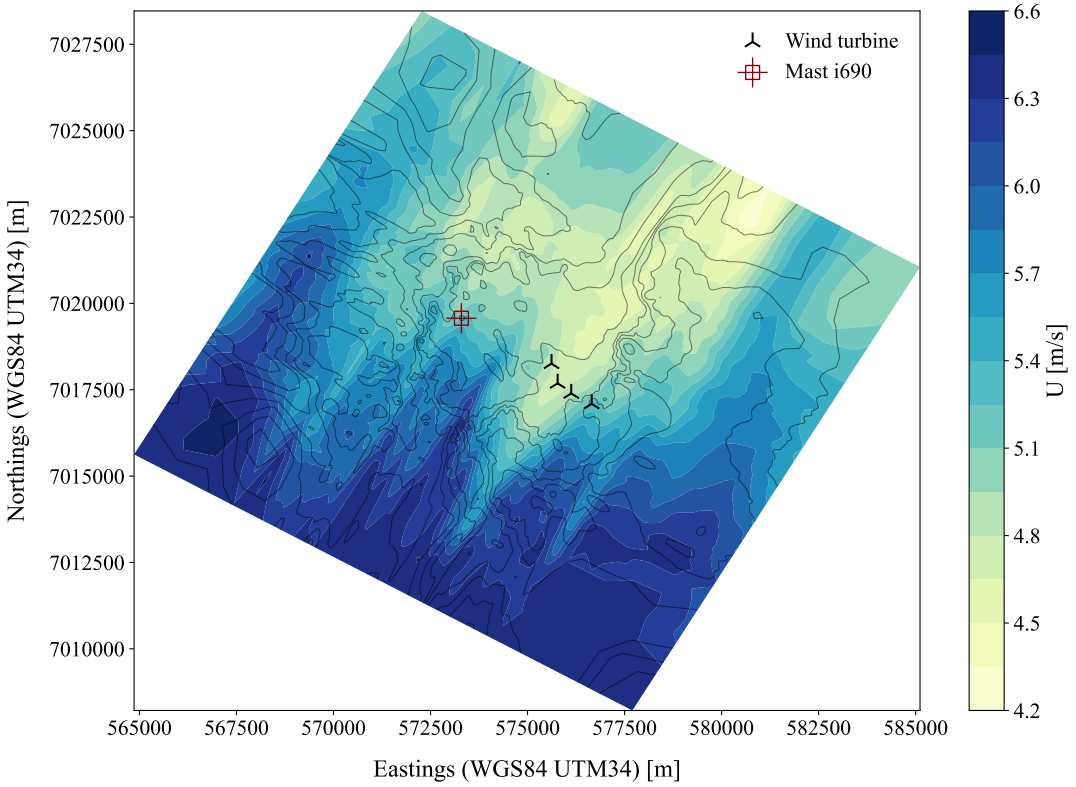


Figure 5.9: SET_{214} velocity field for 210° wind at 166 m AGL (hub height)

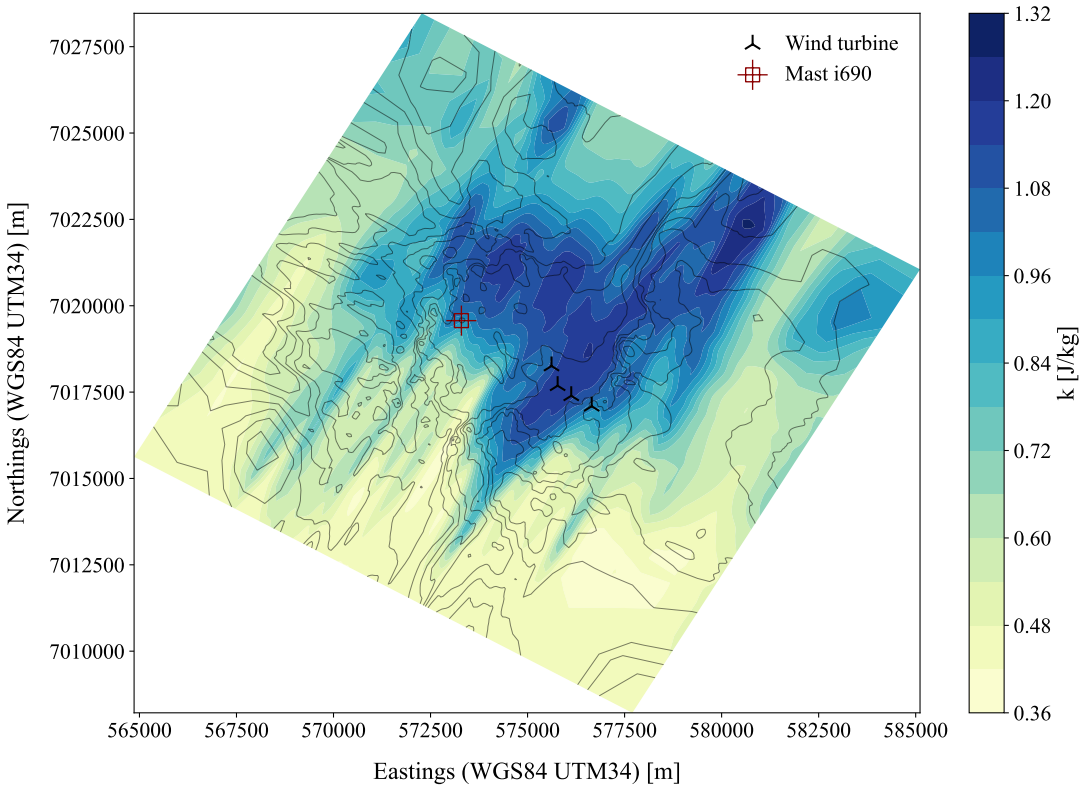


Figure 5.10: SET_{214} TKE field for 210° wind at 166 m AGL (hub height)

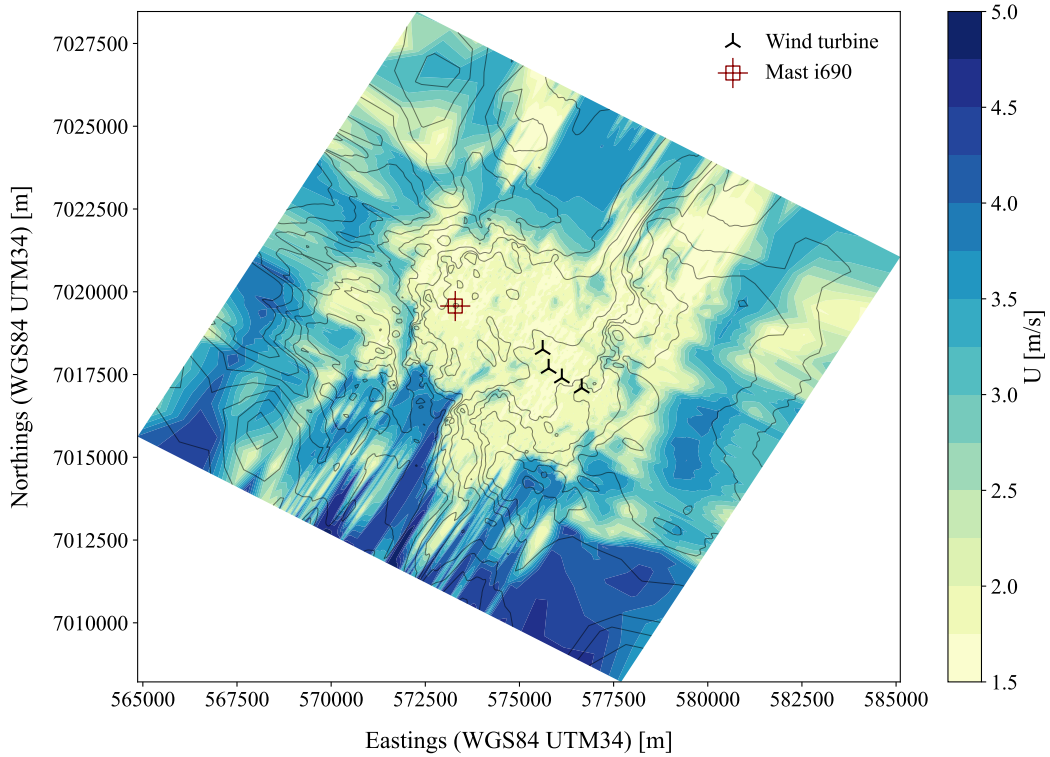


Figure 5.11: SET_{214} velocity field for 210° wind at 17.5 m AGL (top canopy height)

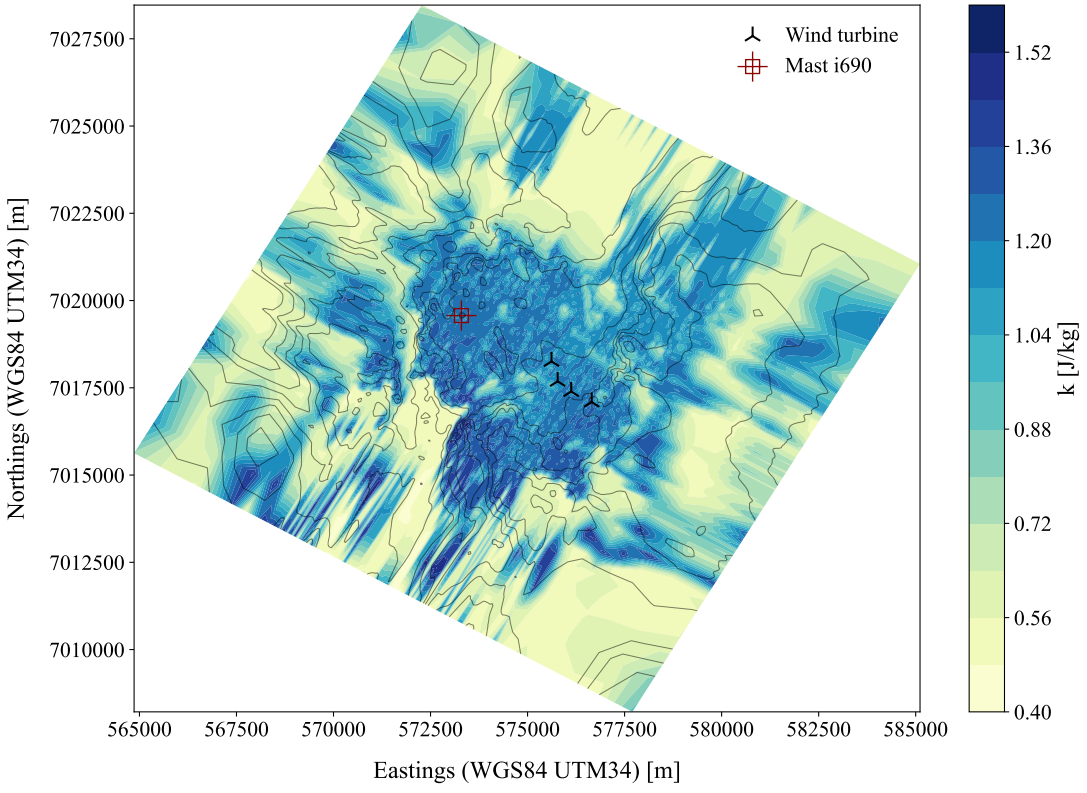


Figure 5.12: SET_{214} TKE field for 210° wind at 17.5 m AGL (top canopy height)

5.5.1 Canopy height

Figure 5.13 shows the vertical wind profiles obtained for a 210° wind from OpenFOAM simulations compared with the measured data at mast i690 coordinates for SET_{A2}, SET_{C2}, SET_{E2} and SIM_{102}, in terms of non-dimensional horizontal velocity U_h^* and non-dimensional TKE k^*.

These sets were selected to evaluate the effect of canopy height h_{cano} on the simulations, having been chosen the lowest, medium, and highest heights tested, corresponding to the values of 15.0, 20.0, 25.0 m AGL. The same trio of sets could have been chosen for any of the drag coefficient C_d values tested between 0.1 and 0.3 in Section 5.3, but it was chosen $C_d = 0.15$ since it was the one that presented the best agreement between experimental and calculated values. For all following sections, SET_{102} (Roug×2) was also included into the analysis to assess the impact of having implemented a forest model on the simulations.

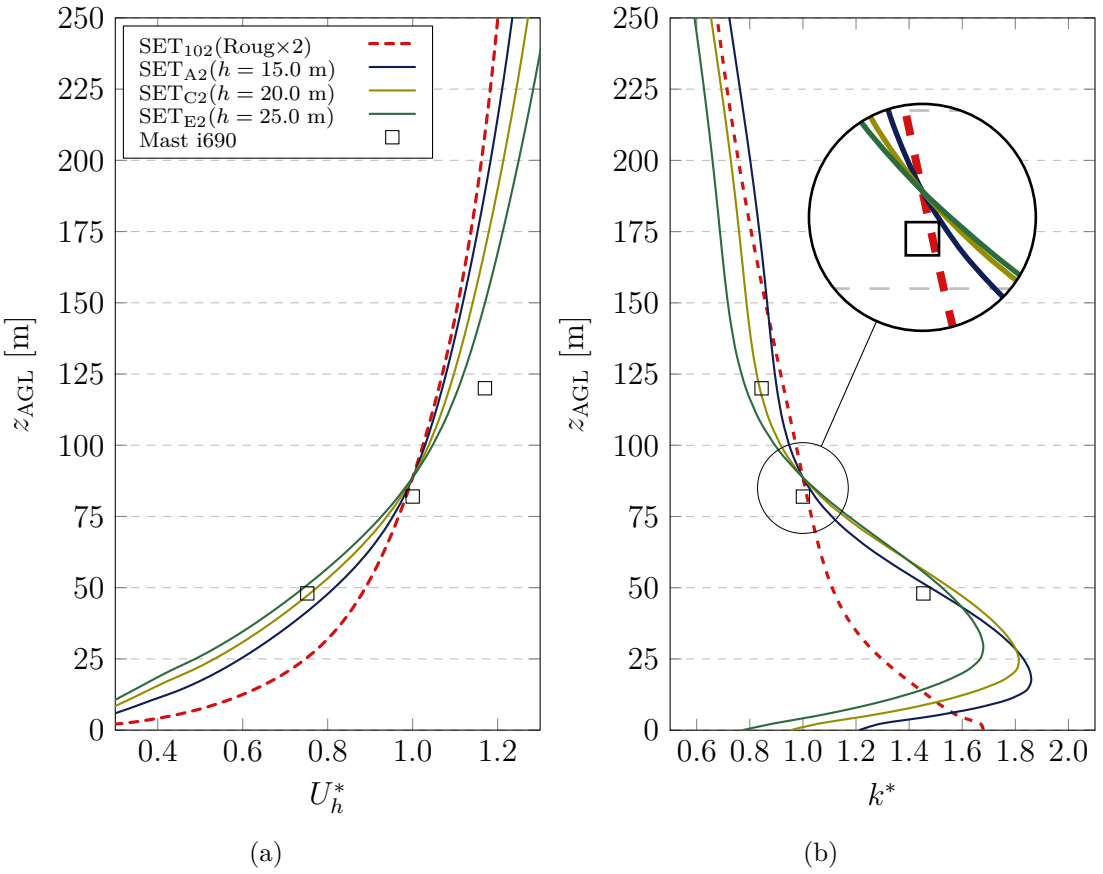


Figure 5.13: Vertical profiles at mast i690 coordinates for a 210° wind to analyse the influence of canopy height. (a) Non-dimensional horizontal velocity U_h^*. (b) Non-dimensional TKE k^*.

Presented results demonstrated a significant improvement in vertical profiles with the introduction of a forest model, regardless of the configured canopy height. There is a notorious improvement of the velocity prediction at 48 m AGL as a result of the momentum dissipation caused by the trees, which decreases the velocity at lower altitudes. At 120 m AGL, there is also a better agreement between calculated and

experimental values measured at mast i690 due to the increase in velocity caused by a higher vertical component of the velocity w.

Non-dimensional TKE k^* vertical profile was completely modified due to a turbulence increase near the forest caused by flow separation and compression phenomena. Thus, agreement between calculated and experimental results is vastly increased, enabling to capture the measured value at 48 m AGL. Graphical analysis also shows that lower canopy heights have stronger turbulence peaks due to higher magnitude and more frequent velocity variations at lower heights.

Non-dimensional turbulent intensity I^* vertical profiles illustrated in Figure 5.14 were consistent with the improvements presented in Table 5.3, with cross-predictions resulting in RMS error values at mast i690 between 0.85% and 1.60%, which are significantly lower than the best result obtained with 3.21% in SET_{102}.

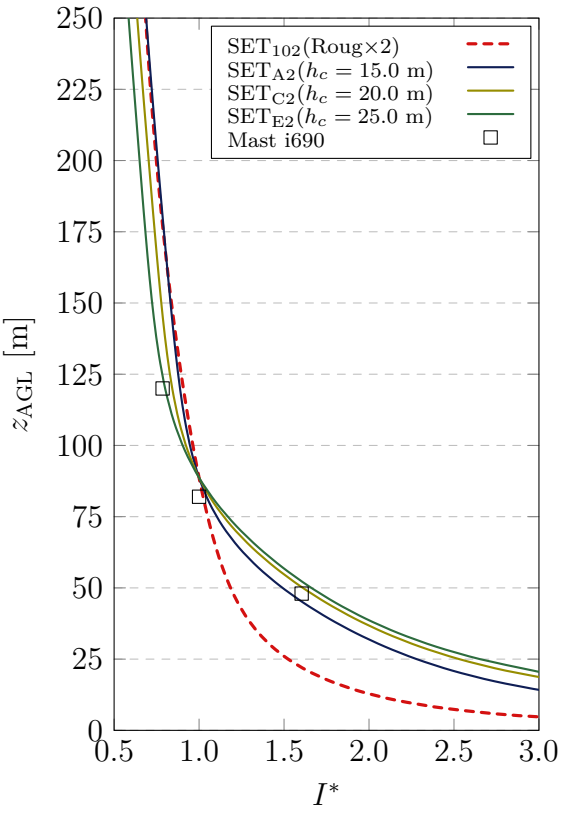


Figure 5.14: Non-dimensional turbulent intensity I^* vertical profile at mast i690 coordinates for a 210° wind to analyse the influence of canopy height.

In terms of shape, turbulent intensity profiles are very identical in all simulations performed with forest model, regardless of canopy parameters, mesh type or grid dimensions, so its representation in the following sections was omitted.

5.5.2 Canopy parameter C_z

Canopy parameter $C_z = C_d\overline{a(z)}$, which was defined with $\overline{a(z)} = 0.125$ m^2/m^3 and C_d values between 0.1 and 0.3, has a notorious influence on the mathematical model validation results due to its direct action in the governing equation of `powerLawLopesdaCosta` porosity model 5.2.

Figure 5.15 shows both the calculated and experimental vertical wind profiles obtained for a 210° wind referring to SET_{B1}, SET_{B3}, SET_{B5} and SET_{102}, in terms of non-dimensional horizontal velocity U_h^* and non-dimensional TKE k^*.

These sets were selected to evaluate the effect of canopy parameter C_z on the simulations, having been chosen the lowest, medium, and highest drag coefficients C_d tested, corresponding to the values of 0.1, 0.2 and 0.3. The same trio of sets could have been chosen for any of the canopy height h_{cano} tested between 15.0 and 25.0 m AGL in Section 5.3, but it was decided to use h_{cano} = 17.5 m AGL since it was the height that presented the best agreement between experimental and calculated values. SET_{102} was also included in the analysis to verify the impact of having implemented a forest model on the simulations.

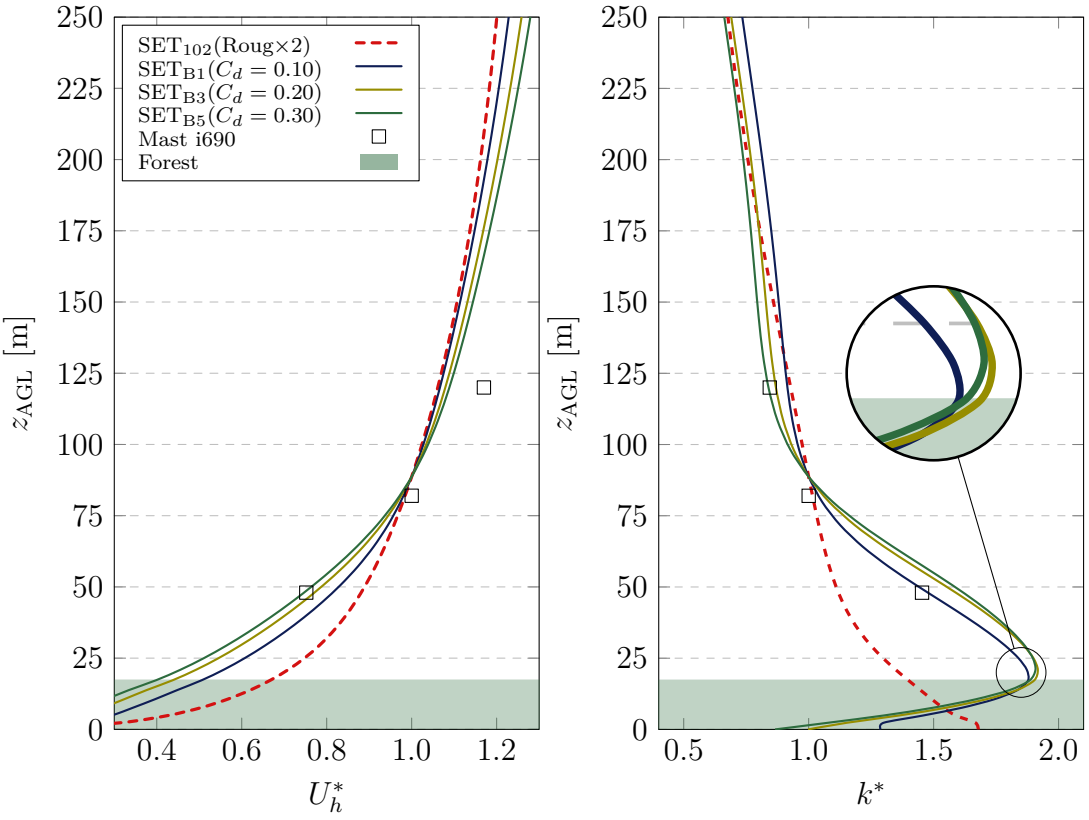


Figure 5.15: Vertical profiles at mast i690 coordinates for a 210° wind to analyse the influence of canopy parameter $C_z = C_d\overline{a(z)}$. (a) Non-dimensional horizontal velocity U_h^*. (b) Non-dimensional TKE k^*.

Similar to the conclusions obtained in the previous Section 5.5.1, results demonstrated a significant improvement on the velocity prediction at 48 m AGL, as well as at 120 m AGL. In terms of the non-dimensional TKE k^* profile, it was concluded that lower values of drag coefficient C_d result in better predictions of the turbulent intensity at 48 m AGL, while high values of C_d result in better predictions at 120 m AGL.

Non-dimensional TKE k^* peak values obtained at canopy height are higher for higher values of C_d due to higher resistance to the fluid motion. This is due to the

fact that a higher interaction between the flow and the surface tends to increase flow separation, recirculation and vortex generation, leading to an increase in TKE.

5.5.3 `Default` vs `canopyHD` mesh type

To evaluate the effect of using the new `canopyHD` vertical mesh generation type described in Section 5.4 instead of the standard `default` mesh type developed in [54], a comparative analysis was performed between SET_{210}, SET_{211} and SET_{212} generated with this new tool, and SET_{B2} generated with the standard procedure.

As summarised in Table 5.3, these sets were designed for three vertical mesh refinement levels under the canopy top with constant spacing, with $nz_{cano} = 10$ for SET_{210}, $nz_{cano} = 20$ for SET_{211}, and $nz_{cano} = 30$ for SET_{212}, with Δz_{cano} of 1.750, 0.875 and 0.583 m, respectively. Horizontal mesh with 150×150 resolution was kept equal for all simulations, as well as the canopy height of $h_{cano} = 17.5$ m AGL and constant drag coefficient of $C_d = 0.15$.

Figure 5.16 presents the vertical profiles obtained for a 210° wind referring to these sets. Graphical analysis revealed global improvements when using `canopyHD` mesh type, particularly in cross-predictions of U_h^* at 120 m AGL and overall profile of k^*.

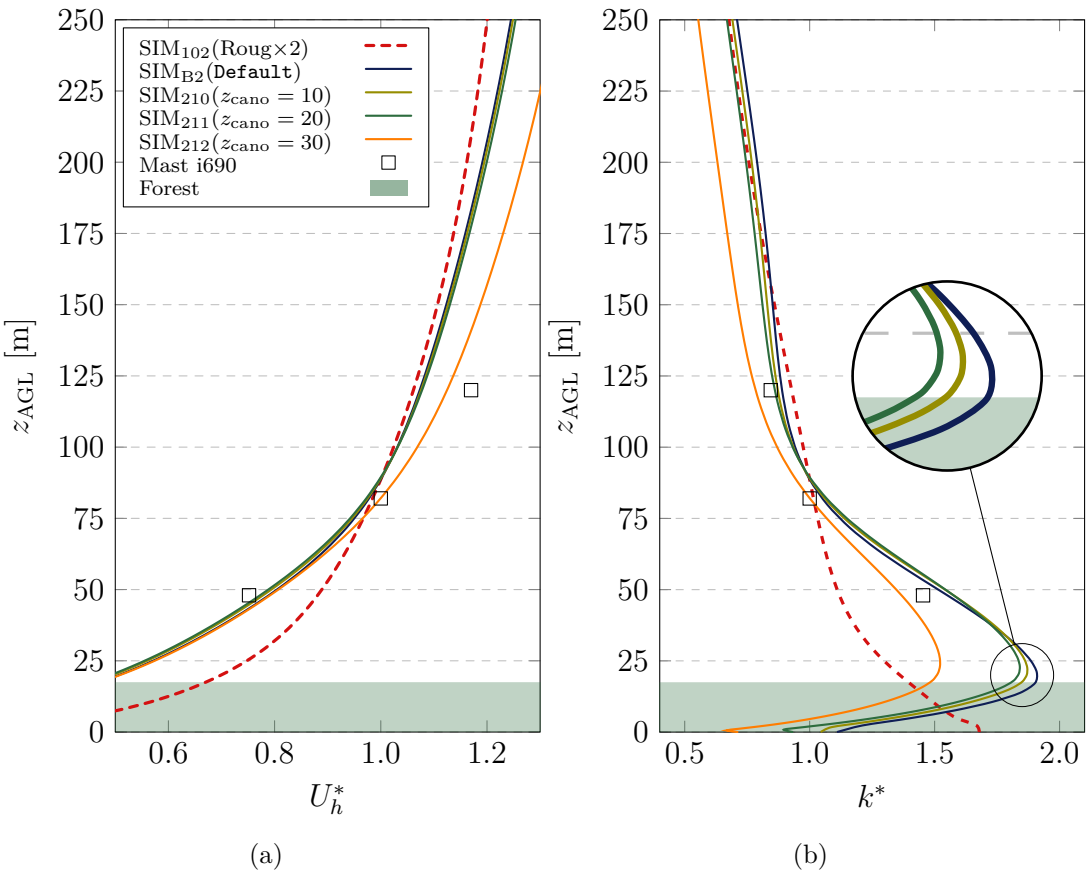


Figure 5.16: Vertical profiles at mast i690 coordinates for a 210° wind to analyse the influence of using `canopyHD` mesh type.(a) Non-dimensional horizontal velocity U_h^*. (b) Non-dimensional TKE k^*.

The result is due to the large increase in resolution between the canopy top and the terrain surface (*ground* patch), which by becoming numerically better defined appears to capture conditions at forested sites more accurately. Note that, all sets from A1 to E5 produced in the calibration of canopy parameters (see Section 5.3), including SET_{B2}, only have 7 complete control volumes below the canopy height at $h_{cano} = 17.5$ m AGL, with vertical spacing between cells varying from 2.34 and 3.74 m due to the geometric expansion factor R in the z direction.

With the new mesh, even comparing with SET_{210} which has the lowest resolution between the three sets with $nz_{cano} = 10$, the increase in resolution along the tree height ranges between +133.8% near the ground surface and +213.6% at $z \approx 17.5$ m AGL, corresponding to an average of +171.1%. That increase is much higher in SET_{212} with $nz_{cano} = 30$, ranging from +401.3% near the ground surface and +640.9% at $z \approx 17.5$ m AGL, corresponding to an average of +513.3%.

5.5.4 Horizontal grid dimensions

The influence of the horizontal mesh refinement was analysed using the latest sets produced, which concerned SET_{212} (150×150), SET_{213} (180×180) and SET_{214} (250×250), which were configured for numerical meshes with horizontal spacing between cells within the highest resolution central region of 50, 40 and 30 m, respectively, in both x and y directions.

Figure 5.17 shows the vertical wind profiles obtained for a 210° wind from OpenFOAM simulations compared with the measured data at mast i690 coordinates for the previously described sets.

Once more, calculated results revealed interesting improvements in the vertical wind profiles, especially in terms of the velocity prediction enhancement at 120 m AGL, indicating a better agreement between calculated and experimental data at higher altitudes, despite the fact that results are progressively worse in terms of the mathematical model validation where the overall RMS error obtained for velocity cross-predictions increased from 1.16% in SET_{212} to 1.24% in SET_{213} and 1.57% in SET_{214}.

However, these small differences are considered to be negligible when compared to the benefits associated with a rather more refined mesh, which in this case has an increase in resolution of +277.8% between SET_{212} (150 × 150) and SET_{214} (250 × 250) which translates into +3600000 additional points, even though it possesses the disadvantage of additional computational resources needed.

This behaviour leads to the conclusion that SET_{214} (250 × 250) was the best in terms of proximity to the measured velocity results obtained at mast i690. Thus, it increases the confidence regarding the extrapolated results when transporting the measured time series at mast i690 to the coordinates of the wind turbines at 166 m AGL, as a result of the improved prediction for higher altitudes.

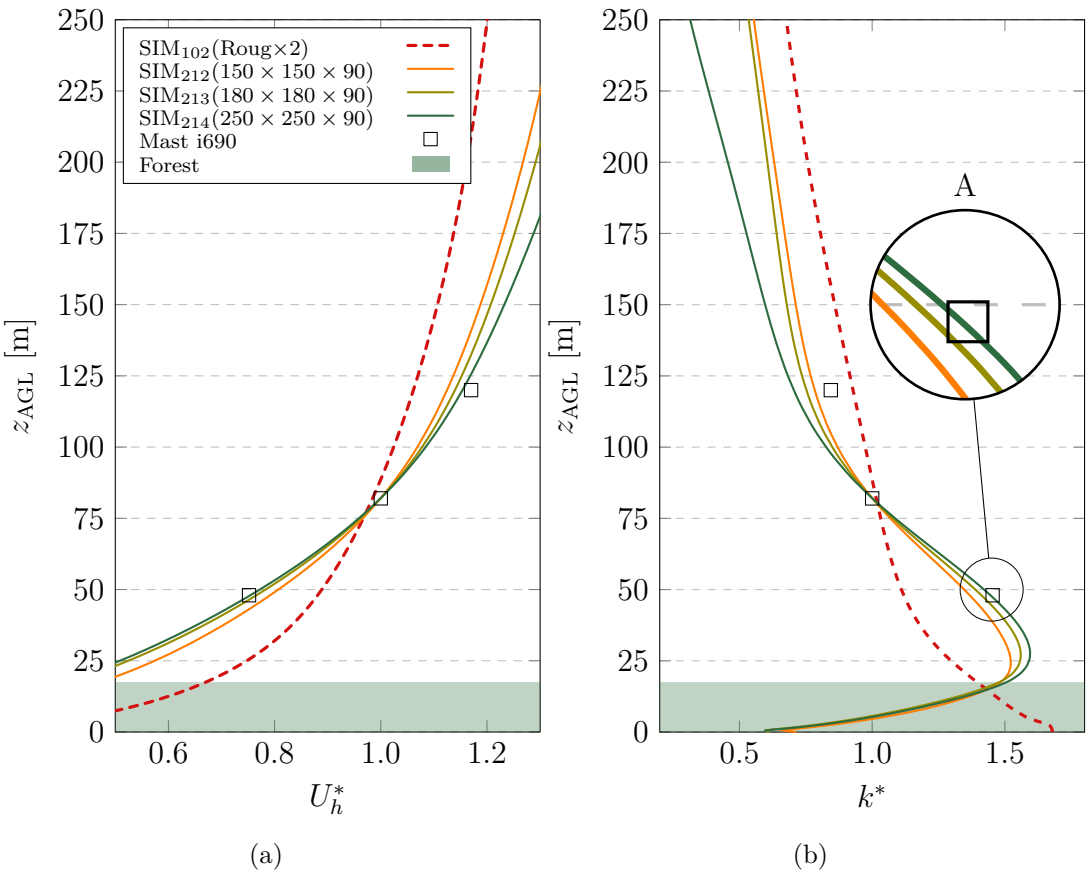


Figure 5.17: Vertical profiles at mast i690 coordinates for a 210° wind to analyse the influence of horizontal mesh resolution.(a) Non-dimensional horizontal velocity U_h^*. (b) Non-dimensional TKE k^*.

As for the turbulence profile, graphical analysis demonstrates a perfect agreement at 48 m AGL for SET_{214} (250 × 250), as shown in the Detail A in Figure 5.17. For higher heights, approximately above 82 m AGL, results begin to show worse agreements despite the fact that in terms of mathematical model validation the RMS errors obtained for turbulence intensity I remain low with 1.45% for SET_{212}, 1.60% in SET_{213} and 1.54% in SET_{214}.

5.5.5 Wind farm energy production

Having chosen SET_{214} as the best set of simulations, the wind farm's theoretical AEP (Annual Energy Production) is calculated based on the time series synthesised for the coordinates of each turbine at hub height, which together represents the average wind climates in the region. This data is then combined with the power curve of the installed turbine model, to estimate the reference gross annual energy production. This estimate corresponds to the most likely scenario for the wind farm gross annual energy production, assuming an ordinary year with 365 days.

This calculation procedure was included into the Python post-processing algorithm (see Appendix B.2), whose strategy for calculating the AEP_i for each wind turbine i was the following:

(i) Read the synthesised time series that were transported to each wind turbine,

the result of which is partially shown in Figure 5.18.

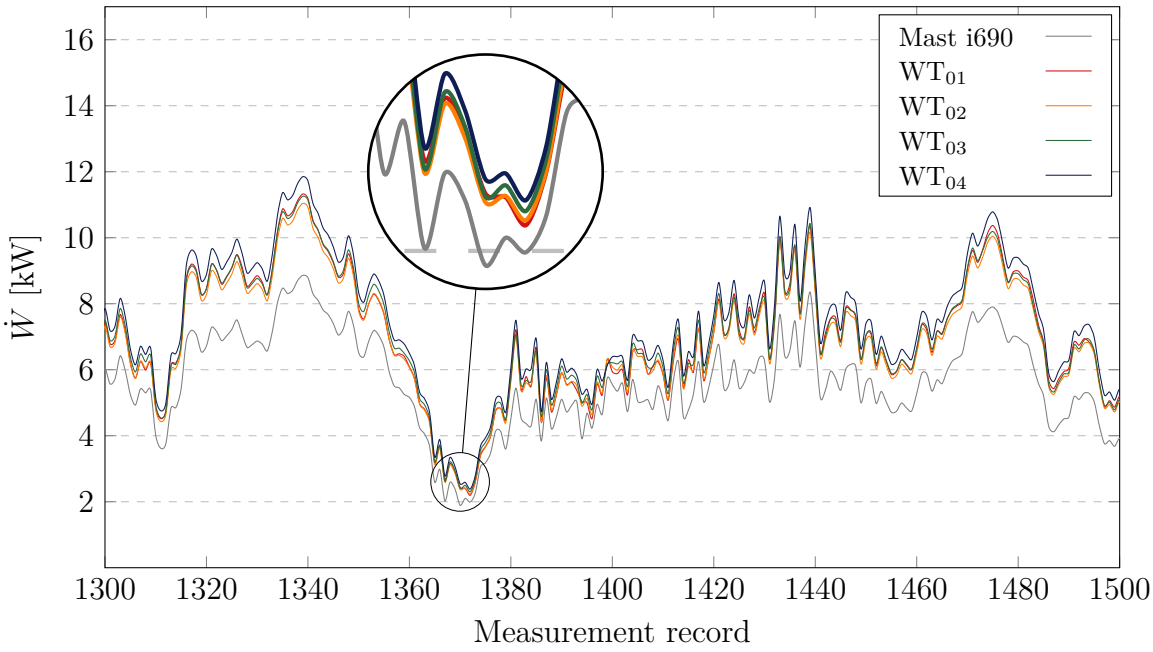


Figure 5.18: Partial section of the transported time series from mast i690 to the four wind turbines installed using results from SET_{214}

(ii) For each record, look for the value of the magnitude of the average speed and, using the power curve presented in Figure 5.19, look up the respective power for this wind speed through linear interpolation.

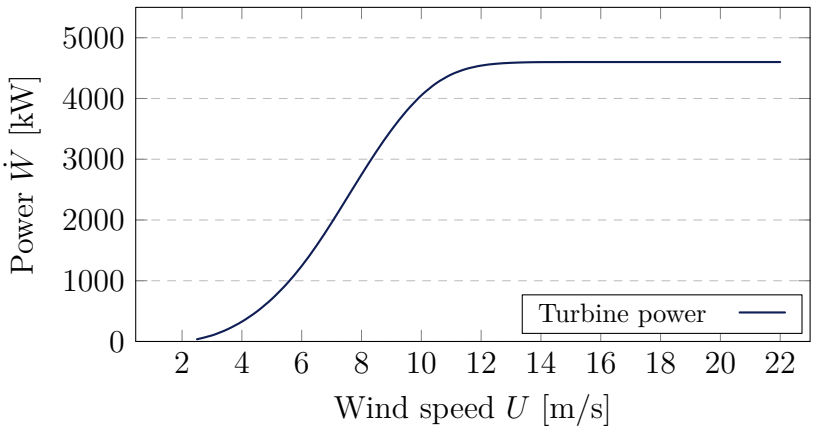


Figure 5.19: Power curve for turbine model installed in Mörknässkogen wind farm

(iii) Apply the power associated with the wind speed measured in each record to calculate the annual energy consumption AEP with the expression:

$$AEP = \frac{365}{d_m} \sum_{r=1}^{n} \left(\dot{W}_r \frac{\Delta t_r}{60} \right), \tag{5.3}$$

where $d_m = 684$ is the total time of the measurement campaign in days, $\dot{W}_r$ is the turbine power for record r, Δt_r is the measurement time interval between records equal to 10 minutes, and n the total number of records.

(iv) Calculate the theoretical maximum annual energy production $AEP_{\max}$ considering an availability factor of 100% and the nominal power of the installed turbine model of $\dot{W}_{\text{nom}} = 4.6$ MW, with the expression:

$$AEP_{\max} = \dot{W}_{\text{nom}}(365 \times 24). \tag{5.4}$$

(v) Calculate the capacity factor CF, which measures the efficiency with which the wind turbine is able to convert the energy generation potential into energy actually produced over a certain period of time, given by:

$$CF = \frac{AEP}{AEP_{\max}}. \tag{5.5}$$

(vi) Finally, estimate the number of full loading hours flh with the equation:

$$flh = \frac{AEP}{\dot{W}_{\text{nom}}}. \tag{5.6}$$

Table 5.5 presents the synthesised results of this procedure, that was applied to the four 4.6 MW wind turbines installed in Mörknässkogen wind farm. For this model of wind turbine, the respective rated power corresponds to a theoretical maximum annual energy production equal to $AEP_{\max} = 40296$ MWh/year.

Table 5.5: Annual energy production estimate for wind turbines installed in Mörknässkogen wind farm, with $h_{\text{hub}} = 166.0$ m AGL

Turbine	Δz_{hub} [m]	$\overline{U}_{\text{hub}}$ [m/s]	$\overline{I}_{\text{hub}}$ [%]	A_{Weibull} [m/s]	k_{Weibull}	AEP [MWh/year]	flh [h]	CF [%]
WT_{01}	33.2	7.715	8.147	8.86	2.64	19613.6	4264	48.67
WT_{02}	34.7	7.740	8.297	8.88	2.59	19628.9	4267	48.71
WT_{03}	35.0	7.769	8.558	8.91	2.56	19667.6	4276	48.81
WT_{04}	40.0	8.135	7.638	9.32	2.47	20674.6	4494	51.31
Wind farm						**79584.7**	**4325**	**49.38**

5.5.6 Extrapolated analysis within a grid of points

For a wider analysis of the region surrounding Mörknässkogen wind farm, an additional study was performed in which the *Synthesis* algorithm was applied to transport the time series measured at 120 m AGL on mast i690 to a grid of points, in this case defined to capture an area of 6000×6000 m^2 with its center located at the center of the computational domain. All these points were extrapolated to the turbine hub height of 166 m AGL.

Considering the high computational resources associated with the transport of these time series, it was decided to divide this area of analysis into a relatively small grid with 15×15 cells in x and y directions, corresponding to a horizontal resolution of 400 m, which is already an acceptable distance to evaluate the energy distribution in the region with reasonable precision in qualitative terms.

The results of this study are presented in Figure 5.20 in terms of the average wind speed at hub height of 166 m AGL that a wind turbine of the same model would have if it had been installed in the corresponding geographic coordinates.

Graphical analysis demonstrates that the four wind turbines were installed following a similar energy level, which for this turbine model is coincident with the established limit for average wind speed at hub height of 7.5 m/s according to *IEC 61400-1:2019 (4th Edition)* [20] for wind class III A turbines, based on long term correlated measurement data.

Results are also consistent with the analysis that was performed in Section 3.4.3, which indicates a highest amount of energy available for 210° winds (see Figure 3.7). For this reason, the layout chosen to position the wind turbines consisted of orienting them almost perpendicular to 210° direction, so that they could face the prevailing wind, minimizing the presence of wake effects from other neighbouring wind turbines.

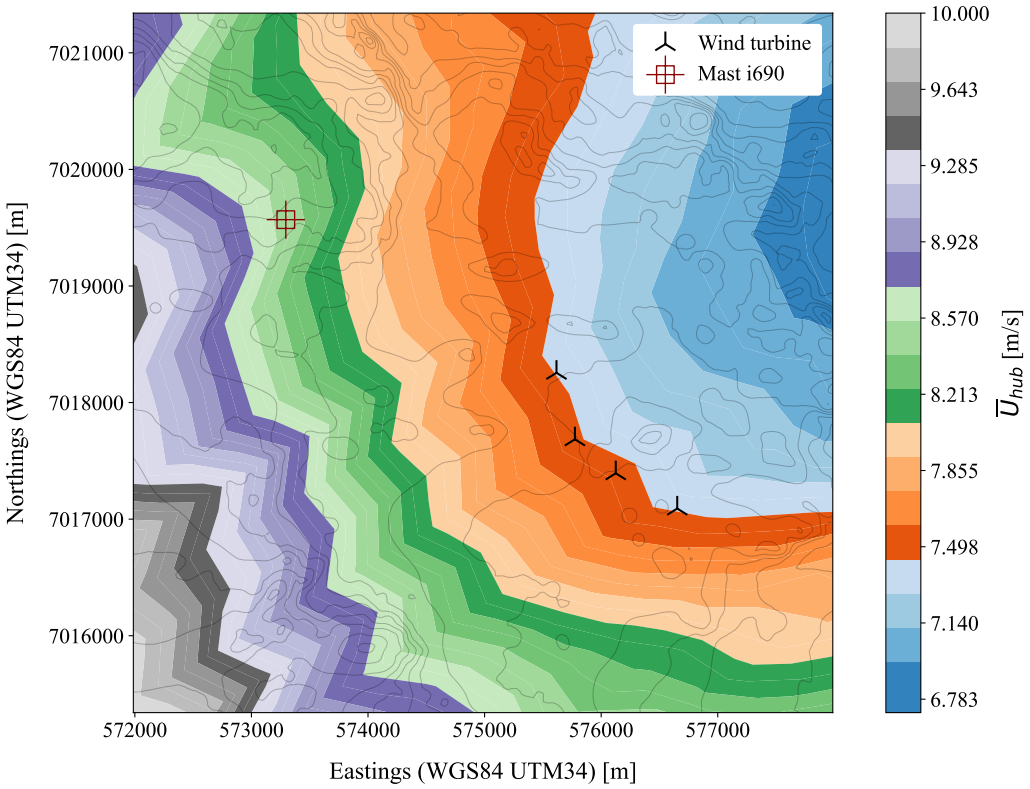


Figure 5.20: Estimated wind speed at 166 m AGL $\overline{U}_{\text{hub}}$ in a grid of points (Horizontal grid resolution of 400 m)

5.5.7 Site verification for adverse wind conditions

Site verification parameters were calculated for turbine hub height at 166 m AGL, to assess IEC quantities that may have a negative impact on wind turbine operation.

In this work, wake-added turbulence[7] from neighbouring wind turbines was not taken into account during normal operation, since this effect would only occur for winds coming from ≈ 135° or ≈ 315°, whose frequency of occurrence and energy contribution are relatively low and was therefore disregarded (see Figures 3.6 and 3.7). According to the standard *IEC 61400-1:2019 (4th Edition)* [20], these effects are considered for fatigue analysis and are evaluated based on the effective turbulence intensity I_{eff}.

The complex-terrain safety factor[8] was also disregarded in this analysis, and thus no complex-terrain correction was applied. However, in nearly flat terrain topography such as the Morknaskogen wind farm, the complex-terrain safety factor tends to be equal to or very close to 1. This is due to the fact that ABL flows in such areas are more predictable, with fewer abrupt changes in speed, direction, or turbulence caused by geographic features. As a result, the additional mechanical loads on wind turbines are minimal, and the local wind conditions are typically consistent with forecasts.

To assess whether the limits established by *IEC 61400-1:2019 (4th Edition)* [20] are met for wind turbines with class III A (designates the category for higher turbulence characteristics), like those installed in Morknaskogen wind farm, the conditions that were considered are:

(i) Frequency-averaged wind speed at hub height below 7.5 m/s;

(ii) 360° average wind shear factor $\overline{\alpha}$ between 0 and 0.2;

(iii) 360° average flow inclination $\overline{\gamma}$ for 30° sectors between −8° and +8°;

(iv) Representative turbulence intensity I_{rep} at 15 m/s below 17.97%.

According to *IEC 61400-1:2019 (4th Edition)* [20], the representative value of the turbulence standard deviation σ_1 (see Equation 2.25), shall be greater or equal to the site value of the estimated 90th percentile of the sites turbulence standard deviation at all values U_{hub} between the wind speed $0.2U_{\text{ref}}$ and $0.4U_{\text{ref}}$, thus establishing that the representative turbulent intensity I_{rep} is calculated with the expression:

$$I_{\text{rep}} = I_{\text{amb}} + 1.28\sigma_{I_{\text{amb}}}, \tag{5.7}$$

where I_{amb} is the ambient turbulent intensity and $\sigma_{I_{\text{amb}}}$ its standard deviation.

[7] In wind energy, wake-added turbulence refers to an increase in turbulence generated downstream of the wind turbine when the airflow passes through the rotor, causing a reduction in wind speed and more chaotic flow conditions, which can negatively impact downstream turbines by decreasing their efficiency and subjecting them to higher mechanical stress. The added turbulence also redistributes kinetic energy, resulting in slower wind recovery and increased structural loads, which can reduce the lifespan of components in downstream turbines.

[8] The complex-terrain safety factor is a correction applied in the design and operation of wind farms located in areas with complex orography, such as hills, mountains, or valleys. These terrains introduce variability in wind speed, direction, and turbulence patterns, which can impose additional mechanical loads on wind turbines. In order to account for these effects, this safety factor ensures that turbines operate reliably and are not subjected to unexpected loads that could reduce their lifespan or performance.

Table 5.6 summarises the results obtained from the analysis performed to SET_{214}, in terms of harmful quantities previously indicated, and also the overall average ambient turbulent intensity at hub height $\overline{I}_{amb}$, the non-dimensional speed-up ratio ΔS and Weibull parameters.

Table 5.6: Overall site verification for adverse wind conditions considering the results obtained for SET_{214}. In gray colour, the same results were added to SET_{102} to compare the adverse wind conditions with and without considering the forest. $\Delta S = \widetilde{U}_h^t / \widetilde{U}_h^{\oplus}$ is the non-dimensional speed-up ratio. Results that do not meet the standard requirements on SET_{214} identified with ( ). For additional geographic information on mast i690 and installed wind turbines, see Table 3.1.

Turbine		$\overline{U}_{hub}$ [m/s]	$\overline{I}_{amb}$ [%]	$\overline{I}_{rep}$ [%]	$A_{Weibull}$ [m/s]	$k_{Weibull}$	ΔS	$\overline{\alpha}$	$\overline{\gamma}$ [°]
WT_{01}	SET_{214}	7.72	8.15	15.07	8.86	2.64	1.126	0.436	0.326
	SET_{102}	7.14	8.92	16.31	8.20	2.77	1.027	0.208	0.092
WT_{02}	SET_{214}	7.74	8.30	15.32	8.88	2.59	1.125	0.430	0.313
	SET_{102}	7.14	8.93	16.34	8.20	2.76	1.023	0.208	0.088
WT_{03}	SET_{214}	7.77	8.56	15.71	8.91	2.56	1.130	0.442	0.310
	SET_{102}	7.12	8.99	16.44	8.18	2.76	1.019	0.212	0.081
WT_{04}	SET_{214}	8.14	7.64	14.40	9.32	2.47	1.172	0.424	0.326
	SET_{102}	7.16	8.93	16.32	8.22	2.76	1.025	0.205	0.095
IEC limit		7.5		17.97				[0,0.2]	[−8°,8°]

Results obtained from Table 5.6 revealed a high wind shear across the four wind turbines, exceeding the established limit of $\alpha_{limit} = 0.2$ by more than two times. Through the analysis of the values obtained individually for each wind direction in each turbine, it was found that overall wind shear levels vary between a minimum of $\alpha_{min} = 0.3715$ at WT_{01} (30° wind) and a maximum of $\alpha_{max} = 0.4780$ at WT_{03} (180° wind). This result was expected due to the existing forest in the region which covers vast tracts of land all around the site, that causes high wind shear with little variation between wind sectors and high values all around the wind rose.

Besides the high wind shear, the site presents a slightly high wind speed at hub height, which only exceeds the maximum established value between the smallest deviation of +2.79% at WT_{01} and the largest deviation of +7.81% at WT_{04}.

The representative turbulent intensity $\overline{I}_{rep}$ was also evaluated as a function of different velocity ranges, the result of which is shown in Table 5.7. The results demonstrated that the representative turbulence exceeds the limits imposed by *IEC 61400-1:2019 (4th Edition)* [20] for some average results obtained for wind speeds equal to or greater than 17 m/s. However, this violation should not be of concern since the respective frequency of occurrence of these winds is very low and can be disregarded.

Table 5.7: Verification of turbulent intensity by wind speed ranges. Results that do not meet the standard requirements on SET_{214} identified with ( ).

	WT_{01}		WT_{02}		WT_{03}		WT_{04}		
U [m/s]	f [%]	I_{rep} [%]	f [%]	I_{rep} [%]	f [%]	I_{rep} [%]	f [%]	I_{rep} [%]	$I_{IEC\ limit}$ [%]
3.0	-	-	-	-	-	-	-	-	41.87
5.0	6.36	16.31	6.76	16.57	7.08	16.83	5.88	15.80	29.92
7.0	26.42	14.65	25.87	14.81	25.61	15.10	22.53	13.60	24.80
9.0	28.17	14.25	27.78	14.36	27.40	14.63	25.86	12.78	21.96
11.0	22.76	14.64	22.40	14.74	22.04	15.01	22.05	12.66	20.15
13.0	11.62	15.51	11.87	15.43	11.91	15.51	14.13	12.67	18.89
15.0	**3.67**	**16.86**	**4.20**	**16.63**	**4.66**	**16.37**	**6.78**	**13.07**	**17.97**
17.0	0.83	17.74	0.91	17.66	1.06	17.46	2.10	13.93	17.27
19.0	0.12	19.58	0.17	19.08	0.21	18.65	0.54	14.68	16.72
21.0	0.03	21.04	0.03	21.01	0.03	20.98	0.10	16.01	16.27
23.0	-	-	-	-	-	-	0.01	16.24	15.90
25.0	-	-	-	-	-	-	-	-	15.58
27.0	-	-	-	-	-	-	-	-	15.32
29.0	-	-	-	-	-	-	-	-	15.09
31.0	-	-	-	-	-	-	-	-	14.89
33.0	-	-	-	-	-	-	-	-	14.72

Chapter 6

Simulations of non-neutrally stratified ABL flows combined with canopy model

6.1 Introduction

In this Chapter we present the partial development of the study that was performed in terms of simulations of non-neutrally stratified ABL flows over Mörknässkogen wind farm combined with the forest model implemented through the `kEpsilonLopesdaCosta` turbulence model, as was applied in the simulations performed in Chapter 5.

As previously mentioned in Chapter 1, in an advanced stage of this work it was concluded that the amount of work required to develop all necessary tools to perform this simulations would not be possible to accomplish in the available schedule, and for this reason only what has been developed so far will be presented.

In terms of procedure, the contribution of thermal effects and the Coriolis force was introduced into the CFD simulations using *buoyantBoussinesqSimpleFoam* solver along with specific source terms that were added to the `fvOptions` file.

Unlike the *simpleFoam* used so far, the solver *buoyantBoussinesqSimpleFoam* is suited for steady-state simulations of incompressible and turbulent flows that account for buoyancy effects, making use of the Boussinesq approximation. This method is a simplification technique used in fluid dynamics to handle problems where density variations are small but still influence the fluid motion due to buoyancy. In these cases, if the density variation is not large, one may treat the density as constant in the unsteady and convection terms, and treat it as variable only in the gravitational term [2]. If one includes the effect of the body force on the mean density in the pressure term as described in Equation 2.35, the remaining term can be expressed as:

$$(\rho - \rho_0)g_i = -\rho_0 g_i(T - T_0), \tag{6.1}$$

where β is the coefficient of volumetric expansion, and T_0 the cold wall temperature. This approximation introduces errors of $\approx 1\%$ if the temperature difference ΔT is below certain values, such as $\Delta T < 2$°C for water and $\Delta T < 15$°C for air. However, the error can be substantially larger if the temperature difference is high, so caution is needed when applying it.

As for the Coriolis force and other necessary source terms to account for thermal contributions, such as the specification of Monin-Obukhov lengths for different atmospheric stability regimes, their application was tested following a precursor study performed in a type of application similar to our study [60], namely the OpenFOAM tutorials `atmFlatTerrain` and `atmForestStability`. These tutorials, along with the respective article that was found about its implementation, were the basis and source of research used in this part of the work.

6.2 Compiling `mykEpsilonLopesdaCosta` turbulence model

The first challenge that was encountered when running the solver *buoyantBoussinesqSimpleFoam* was the absence of the internal variable `GbyNu` in the source code of `kEpsilonLopesdaCosta` turbulence model, since it is an input parameter that is needed for *buoyantBoussinesqSimpleFoam*. The field `GbyNu` is present in the source code of other turbulence models such as $k-\varepsilon$ or $k-\omega$[1]. With this, it was verified that each turbulence model has its own specific variables, so that turbulent fields are created by the model itself, and not from the solver.

6.2.1 Compilation setup

To overcome this, the procedure involved compiling a modified version of `kEpsilonLopesdaCosta` turbulence model, latter called `mykEpsilonLopesdaCosta`, whose source code was changed to include the calculation of the `GbyNu` field.

This procedure is relatively complex due to the nature of OpenFOAM's dependency structure, but it was successfully achieved with the following sequence of setup operations:

1) Copy the folder containing the `kEpsilonLopesdaCosta` source code, files and classes to a user-defined directory, in this case defined to home directory `$WM_PROJECT_USER_DIR`. This procedure is achieved with the commands shown in Figure 6.1.

```
# Copy kEpsilonLopesdaCosta source code to home directory $WM_PROJECT_USER_DIR

foam
cp -r --parents src/atmosphericModels/turbulenceModels/RAS/kEpsilonLopesdaCosta
$WM_PROJECT_USER_DIR

cd $WM_PROJECT_USER_DIR/src/atmosphericModels/turbulenceModels/RAS
mv kEpsilonLopesdaCosta mykEpsilonLopesdaCosta
cd mykEpsilonLopesdaCosta
mv kEpsilonLopesdaCosta.C mykEpsilonLopesdaCosta.C
mv kEpsilonLopesdaCosta.H mykEpsilonLopesdaCosta.H
sed -i s/kEpsilonLopesdaCosta/mykEpsilonLopesdaCosta/g mykEpsilonLopesdaCosta.*
```

Figure 6.1: Shell commands to copy the folder containing the `kEpsilonLopesdaCosta` source code, files and classes to a user-defined directory

[1] But not in `kEpsilonLopesdaCosta` turbulence model due to a slightly different implementation of the ε equation.

2) Copy the file with the macros that provide information to the compiler in terms of the instances of turbulence models to compile, with the commands listed in Figure 6.2.

```
# Copy compilation MACROS to home directory $WM_PROJECT_USER_DIR

foam
cp --parents src/atmosphericModels/turbulenceModels/
atmosphericTurbulentTransportModels.C $WM_PROJECT_USER_DIR

cp -r --parents src/atmosphericModels/Make $WM_PROJECT_USER_DIR
cd $WM_PROJECT_USER_DIR/src/atmosphericModels/turbulenceModels
mv atmosphericTurbulentTransportModels.C myatmosphericTurbulentTransportModels.C
```

Figure 6.2: Shell commands to copy the compilation macros to a user-defined directory

3) Open myTurbulentTransportModels.C file and modify the turbulence model to be compiled for mykEpsilonLopesdaCosta, and delete or comment the other commands as they are not necessary. The file should look like the one shown in Figure 6.3.

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | www.openfoam.com
     \\/     M anipulation  |
-------------------------------------------------------------------------------
    Copyright (C) 2018 OpenFOAM Foundation
    Copyright (C) 2021 OpenCFD Ltd.
-------------------------------------------------------------------------------

\*---------------------------------------------------------------------------*/

#include "turbulentTransportModels.H"

// ------------------------------------------------------------------------ //
// RAS models
// ------------------------------------------------------------------------ //

#include "mykEpsilonLopesdaCosta.H"
makeRASModel(mykEpsilonLopesdaCosta);
```

Figure 6.3: Compilation macro file myTurbulentTransportModels.C set to compile mykEpsilonLopesdaCosta turbulence model.

4) Go to the $WM_PROJECT_USER_DIR/src/atmosphericModels directory and modify the Make/files and Make/options files. For Make/files file, the content must be the same as shown in Figure 6.4, and for Make/options the the content must be the same as shown in Figure 6.5, in which only the line "-I$(LIB_SRC)/atmosphericModels/lnInclude \" was added to the original version.

```
turbulenceModels/myatmosphericTurbulentTransportModels.C
LIB = $(FOAM_USER_LIBBIN)/mylibatmosphericModels
```

Figure 6.4: Macro file `Make/files` set to compile `mykEpsilonLopesdaCosta` turbulence model.

```
EXE_INC = \
    -I$(LIB_SRC)/finiteVolume/lnInclude \
    -I$(LIB_SRC)/meshTools/lnInclude \
    -I$(LIB_SRC)/surfMesh/lnInclude \
    -I$(LIB_SRC)/transportModels \
    -I$(LIB_SRC)/atmosphericModels/lnInclude \
    -I$(LIB_SRC)/transportModels/compressible/lnInclude \
    -I$(LIB_SRC)/transportModels/incompressible/lnInclude \
    -I$(LIB_SRC)/thermophysicalModels/basic/lnInclude \
    -I$(LIB_SRC)/thermophysicalModels/specie/lnInclude \
    -I$(LIB_SRC)/thermophysicalModels/solidThermo/lnInclude \
    -I$(LIB_SRC)/TurbulenceModels/turbulenceModels/lnInclude \
    -I$(LIB_SRC)/TurbulenceModels/compressible/lnInclude \
    -I$(LIB_SRC)/TurbulenceModels/incompressible/lnInclude \
    -I$(LIB_SRC)/fvOptions/lnInclude

LIB_LIBS = \
    -lfiniteVolume \
    -lfvOptions \
    -lmeshTools \
    -lsurfMesh \
    -lfluidThermophysicalModels \
    -lsolidThermo \
    -lturbulenceModels \
    -lincompressibleTurbulenceModels \
    -lcompressibleTurbulenceModels
```

Figure 6.5: Macro file `Make/options` set to compile `mykEpsilonLopesdaCosta` turbulence model.

5) Modify the source code of the file `mykEpsilonLopesdaCosta.C` to include the field `GbyNu`. Due to the nature of *buoyantBoussinesqSimpleFoam*, `GbyNu` must be programmed as an internal volumetric scalar field named `mykEpsilonLopesdaCosta:GbyNu`. This was extracted from the standard $k-\varepsilon$ turbulence model, and adapted according to the input structure required by *buoyantBoussinesqSimpleFoam*, as shown in Figure 6.6.

```
tmp<volTensorField> tgradU = fvc::grad(U);
const volScalarField::Internal GbyNu
(
    IOobject::scopedName(this->type(), "GbyNu"),
    tgradU().v() && devTwoSymm(tgradU().v())
);
const volScalarField::Internal G(this->GName(), nut()*GbyNu);
tgradU.clear();
```

Figure 6.6: Added code to program the `GbyNu` field in the source code of `mykEpsilonLopesdaCosta.C`

6) Finally, in order to compile mykEpsilonLopesdaCosta turbulence model, it is necessary to create an lnInclude directory for the available turbulence models using wmakeLnInclude command, and compile with wmake. The commands are presented in Figure 6.7.

```
# Compile mykEpsilonLopesdaCosta turbulence modelo with wmake

wmakeLnInclude -u ./turbulenceModels
wmake
```

Figure 6.7: Shell commands to copy the folder containing the Shell commands to compile mykEpsilonLopesdaCosta source code

This new tool was tested in several simulations that account for thermal effects with *buoyantBoussinesqSimpleFoam* solver, and was considered validated in terms of its implementation in OpenFOAM, although the simulations did not converge. However, it is considered that the phenomenon is related to boundary conditions and instability in temperature convergence, and not to the recently compiled mykEpsilonLopesdaCosta turbulence model.

6.2.2 Additional analysis of GbyNu field

As previously mentioned, the GbyNu field is calculated in OpenFOAM using the expression "tgradU() && dev(twoSymm(tgradU()))" which was extracted from the standard $k-\varepsilon$ turbulence model. To better understand the importance of this field for *buoyantBoussinesqSimpleFoam*, the expression was mathematically analysed, and the following aspects were verified:

(i) tgradU() is a function that computes the gradient of the velocity field ∇U and returns a tensor representing how the velocity field U varies in space.

(ii) && is an operator used to perform the double inner product.

(iii) twoSymm(tgradU()) is a function that applies the symmetrization operation to the tensor obtained from tgradU(), converting the gradient tensor into its symmetric part. Therefore, tgradU() $= 2\mathcal{D}$ being $\mathcal{D}$ the strain rate tensor (see Equation 2.37).

(iv) dev() is a function that calculates the deviatoric part of the tensor. It subtracts the isotropic part (mean of the diagonal elements) from the symmetric part of the tensor. Thus, dev(twoSymm(tgradU())) is equal to $2\mathcal{D} - {}^{2}/_{3}(\nabla \cdot \mathbf{U})\mathbf{I}$.

(v) Hence, GbyNu is the velocity gradient by the viscous stress tensor ($\mathcal{G}$) divided by μ, since $\tau = 2\mu\mathcal{D} - {}^{2}/_{3}\mu(\nabla\cdot\mathbf{U})\mathbf{I} \equiv 2\mu\mathtt{dev}(\mathcal{D})$ (see Equation 2.40). $\mathcal{G}$ appears in the energy equation representing mechanical power dissipated as heat, and in the TKE equation as turbulence generation.

In the ε equation, OpenFOAM uses GbyNu so that C_1, ρ, G, ε/k is used by the standard $k-\varepsilon$ turbulence model with C1_*alpha()*rho()*GbyNu*Cmu_*k_() (line 266 of kEpsilon.C).

In the k equation, the source term is simply $\rho\mathcal{G}$ and its programmed in the code with `alpha()*rho()*G` (line 287 of `kEpsilon.C`).

However, since `kEpsilonLopesdaCosta` uses $\mathcal{G}$ in the ε equation with `C1_*alpha()*rho()*G*epsilon_()/k_()` (line 437 of `kEpsilonLopesdaCosta.C`), it indicates that whoever programmed `kEpsilonLopesdaCosta` turbulence model did not see the need to define `GbyNu`.

6.3 Analysis of the `atmFlatTerrain` OpenFOAM tutorial

This OpenFOAM tutorial refers to a study that was developed on non-neutrally stratified ABL flows over a flat terrain with different types of turbulence models, where the $k-\varepsilon$, kL and $k-\omega$ SST models were chosen.

In terms of the article that was later published about the study [60], it is a document that describes Enercon's E-Wind solver which uses OpenFOAM to predict resource and site assessment conditions over real wind farm sites, that includes Coriolis, forest modelling and stability via buoyancy effects on turbulence (but not momentum). Humidity and mesoscale models are both disregarded.

The tutorial is divided into two parts, the first referring to a precursor simulation, and the second to a successor simulation, whose simulation strategy underwent significant changes between the two, and which will be discussed and analysed in the next sections. In terms of computational domain and mesh structure, the two simulations have a mesh of $20 \times 20 \times 6$ km^3, with the precursor one having a computational domain with a cubic shape, while the successor has a cylindrical shape, like shown in Figure 6.8.

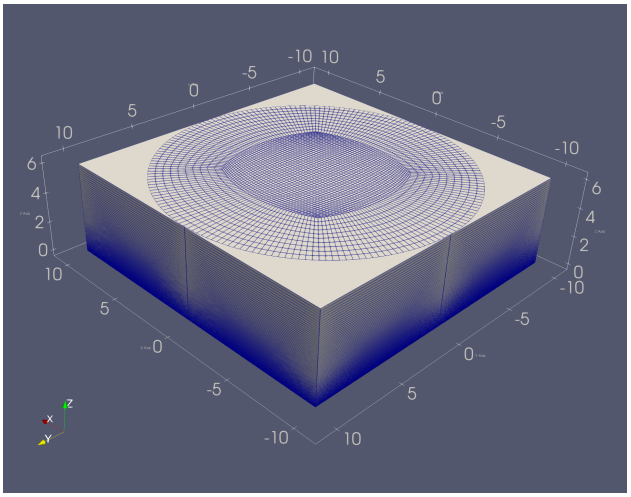
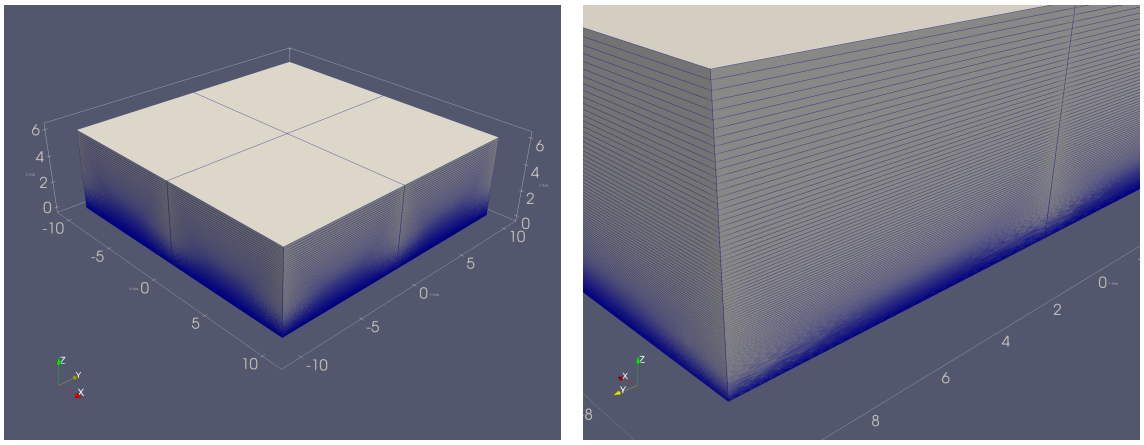


Figure 6.8: Representation of the computational domains of the precursor and successor simulations overlapped.

6.3.1 Precursor simulation

The precursor simulation is a cyclic simulation in a simple $2 \times 2 \times 160$ mesh, forming a hexaedron, as illustrated in Figure 6.9.



(a) (b)

Figure 6.9: Representation of the precursor simulation numerical mesh of the `atmFlatTerrain` tutorial. (a) Isometric domain overview. (b) Zoomed isometric view focusing on vertical mesh refinement.

In terms of boundary conditions, the simulations were defined following a strategy of applying conditions of `cyclic` type to the boundaries *inlet*, *outlet*, *left* and *right*, which is used to model periodic symmetry, where opposite faces of the mesh are treated as if they are connected, creating a continuous solution without discontinuities. The *top* boundary is defined with the type `slip` for U, k and ε, and with type `fixedValue` or `calculated` for the remaining quantities. The *bottom* boundary, which represents the terrain, is defined with more sophisticated boundary condition types. For more details, see the tutorial `0` folder.

The contribution of the Coriolis force and thermal effects associated with the Monin-Obukhov length to different atmospheric stability regimes is configured using source terms that are inserted into the `fvOptions` file. These source terms were analysed to better understand the context of their application, whose conclusions are discussed in the following sections.

6.3.1.1 `atmCoriolisUSource1` source term

This source term is used apply corrections to incorporate the horizontal and vertical components of the Coriolis force[2] for which the rotating frame is Earth. The Coriolis force is typically evaluated based on the respective Coriolis frequency f_c, which can be expressed with the equation:

$$f_c = 2\Omega sin(\varphi), \tag{6.2}$$

[2]The Coriolis force is an inertial or fictitious force that acts on objects that are in motion within a frame of reference that rotates with respect to an inertial frame. In the context of ABL flows, the Coriolis effect is primarily associated with Earth's rotation. Because Earth spins, Earth-bound observers need to account for the Coriolis force to correctly analyse the motion of objects. Earth completes one rotation per day, so for motions of everyday objects the Coriolis force is usually quite small compared with other forces; its effects generally become noticeable only for motions occurring over large distances and long periods of time, such as large-scale movement of air in the atmosphere or water in the ocean. Such motions are constrained by the surface of Earth, so only the horizontal component of the Coriolis force is generally important.

where φ is the latitude where the motion occurs, and Ω the angular velocity, which in the case of Earth's rotation around its axis yields $\Omega = 7.2921 \times 10^{-5}$ rad/s corresponding to a rotation period $T = 23.9344$ h (23 hours, 56 minutes, 4 seconds).

This source term was added to the `fvOptions` settings in the Mörknässkogen wind farm simulations, having been defined in a more intuitive way using the Earth rotation period T and the site latitude of $\varphi \approx 63.2861°$. The performed adaptations are presented in Figure 6.10. Note that in the tutorial, the `Omega` parameter does not refer to Earth's rotation velocity Ω, but rather to $\Omega sin(\varphi)$ for a latitude of $\varphi = 51°$ and a rotation period approximated to $T = 24$ h.

`atmFlatTerrain` tutorial

```
atmCoriolisUSource1
{
    type            atmCoriolisUSource;
    selectionMode   all;

    Omega           ( 0 0 5.65156e-05 );
}
```

(a)

Mörknässkogen wind farm

```
atmCoriolisUSource1
{
    type            atmCoriolisUSource;
    selectionMode   all;
    latitude                    63.2861005;
    planetaryRotationPeriod     23.9344694;
}
```

(b)

Figure 6.10: Adaptations performed to add the source term `atmCoriolisUSource1` to the `fvOptions` file in the simulations of non-neutrally stratified ABL flows over Mörknässkogen wind farm. (a) Code content of `atmFlatTerrain` tutorial. (b) Code content of Mörknässkogen wind farm simulations.

6.3.1.2 `pressureGradient` source term

This source term is used to sustain an Earth-bound 17.5 m/s geostrophic wind by imposing a pressure gradient ∇p that is calculated with the expression:

$$a_g = \frac{1}{f_c}(\hat{k} \times \nabla p), \tag{6.3}$$

where a_g is the the acceleration applied to the fluid due to the pressure gradient, and $\hat{k}$ a unit vector in the z direction. However, the parameter that is used in the momentum equations and that is specified in `fvOptions` configuration is not ∇p but $-(1/\rho)\nabla p$. Thus, Equation 6.3 is modified to:

$$-\frac{1}{\rho}\nabla p = -f_c(a_g \times \hat{k}). \tag{6.4}$$

The case in the tutorial is for $\varphi = 51°$ with a geostrophic wind $(17.5, 0, 0)$ which yields a Coriolis frequency of $f_c = 1.133 \times 10^{-4}$ Hz, resulting in the acceleration value of $a_g = 1.978 \times 10^{-3}$ m/s^2, which is consistent with the value configured in the tutorial. In the case of Mörknässkogen wind farm, the homologous value obtained is $a_g = 2.280 \times 10^{-3}$ m/s^2. The performed adaptations are presented in Figure 6.11.

`atmFlatTerrain` tutorial

```
pressureGradient
{
  type       vectorSemiImplicitSource;
  volumeMode     specific;
  selectionMode  all;
  sources
  {
    U            ((0 0.00197805 0) 0);
  }
}
```

(a)

Mörknässkogen wind farm

```
pressureGradient
{
  type       vectorSemiImplicitSource;
  volumeMode     specific;
  selectionMode  all;
  sources
  {
    U            ((0 0.00227982 0) 0);
  }
}
```

(b)

Figure 6.11: Adaptations performed to add the source term `pressureGradient` to the `fvOptions` file in the simulations of non-neutrally stratified ABL flows over Mörknässkogen wind farm. (b) Code content of Mörknässkogen wind farm simulations.

6.3.1.3 `atmAmbientTurbSource1` source term

This source term prevents k and ε (or ω if applicable) dropping below ambient values, increasing numerical stability for stable ABL flows as a result. In this source term, no adaptations were performed to the original settings shown in Figure 6.12.

```
atmAmbientTurbSource1
{
    type            atmAmbientTurbSource;
    selectionMode   all;
    kAmb            0.0001;
    epsilonAmb      7.208e-08;
}
```

Figure 6.12: `atmAmbientTurbSource1` source term added to the `fvOptions` file in the simulations of non-neutrally stratified ABL flows over Mörknässkogen wind farm

6.3.1.4 `atmBuoyancyTurbSource1` source term

This source term is used to incorporate buoyancy effects on k and ε equations, that are taken into account through the parameter B [60], which is given by:

$$B = \beta g_i \frac{\nu_t}{\sigma_\theta} \frac{\partial \overline{\theta}}{\partial x_i}, \tag{6.5}$$

where $\beta = 0.0033\ \mathrm{K}^{-1}$ is the thermal expansion coefficient referring to an ambient temperature of $T = 300.0$ K, and σ_θ the turbulent Prandtl number, which can be evaluated as:

$$\sigma_\theta = \begin{cases} 0.74 & , \mathrm{Ri}_g \geq 0 \\ 0.74(1 - 15\,\mathrm{Ri}_G)^{-1/4} & , \mathrm{Ri}_g < 0 \end{cases}, \tag{6.6}$$

where $\mathrm{Ri}_g = -B/P_k$ is the gradient Richardson number. Ri_G is the modified gradient Richardson number introduced to avoid singularities for small turbulence productions P_k, that is given by:

$$\mathrm{Ri}_G = -\frac{B}{P_k + \left|\frac{\alpha_\theta}{\sigma_\theta} B\right|}, \tag{6.7}$$

where α_θ is a coefficient set to achieve equilibrium between turbulent kinetic energy production by shear P_k and dissipation in the stable case, and between buoyancy production B and dissipation ε in the unstable case [60]:

$$\alpha_\theta = \begin{cases} 1 - \dfrac{L}{L_{\max}} & , \mathrm{Ri}_g \geq 0 \\ 1 - \left(1 + \dfrac{C_{\varepsilon 1}}{C_{\varepsilon 2} - C_{\varepsilon 1}}\right) \dfrac{L}{L_{\max}} & , \mathrm{Ri}_g < 0 \end{cases}. \tag{6.8}$$

With this, it is possible to establish a simple relationship between the maximum turbulent length scales $L_{\max}$ and the stability class to be used based on Monin-Obukhov length L (see Table 6.1), instead of estimating $L_{\max}$ from the location of the geometric center of the turbulent kinetic energy along the vertical direction, as proposed by other authors [61].

Table 6.1: Tabular relation between Monin-Obukhov length L and maximum turbulent length scale $L_{\max}$ [60].

Monin-Obukhov Length L [m]	-50	-60	-80	-130	∞	660	350	230	160	110	80	70
Max. length scale $L_{\max}$ [m]	200	160	120	81	41	35	30	25	21	16	13	10

In this source term, no adaptations were performed to the original settings presented in Figure 6.13.

```
atmBuoyancyTurbSource1
{
    type            atmBuoyancyTurbSource;
    selectionMode   all;
    rho             rho;
    Lmax            41.8;
    beta            0.0033;
}
```

Figure 6.13: `atmBuoyancyTurbSource1` source term added to the `fvOptions` file in the simulations of non-neutrally stratified ABL flows over Mörknässkogen wind farm. Values above are for neutral stratification. For other regimes, values will be modified as described in Section 6.5.

6.3.1.5 `atmLengthScaleTurbSource1` source term

This source term is applied to the ε equation to correct mixing-length estimations, adjusting the contribution of large turbulent structures to the overall flow dynamics. This source term is defined by the maximum turbulent length scale $L_{\max}$ corresponding to the stability regime to be simulated, and also by the parameter n which defines the exponent in the scaling law that governs the distribution or decay of TKE across different length scales[3]. It influences how energy cascades from large turbulent eddies to smaller ones. In this source term, no adaptations were performed to the original settings presented in Figure 6.14.

```
atmLengthScaleTurbSource1
{
    type            atmLengthScaleTurbSource;
    selectionMode   all;
    Lmax            41.8;
    n               3;
}
```

Figure 6.14: `atmLengthScaleTurbSource1` source term added to the `fvOptions` file in the simulations of non-neutrally stratified ABL flows over Mörknässkogen wind farm. Values above are for neutral stratification. For other regimes, values will be modified as described in Section 6.5.

6.3.1.6 Other source terms to incorporate forest

In addition to the source terms described in the previous sections, the tutorial also has other additional source terms for incorporating the forest into the simulations, which are shown in Figure 6.15. However, these source terms were disregarded in the simulations of non-neutrally stratified ABL flows over Mörknässkogen wind farm since the goal is to use `kEpsilonLopesdaCosta` turbulence model, which already includes a canopy model that produced good results in this work.

6.3.1.7 Final remarks

After running the tutorial and analysing the results, the following points were verified:

(i) On inspection of results, it was observed the Coriolis source produce a clockwise Ekman spiral aligning with the geostrophic wind at 1500 m.

(ii) Setups differ in the turbulence model used. Turbulence models $k-\varepsilon$ and $k-\omega$ SST use the same `fvOptions`. However, kL only uses `pressureGradient` and `atmCoriolisUSource1` source terms.

(iii) At the *ground* patch, the boundary condition for temperature is defined to `atmTurbulentHeatFluxTemperature` where a small uniform turbulent heat flux of $q_t'' = 0.0001$ W/m^2 is imposed.

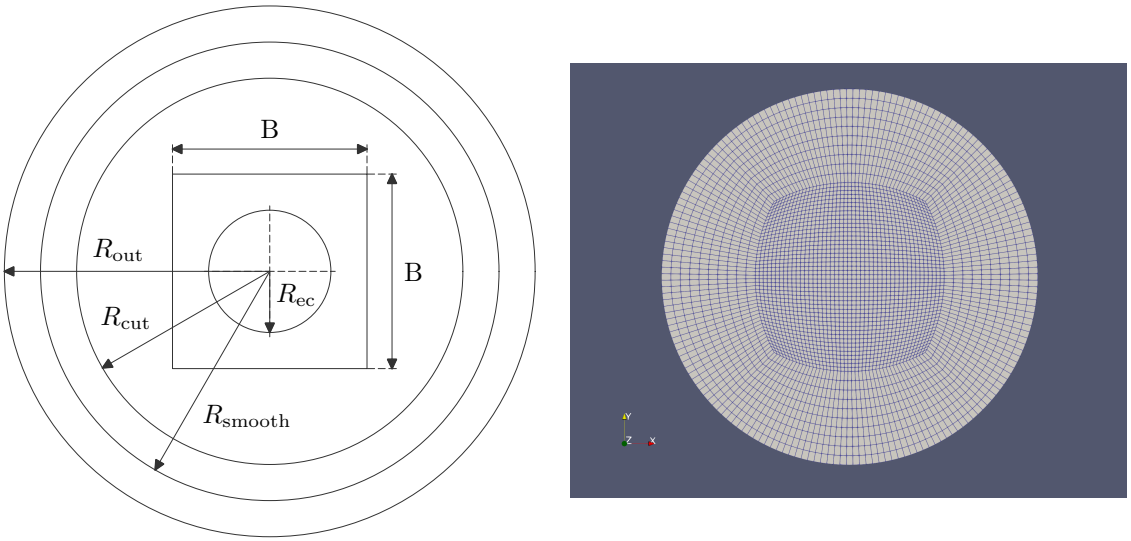
[3]The value $n = 3$ is often associated with a cubic law for energy dissipation, consistent with Kolmogorov's turbulence theory. In this context, it implies that energy decays proportionally to the cube of the length scale, following the classical turbulent energy cascade where large-scale eddies break into smaller ones until the energy is dissipated.

```
atmPlantCanopyUSource1
{
    type            atmPlantCanopyUSource;
    selectionMode   all;
}

atmPlantCanopyTSource1
{
    type            atmPlantCanopyTSource;
    selectionMode   all;
}

atmPlantCanopyTurbSource1
{
    type            atmPlantCanopyTurbSource;
    selectionMode   all;
    rho             rho;
}
```

Figure 6.15: Additional source terms in `fvOptions` file used to incorporate the forest.

6.3.2 Successor simulation

In the successor simulation, the mesh generation strategy was completely modified. The concept, presented in Figure 6.16, consists of creating a central square B with uniform resolution and slightly higher than the central circle of radius R_{ec}, whose area encloses all wind turbines and masts. Beyond the square, the mesh expands to the circular border of circle R_{out}.



(a) (b)

Figure 6.16: Representation of the numerical mesh of the successor simulation of the `atmFlatTerrain` tutorial. (a) Representative sketch of the mesh division strategy by smoothing sections. (b) Top view of *top* patch.

The terrain is manipulated following a progressive smoothing strategy until it becomes flat at the last edge of the computational domain. Inside R_{cut}, the terrain is

kept as is. Between R_{cut} and R_{smooth}, terrain is progressively smoothed, and outside R_{smooth} the terrain is transformed to become flat.

In terms of the vertical mesh, it appears that from a value of height of the first control volume, the mesh expands not to the top of the computational domain, but to a height of 500 m plus the maximum terrain height. Above this height, the vertical expansion continues but the horizontal resolution is doubled to save computational resources.

The source terms used in `fvOptions` are the same as those defined in the precursor simulations, with the exception of the terms referring to the forest (see Figure 6.15), which in this case were not used.

6.3.2.1 Boundary conditions

As for the boundary conditions used, *top* boundary is set to `zeroGradient` except for pressure, in which it was used a boundary condition of the type `fixedFluxPressure` which tells OpenFOAM to set a pressure gradient such that the flux from the `U` boundary condition is ensured (0 here) and a constant temperature of $T = 300.0$ K used at *top* patch.

For the side boundaries, precomputed numerical profiles are used as inflow conditions if the flow direction is pointing inside the domain. If it points outside, `zeroGradient` boundary condition type is used instead. This is achieved with the `freestream` boundary condition in OpenFOAM, which uses the fixed values from the profiles when the flux is inward to the domain and `zeroGradient` otherwise.This technique could be interesting to explore in coupling applications with WRF-ARW.

In the tutorial, the side boundaries are split into 36 patches of 10°each, and the profiles used are all equal for all patches and quantities except for patch `p1` that is slightly different from the other 35.

6.4 Analysis of the `atmForestStability` OpenFOAM tutorial

The tutorial contains only a precursor simulation with a similar setup to the `atmFlatTerrain` case. The computational domain has now a $200 \times 200 \times 6000$ m^2 mesh with an arrangement of $3 \times 3 \times 160$ control volumes (i.e. one more CV in the x and y directions when compared to `atmFlatTerrain`), as shown in Figure 6.17. All simulations use k_ε turbulence model.

The same boundary conditions are used, with the `wall` type assigned to the *ground* patch, `slip` to *sky*, and `cyclic` for *front*, *back*, *left* and *right*.

The case includes forest with a height of 18.2192 m where *setFields* is used to set `qPlant` $= -9$, `Cd` $= 0.2$ and `LAD` $= 0.14$. In terms of vertical mesh resolution, the mesh was designed to contain approximately 7 control volumes within inside the canopy, a value that would have to be increased for the simulations of ABL flows over complex terrain.

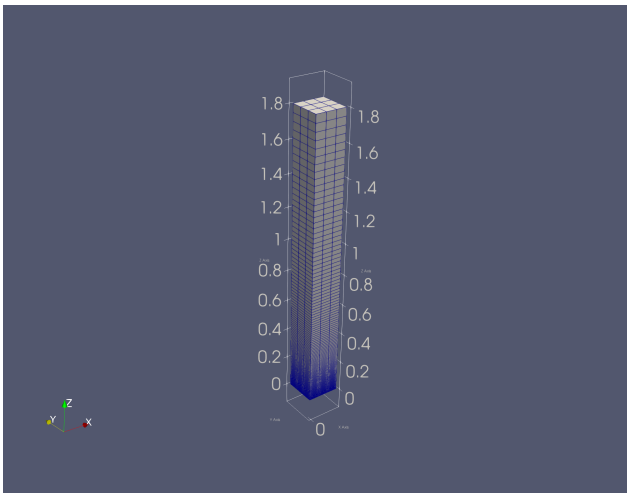


Figure 6.17: Representation of the computational domains of the precursor and successor simulations overlapped.

The tutorial aims to analyse the effect of stability only. The different stability regimes are achieved through 2 parameters:

1) `qPlant`, representing the tree-height specific heat flux, which refers to the volumetric heat generation over density. This field is defined in the `setFieldsDict` file for each stability regime. For the case of stable regime, `qPlant` $= -9$ m^2/s^3, making the canopy to act as a sink term. On the other hand, and for a unstable regime, `qPlant` $= 60$ m^2/s^3, making the canopy to act as a heat source. For a neutral regime, `qPlant` $= 0$ m^2/s^3. This parameter is used by `atmPlantCanopyTSource1` source term.

2) `Lmax`, representing the maximum mixing-length scale. For a stable regime, `Lmax` $= 13$ m (corresponding to a Monin-Obukhov length of $L = 80$ m). For a unstable regime, `Lmax` $= 200$ m (corresponding to a Monin-Obukhov length of $L = -50$ m). For neutral regime, `Lmax` $= 41$ m (corresponding to a Monin-Obukhov length of $L = \infty$ m). This parameter is used by `atmBuoyancyTurbSource1` and `atmLengthScaleTurbSource1` source terms.

6.5 Next Steps for further work

The final goal is to introduce the configurations, source terms and strategies that were described in the previous sections into the simulations of non-neutrally stratified ABL flows over the Mörknässkogen wind farm. To achieve this, a workflow was designed with the necessary steps to get there:

1) Use MERRA2 data to obtain the geostrophic wind and hence the source term associated with the pressure gradient, previously described in Section 6.3.1.2.

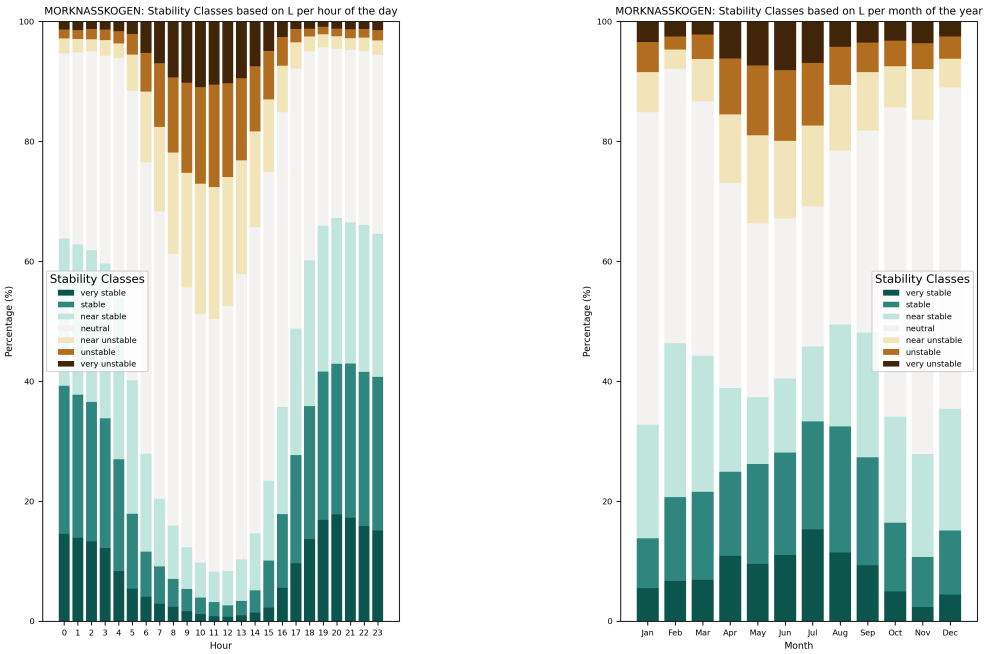
2) Perform a preliminary stability study with data obtained from MERRA2 to define the stability classes to be studied, the average Monin-Obukhov length L of each one, and the sensible heat flux in the the terrain ground surface. From there, establish the relationship between the Monin-Obukhov lengths L and the respective maximum mixing-length scales $L_{\max}$, to set this parameters in `fvOptions` for `atmBuoyancyTurbSource1` and `atmLengthScaleTurbSource1` source terms.

3) With this, perform preliminary simulations with the terms defined in point 2) for the classes of stability to be studied, thus obtaining a set of vertical profiles to be used in the simulations that are intended to be run. These simulations would include the Coriolis effect of site, the pressure gradient associated with the geostrophic wind, and the `mykEpsilonLopesdaCosta` forest model (although in the initial testing phase, the forest may be temporarily excluded to facilitate the process).

4) Perform successor simulations that take into account the results taken from the precursor simulations. It would be desirable to make a transition from real terrain to flat terrain away from the center of the mesh, due to the fact that the profile was generated on flat terrain in the precursor simulations and will vary according to the stability regime. At this stage, it is important to analyse the mesh generation process once again, to decide the path to follow. The first option would be to modify *write_blockMeshDict* to produce a circular numerical mesh, and the second option would be to test the procedures on a square mesh with *gsrf3*, used in all previous simulations.

5) As for boundary conditions, the aim is to use `freestream` boundary condition, previously described in Section 6.3.2.1, which was considered an intelligent application for this type of studies.

6) Refine the *Synthesis* algorithm to weight the results of each set depending on the frequency of each stability regime, to estimate production during the day period, during the night period, seasonal, or other temporal variation. Furthermore, compare the results obtained in terms of atmospheric stability signature with the data already extracted from MERRA2 for the region around Mörknässkogen wind farm, as illustrated in Figure 6.18.



(a) (b)

Figure 6.18: Atmospheric stability signature for the Mörknässkogen wind farm region obtained from MERRA2 analysis. (a) Daily. (b) Monthly.

Chapter 7

Conclusions and future work

7.1 Conclusions

In this work, it was performed an analysis on the wind resource assessments and adverse wind conditions in the installation area of Mörknässkogen wind farm, located in western Finland, using simulations of neutrally stratified ABL flows over complex terrain with OpenFOAM CFD model and tools developed in Python.

The analysis of wind data obtained from the measurement campaign that was conducted between 24/05/2014 and 07/04/2016 showed a predominant wind direction of 210° (SSW), both in terms of wind frequency and available energy in that same direction. Besides that, the algorithm developed in Python for the analysis of wind data (see Appendix B.1) was processed with the tree time series of measurements on mast i690 (120, 82, and 48 m AGL), which returned the average horizontal velocity $\overline{U_h}$ corresponding to these heights of 6.741 m/s, 5.838 m/s and 4.588 m/s, respectively. In the same way for turbulent intensity I, the values obtained were 10.010%, 12.575% and 18.683%, respectively.

In order to evaluate the wind resource with high precision, several configurations of the applied mathematical model were tested to find the one that best represents the local wind regime, considering a topographic profile with relatively flat terrain and the vast area of forest in the region, which consists of very tall evergreen pine and fir trees that can reach canopy heights of 30 m AGL.

Best overall results were achieved in sets of simulations with a forest model that was introduced using OpenFOAM's recently developed `kEpsilonLopesdaCosta` turbulence model. The implementation of this model involved the development of tools and procedures to increase the vertical mesh resolution bellow the canopy top, and also to model the trees above *ground* patch through the manipulation of both topography and roughness maps.

Validation of the mathematical model through the *Synthesis* algorithm revealed very significant improvements in the cross-predictions between mast i690 measurement heights, especially in terms of the non-dimensional horizontal velocity U_h^*, with a reduction in the RMS error from 13.06% to 1.13% (-11.93%), with the first value corresponding to SET_{102}, which was the best set among those performed considering only the aerodynamic roughness length z_0 (without forest), and the second value referring to SET_{210}, which obtained the best overall result.

This set included the application of the forest model calibrated with a canopy height of $h_{\text{cano}} = 17.5$ m AGL, drag coefficient of $C_d = 0.15$, mean leaf area density of $\overline{a(z)} = 0.125$ m^2/m^3, in addition to the numerical mesh of $150 \times 150 \times 70$ being generated with the new `canopyHD` mesh type with vertical mesh resolution of $nz_{\text{cano}} = 30$ bellow the canopy top.

Following the same comparison logic, turbulence behaviour also showed considerable improvements in terms of the non-dimensional TKE k^*, with a reduction in the respective RMS error from 3.21% corresponding to SET_{102} to 0.85% (-2.36%) corresponding to SET_{D5}, result that is consistent with the high canopy height of $h_{\text{cano}} = 22.5$ m AGL and $C_d = 0.30$ configured (see Table 5.3).

Although SET_{210} was the one that presented the best results, it was decided to consider SET_{214} as the best set due to the fact that it has a much more refined horizontal mesh, which in this case has an increase in resolution of +357.1% between SET_{210} ($150 \times 150 \times 70$) and SET_{214} ($250 \times 250 \times 90$), even though it possesses the disadvantage of additional computational resources needed, and also a small loss of precision in terms of mathematical model validation of -0.44%, since RMS erros obtained were 1.13% for SET_{210} and 1.57% for SET_{214}.

In terms of annual energy production, AEP value was calculated for each wind turbine based on time series synthesized to the real coordinates, which was then combined with the respective power curve to obtain an estimated combined AEP of 79584.7 MWh/year, that also corresponds to 4325 h in full load and a capacity factor of 49.38%.

In addition, another study was performed where the *Synthesis* algorithm was applied to transport the time series measured at 120 m AGL on mast i690 to a grid of points defined around the wind farm. This extrapolated analysis demonstrated that the four wind turbines were installed following a similar energy level, and were installed almost perpendicular to the prevailing wind direction of 210°, in order to face winds coming from this direction with high energy density, thus minimizing wake effects to increase its performance.

7.2 Future work

This Section presents suggestions for future work, from the perspective of the simulation of ABL flows to assess the wind resource and site verification with OpenFOAM:

1) Evaluate the tools and procedures developed in this work through their application in other existing wind farms with more complex topography and heavily forested terrain, namely in terms of postprocessing tools and forest modelling using the `kEpsilonLopesdaCosta` turbulence model.

2) Complete the development steps described in Section 6.5, in order to evaluate the accounting of thermal effects and Coriolis force in the simulations, and validate the procedures through their application in a wind farm in which this component is significant in the results obtained, which typically occurs in countries with strong sunlight during the day, such as Brazil.

3) Explore the option of conducting transient simulations of non-neutrally statified ABL flows, which also account for thermal and Coriolis effects using the *buoyantBoussinesqPimpleFoam* solver.

4) Explore the possibility of using a coupling strategy, for example with WRF-ARW, thus using mesoscale models that provide more sophisticated and precise boundary conditions for the microscale model (in this case OpenFOAM), being a procedure that revealed good improvements in the results obtained from the mathematician model validation in certain studies that used this technique [62].

This work is planned to yield two publications:

- Implementation of the `kEpsilonLopesdaCosta` canopy model for 2 real wind farm locations (Mörknässkogen and an additional site in Brazil).
- Extension of the use of the `kEpsilonLopesdaCosta` canopy model with stratification (as detailed in Section 6.5) for a forested site in Brazil where the impact of stratification is known to have significant impact on AEP estimates.

References

[1] "OpenFOAM v2306." [Online]. Available: https://www.openfoam.com/news/main-news/openfoam-v2306 (Accessed 2023-12-28).

[2] J. H. Ferziger, M. Peri, and R. L. Street, *Computational methods for fluid dynamics*, fourth edition ed. Cham: Springer, 2020.

[3] R. B. Stull, *An Introduction to Boundary Layer Meteorology*. Dordrecht: Kluwer Academic Publishers, 1988, (Accessed 2023-11-01).

[4] J. M. Wallace and P. V. Hobbs, *Atmospheric Science: An Introductory Survey*, 2nd ed., ser. International geophysics series. Amsterdam Paris: Academic press, 2006, no. volume 92.

[5] S. P. Arya, *Introduction to micrometeorology*, 2nd ed., ser. This is volume 79 in the International geophysics series. San Diego: Academic Press, 2001.

[6] T. R. Oke, *Boundary Layer Climates*, second edition ed. London: Routledge, 1987, oCLC: 51200739.

[7] T. L. Bergman and F. P. Incropera, Eds., *Fundamentals of heat and mass transfer*, 7th ed. Hoboken, NJ: Wiley, 2011.

[8] M. Z. Jacobson, *Fundamentals of atmospheric modeling*, 2nd ed. Cambridge, UK ; New York: Cambridge University Press, 2005.

[9] A. Monin and A. Obukhov, "Basic laws of turbulent mixing in the surface layer of the atmosphere," 2009.

[10] J. A. Businger, J. C. Wyngaard, Y. Izumi, and E. F. Bradley, "Flux-Profile Relationships in the Atmospheric Surface Layer," *Journal of the Atmospheric Sciences*, vol. 28, no. 2, pp. 181–189, Mar. 1971. [Online]. Available: http://journals.ametsoc.org/doi/10.1175/1520-0469(1971)028<0181:FPRITA>2.0.CO;2 (Accessed 2024-08-16).

[11] Z. Sorbjan and A. A. Grachev, "An Evaluation of the FluxGradient Relationship in the Stable Boundary Layer," *Boundary-Layer Meteorology*, vol. 135, no. 3, pp. 385–405, Jun. 2010. [Online]. Available: http://link.springer.com/10.1007/s10546-010-9482-3 (Accessed 2024-08-16).

[12] C. J. Desmond, S. J. Watson, S. Aubrun, S. Ávila, P. Hancock, and A. Sayer, "A study on the inclusion of forest canopy morphology data in numerical simulations for the purpose of wind resource assessment," *Journal of Wind Engineering and Industrial Aerodynamics*, vol. 126, pp. 24–37, Mar. 2014. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0167610513002869 (Accessed 2024-08-18).

[13] J. Lopes Da Costa, "Atmospheric Flow over Forested and Non-Forested Complex Terrain," PhD thesis, ISEP, 2007.

[14] U. Svensson and K. Häggkvist, "A two-equation turbulence model for canopy flows," *Journal of Wind Engineering and Industrial Aerodynamics*, vol. 35, pp. 201–211, Jan. 1990. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/016761059090216Y (Accessed 2024-08-18).

[15] R. S. Green, "Modelling Turbulent Air Flow in a Stand of Widely-Spaced Trees," *Phoenics J.*, pp. 294–312, 1992.

[16] J. Liu, J. M. Chen, T. A. Black, and M. D. Novak, "E- modelling of turbulent air flow downwind of a model forest edge," *Boundary-Layer Meteorology*, vol. 77, no. 1, pp. 21–44, Jan. 1996. [Online]. Available: http://link.springer.com/10.1007/BF00121857 (Accessed 2024-08-18).

[17] C. Sanz, "A Note on k - Modelling of Vegetation Canopy Air-Flows," *Boundary-Layer Meteorology*, vol. 108, no. 1, pp. 191–197, Jul. 2003. [Online]. Available: http://link.springer.com/10.1023/A:1023066012766 (Accessed 2024-08-18).

[18] G. G. Katul, L. Mahrt, D. Poggi, and C. Sanz, "ONE- and TWO-Equation Models for Canopy Turbulence," *Boundary-Layer Meteorology*, vol. 113, no. 1, pp. 81–109, Oct. 2004. [Online]. Available: http://link.springer.com/10.1023/B:BOUN.0000037333.48760.e5 (Accessed 2024-08-18).

[19] M. R. Raupach and A. S. Thom, "Turbulence in and above Plant Canopies," *Annual Review of Fluid Mechanics*, vol. 13, no. 1, pp. 97–129, Jan. 1981. [Online]. Available: https://www.annualreviews.org/doi/10.1146/annurev.fl.13.010181.000525 (Accessed 2024-01-05).

[20] Internationale Elektrotechnische Kommission, Ed., *Design requirements*, edition 4.0 ed., ser. Wind energy generation systems / International Electrotechnical Commission. Geneva, Switzerland: International Electrotechnical Commission, 2019, no. part 1.

[21] "Weather Research & Forecasting Model (WRF) | Mesoscale & Microscale Meteorology Laboratory." [Online]. Available: https://www.mmm.ucar.edu/models/wrf (Accessed 2023-12-29).

[22] E. Leblebici, "Unsteady atmospheric flow solutions with OpenFOAM coupled with the numerical weather prediction software, WRF," PhD thesis, Middle East Technical University, Feb. 2018.

[23] R. Laprise, "The Euler Equations of Motion with Hydrostatic Pressure as an Independent Variable," *Monthly Weather Review*, vol. 120, no. 1, pp. 197–207, Jan. 1992. [Online]. Available: http://journals.ametsoc.org/doi/10.1175/1520-0493(1992)120<0197:TEEOMW>2.0.CO;2 (Accessed 2024-01-04).

[24] W. C. Skamarock, J. B. Klemp, J. Dudhia, D. O. Gill, Z. Liu, J. Berner, W. Wang, J. G. Powers, M. G. Duda, D. M. Barker, and X.-Y. Huang, "A Description of the Advanced Research WRF Model Version 4," UCAR/NCAR, Tech. Rep., Mar. 2019. [Online]. Available:

https://opensky.ucar.edu/islandora/object/opensky:2898 (Accessed 2024-01-04).

[25] X.-M. Hu, J. W. Nielsen-Gammon, and F. Zhang, "Evaluation of Three Planetary Boundary Layer Schemes in the WRF Model," *Journal of Applied Meteorology and Climatology*, vol. 49, no. 9, pp. 1831–1844, Sep. 2010. [Online]. Available: http://journals.ametsoc.org/doi/10.1175/2010JAMC2432.1 (Accessed 2024-01-05).

[26] T. Zhang, L. Cao, S. Li, C. Zhan, J. Wang, and T. Zhao, "Comprehensive sensitivity analysis of the WRF model for meteorological simulations in the Arctic," *Atmospheric Research*, vol. 299, p. 107200, Apr. 2024. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0169809523005975 (Accessed 2024-01-05).

[27] R. Borge, V. Alexandrov, J. José Del Vas, J. Lumbreras, and E. Rodríguez, "A comprehensive sensitivity analysis of the WRF model for air quality applications over the Iberian Peninsula," *Atmospheric Environment*, vol. 42, no. 37, pp. 8560–8574, Dec. 2008. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1352231008007954 (Accessed 2024-01-05).

[28] D. Carvalho, A. Rocha, M. Gómez-Gesteira, and C. Santos, "A sensitivity study of the WRF model in wind simulation for an area of high wind energy," *Environmental Modelling & Software*, vol. 33, pp. 23–34, Jul. 2012. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1364815212000382 (Accessed 2024-01-05).

[29] F. A. Castro, "Métodos Numéricos Para A Simulação De Escoamentos Atmosféricos Sobre Topografia Complexa," PhD thesis, Faculdade de Engenharia da Universidade do Porto, May 1997.

[30] "Meteodyn | Numerical wind engineering Software and consultancy." [Online]. Available: https://meteodyn.com/ (Accessed 2024-01-05).

[31] "WindSim." [Online]. Available: https://windsim.com/ (Accessed 2024-01-05).

[32] "Wind energy industry-standard software - WAsP." [Online]. Available: https://www.wasp.dk/ (Accessed 2024-01-04).

[33] P. S. Jackson and J. C. R. Hunt, "Turbulent wind flow over a low hill," *Quarterly Journal of the Royal Meteorological Society*, vol. 101, no. 430, pp. 929–955, Oct. 1975. [Online]. Available: https://rmets.onlinelibrary.wiley.com/doi/10.1002/qj.49710143015 (Accessed 2024-01-04).

[34] I. Troen and E. L. Petersen, *European wind atlas*. Roskilde, Denmark: Published for the Commission of the European Communities, Directorate-General for Science, Research, and Development, Brussels, Belgium by Risø National Laboratory, 1989.

[35] P. Sanderhoff, "PARK- User's Guide. A PC-program for calculation of wind turbine park performance," *Risø National Laboratory*, vol. Risø-I, no. 668, 1993.

[36] Y. Yang, M. Gu, S. Chen, and X. Jin, "New inflow boundary conditions for modelling the neutral equilibrium atmospheric boundary layer in computational wind engineering," *Journal of Wind Engineering and Industrial Aerodynamics*, vol. 97, no. 2, pp. 88–95, Feb. 2009. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0167610508001815 (Accessed 2024-01-05).

[37] C.-Y. Chang, J. Schmidt, M. Dörenkämper, and B. Stoevesandt, "A consistent steady state CFD simulation method for stratified atmospheric boundary layer flows," *Journal of Wind Engineering and Industrial Aerodynamics*, vol. 172, pp. 55–67, Jan. 2018. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0167610517305135 (Accessed 2024-01-05).

[38] M. J. Churchfield, S. Lee, and P. J. Moriarty, "Adding complex terrain and stable atmospheric condition capability to the OpenFOAM-based flow solver of the simulator for on/offshore wind farm applications (SOWFA)," *ITM Web of Conferences*, vol. 2, p. 02001, 2014. [Online]. Available: http://www.itm-conferences.org/10.1051/itmconf/20140202001 (Accessed 2024-01-05).

[39] F. M. White, *Fluid mechanics*, 6th ed., ser. Mcgraw-Hill series in mechanical engineering. New York, NY: McGraw-Hill, 2009.

[40] "Millennium Problems: Navier-Stokes Equation." [Online]. Available: https://www.claymath.org/millennium/navier-stokes-equation/ (Accessed 2023-12-27).

[41] S. B. Pope, *Turbulent flows*. Cambridge ; New York: Cambridge University Press, 2000.

[42] F. A. Castro and C. Silva Santos, *Mecânica dos Fluidos Computacional*, Instituto Superior de Engenharia do Porto, Dec. 2022.

[43] P. Durbin, "A perspective on recent developments in RaNS modeling," in *Engineering Turbulence Modelling and Experiments 5*. Elsevier, 2002, pp. 3–16. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/B9780080441146500028 (Accessed 2023-12-28).

[44] K. Hanjalić, "One-point closuremodels for buoyancy-driven turbulent flows," *Department of Applied Physics, Delft University of Technology Lorentzweg 1*, 2002.

[45] H. K. Versteeg and W. Malalasekera, *An introduction to computational fluid dynamics: the finite volume method*, 2nd ed. Harlow, England ; New York: Pearson Education Ltd, 2007, oCLC: ocm76821177.

[46] B. E. Launder and D. B. Spalding, *The numerical computation of turbulent flow*. Imperial College of Science and Technology , Department of Mechanical Engineering: Computer Methods in Applied Mechanics and Engineering, 1973.

[47] W. Jones and B. Launder, "The prediction of laminarization with a two-equation model of turbulence," *International Journal of Heat and Mass*

Transfer, vol. 15, no. 2, pp. 301–314, Feb. 1972. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/0017931072900762 (Accessed 2024-01-05).

[48] A. C. M. Beljaars, J. L. Walmsley, and P. A. Taylor, "A mixed spectral finite-difference model for neutrally stratified boundary-layer flow over roughness changes and topography," *Boundary-Layer Meteorology*, vol. 38, no. 3, pp. 273–303, Feb. 1987. [Online]. Available: http://link.springer.com/10.1007/BF00122448 (Accessed 2023-12-28).

[49] T. H. Shih, W. W. Liou, A. Shabbir, Z. Yang, and J. Zhu, "A New K-epsilon Eddy Viscosity Model for High Reynolds Number Turbulent Flows: Model Development and Validation," *Computers & Fluids*, 1994.

[50] D. C. Wilcox, "Reassessment of the scale-determining equation for advanced turbulence models," *AIAA Journal*, vol. 26, no. 11, pp. 1299–1310, Nov. 1988. [Online]. Available: https://arc.aiaa.org/doi/10.2514/3.10041 (Accessed 2023-12-29).

[51] O. Temel, S. Porchetta, L. Bricteux, and J. Van Beeck, "RANS closures for non-neutral microscale CFD simulations sustained with inflow conditions acquired from mesoscale simulations," *Applied Mathematical Modelling*, vol. 53, pp. 635–652, Jan. 2018. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0307904X17305693 (Accessed 2024-01-05).

[52] P. Richards and R. Hoxey, "Appropriate boundary conditions for computational wind engineering models using the k- turbulence model," *Journal of Wind Engineering and Industrial Aerodynamics*, vol. 46-47, pp. 145–153, Aug. 1993. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/0167610593901247 (Accessed 2024-01-05).

[53] H. G. Weller, G. Tabor, H. Jasak, and C. Fureby, "A tensorial approach to computational continuum mechanics using object-oriented techniques," *Computers in Physics*, vol. 12, no. 6, pp. 620–631, Nov. 1998. [Online]. Available: https://pubs.aip.org/cip/article/12/6/620/510187/A-tensorial-approach-to-computational-continuum (Accessed 2024-01-05).

[54] J. Azevedo, "Development of procedures for the simulation of atmospheric flows over complex terrain, using OpenFOAM," Master's thesis, ISEP, 2013.

[55] F. A. Castro, C. Silva Santos, and J. C. Costa, "Development of a meso-microscale coupling procedure for site assessment in complex terrain," 2010.

[56] F. A. Castro, J. M. L. M. Palma, and A. Silva Lopes, "Simulation of the Askervein Flow. Part 1: Reynolds Averaged NavierStokes Equations (k Turbulence Model)," *Boundary-Layer Meteorology*, vol. 107, no. 3, pp. 501–530, Jun. 2003. [Online]. Available: http://link.springer.com/10.1023/A:1022818327584 (Accessed 2024-08-24).

[57] Vizzuality, "Interactive World Forest Map & Tree Cover Change Data | GFW." [Online]. Available: https://www.globalforestwatch.org/map (Accessed 2024-09-01).

[58] P. Potapov, X. Li, A. Hernandez-Serna, A. Tyukavina, M. C. Hansen, A. Kommareddy, A. Pickens, S. Turubanova, H. Tang, C. E. Silva, J. Armston, R. Dubayah, J. B. Blair, and M. Hofton, "Mapping global forest canopy height through integration of GEDI and Landsat data," *Remote Sensing of Environment*, vol. 253, p. 112165, Feb. 2021. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0034425720305381 (Accessed 2024-09-01).

[59] J. Finnigan, "Turbulence in Plant Canopies," *Annual Review of Fluid Mechanics*, vol. 32, no. 1, pp. 519–571, Jan. 2000. [Online]. Available: https://www.annualreviews.org/doi/10.1146/annurev.fluid.32.1.519 (Accessed 2024-09-01).

[60] M. Alletto, A. Radi, J. Adib, J. Langner, C. Peralta, A. Altmikus, and M. Letzel, "E-wind: Steady state CFD approach for stratified flows used for site assessment at enercon," vol. 1037, p. 072020. [Online]. Available: https://iopscience.iop.org/article/10.1088/1742-6596/1037/7/072020 (Accessed 2024-09-19).

[61] G. L. Mellor and T. Yamada, "Development of a turbulence closure model for geophysical fluid problems," vol. 20, no. 4, pp. 851–875. [Online]. Available: https://agupubs.onlinelibrary.wiley.com/doi/10.1029/RG020i004p00851 (Accessed 2024-09-19).

[62] H. Monteiro, "Estudo do recurso eólico usando OpenFOAM com condições de fronteira de modelos de mesoescala," Master's thesis, ISEP, 2023.

Appendix A

List of newly developed and inherited codes

In this Appendix is presented the list of codes that were used in this work, with part of these codes being developed from scratch to be implemented in the study, and the other part being inherited, and in some cases modified, from previous studies performed by other engineers.

Therefore, and out of respect and admiration for the work previously developed by them, the lists presented in Tables A.1 and A.2 indicates among these codes which ones were created from scratch, which were modified, and which were used without any changes.

Table A.1: List of inherited codes

Source code	**Development**	**Modifications**
`gsrf3.f90`	An ISEP *in-house* code developed by a group of researchers.	None
`groug.f90`	An ISEP *in-house* code developed by a group of researchers.	None
`write_blockMeshDict.f90`	Developed in Azevedo (2013) [54]	Significantly modified to include canopy simulations, becoming `write_blockMeshDictCano.f90`.
`write_z0.f90`	Developed in Azevedo (2013) [54]	Slightly modified in terms of the code appearance.
`write_bCs.f90`	Developed in Azevedo (2013) [54]	Significantly modified to include canopy simulations, becoming `write_bCsCano.f90`.
`write_turbulenceProperties.f90`	Developed in Azevedo (2013) [54]	Minor modifications to include turbulence model `kEpsilonLopesdaCosta`.

For greater detail, the codes that were specially developed from scratch for this work, as well as those that were modified, are described in the following appendices.

Table A.2: List of newly developed codes

Source code	**Development**	**Modifications**
`Wind_data_analysis.py`	Developed from scratch for this study.	-
`Synthesis_analysis.py`	Developed from scratch for this study.	-
`Generate_canopy_vtk.py`	Developed from scratch for this study.	-
`vtk_to_stl.py`	Developed from scratch for this study.	-

Appendix B

Purpose built Python codes

In this Appendix is presented the developed Python codes used to perform pre-processing and post-processing operations throughout the OpenFOAM simulations. For each one of them, an individual presentation is made, which consists of the name given to the file, a short description of the operations performed by the code, followed by the code itself listed in its entirety.

B.1 `Wind_data_analysis.py`

This algorithm reads and processes all wind records, and subsequently returns the data obtained from the analysis and produces graphs and tables that represent the local wind regime. Initially, It starts by reading the respective `Datalog.txt` file to extract all the records, and then the algorithm executes the following sequence of operations:

(i) Construction of a table with the absolute and relative frequencies referring to the data processed by each 30° sector.

(ii) Construction of a table with the 10 minutes average velocity for each 30°.

(iii) Construction of a table with the energy contribution of each 30°, where the percentage of energy contribution of each sector is calculated, in terms of total energy density.

(iv) Construction of a table with the average turbulent intensity for each wind direction, which is obtained by dividing the standard deviation by the average velocity. Additionally, a scatterplot is created that shows the distribution of turbulent intensity across different velocity ranges.

(v) Construction of a histogram with the relative frequency of occurrences in terms of the number of records for different velocity ranges, with intervals of 1 m/s.

(vi) In the same histogram mentioned in the previous point (v), add a Weibull curve that represents a probability distribution function for the different velocity ranges, and calculates the Weibull parameters A_{Weibull} and k_{Weibull}.

Once this processing is complete, the results obtained are used to plot wind rose graphs and histograms, such as those presented in Section 3.4.3. The source code of `Wind_data_analysis.py` file is presented below.

```python
# »»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»
# Libraries

import math as math
import numpy as np
import pandas as pd
import os
import matplotlib.pyplot as plt
from scipy.optimize import fsolve
from matplotlib import cm
from fluidfoam import readmesh
from fluidfoam import readvector
from fluidfoam import readscalar
from dataclasses import dataclass
from scipy.interpolate import griddata
from scipy.interpolate import LinearNDInterpolator
from scipy.interpolate import RegularGridInterpolator
from scipy.special import gamma
import time
import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)

# »»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»
# Functions

def round_fixed(value,div):
    div2=value/div
    number_dec=div2%1
    result= round(value/div,0)
    if number_dec==0.5:
        result=math.ceil(value/div)-1
    return(int(result))

def get_index_degree(value):
    index=-1
    if (value>345.0 and value<=360.0) or value<=15.0:
        index=0
    if value>15.0 and value<=345.0:
        index=round_fixed(value,30)
    return(int(index))

def Weibull_parameters_A_k(input_array_U):

    cont = 0
    array_U = []
    for i in range(0,len(input_array_U)):
        if input_array_U[i] != 99.99:
            array_U.append(input_array_U[i])
            cont=cont+1

    U_avg = np.mean(array_U)
    print("U_avg =",U_avg)
    Somamin = pow(1,12)
    A_min = round(U_avg,0)-0.4* round(U_avg,0)
    A_max = round(U_avg,0)+0.4* round(U_avg,0)
    contPot = 0
    contSupMed = 0

    for i in range(0,len(array_U)):
        contPot = contPot + pow(array_U[i],3)
        if array_U[i] > U_avg:
            contSupMed = contSupMed + 1

    for A in np.arange(A_min,A_max,0.01):
        for k in np.arange(0.01,15.0,0.01):
            x = 1+(3/k)
            Gama=gamma(x)*(A**3)
            Fw=math.exp(-((U_avg/A)**k))
            Error_1= abs((contSupMed/cont)-Fw)/(contSupMed/cont)*100.0
            Error_2= abs((contPot/cont)-Gama)/(contPot/cont)*100.0
            Soma=Error_1+Error_2
            if Soma<Somamin:
                Somamin=Soma
                A_final=A
                k_final=k
    return(A_final,k_final)

def Weibull_frequencies(input_array_U):

    U_interval = np.zeros(20)
    Freq_by_velocity = np.zeros(20)
    Freq_by_velocity_rel = np.zeros(20)

    for i in range(0,len(input_array_U)):
        posicao = math.trunc(input_array_U[i])
```

```
86          if input_array_U[i] < 20:
87              Freq_by_velocity[posicao] += 1
88      Freq_by_velocity_total = sum(Freq_by_velocity)
89
90      for i in range(0,20):
91          U_interval[i] = i + 1
92          Freq_by_velocity_rel[i]=(Freq_by_velocity[i]/Freq_by_velocity_total)*100.0;
93      return(U_interval,Freq_by_velocity_rel)
94
95  # »»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»
96  # General parameters
97
98  file_name = "DataLog_120m_SIM.txt"
99  U_ref = 5.838 # Average mast U at 82 m agl
100 TI_ref = 12.575 # Average mast TI at 82 m agl
101
102 i = 0
103 ID = ['00','01','02','03','04','05','06','07','08','09','10','11','12']
104 Orientacao = ['N','NNE','ENE','E','ESE','SSE','S','SSW','WSW','W','WNW','NNW']
105 Graus = ['0','30','60','90','120','150','180','210','240','270','300','330']
106 Freq_total = 0
107 Freq_total_turb = 0
108 Freq_descartados = 0
109 Ficheiros_totais = 0
110 U_total_ms_cont = 0
111
112 TI = []
113 Freq_by_sector = [0,0,0,0,0,0,0,0,0,0,0,0]
114 Freq_by_sector_rel = [0,0,0,0,0,0,0,0,0,0,0,0]
115 Freq_by_sector_rel_aux = [0,0,0,0,0,0,0,0,0,0,0,0]
116 Freq_by_velocity = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
117 Freq_by_velocity_rel = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
118 Freq_by_velocity_rel_aux = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
119
120 Velocidade_total = 0.0
121 Velocidade_media = 0.0
122 posicao = 0
123
124 A_min = 0.0
125 A_max = 0.0
126 k = 0.0
127 Gama = 0.0
128 Erro_1 = 0.0
129 Erro_2 = 0.0
130 Soma = 0.0
131 Somamin = pow(1,10)
132 x = 0
133 pi = np.pi
134 Fw = 0
135
136 # Read wind data files
137 file = open(file_name, "r")
138
139 # Wind direction
140 # 00. N : Norte --> [346;15]ž
141 # 01. NNE : Nor-nordeste --> [16;45]ž
142 # 02. ENE : Lestenordeste --> [46;75]ž
143 # 03. E : Este --> [76;105]ž
144 # 04. ESE : Lestesudeste --> [106;135]ž
145 # 05. SSE : Sul-sudeste --> [136;165]ž
146 # 06. S : Sul --> [166;195]ž
147 # 07. SSW : Sul-sudoeste --> [196;225]ž
148 # 08. WSW : Oés-sudoeste --> [226;255]ž
149 # 09. W : Oeste --> [256;285]ž
150 # 10. WNW : Oés-noroeste --> [286;315]ž
151 # 11. NNW : Nor-noroeste --> [316;345]ž
152
153 Velocidade_total = 0
154 desv_pad = 0
155 n = 0
156 U_max = 0
157 data_array_U = []
158 cont = 0
159
160 for line in file.readlines():
161     leitura=line.split(",")
162     data_array_U.append(float(leitura[2]))
163
164     # Frequency by sector
165     index=get_index_degree(float(leitura[1]))
166
167     if (index==-1):
168         Freq_descartados+=1
169     else:
170         Freq_by_sector[index]+=1
```

```
171
172     # Frequency by velocity
173     posicao=math.trunc(float(leitura[2]))
174     if float(leitura[2])<20:
175         Freq_by_velocity[posicao]=Freq_by_velocity[posicao]+1
176         cont += 1
177     if float(leitura[2])!=99.99:
178         Velocidade_total=Velocidade_total+ float(leitura[2])
179         desv_pad = desv_pad + float(leitura[5])
180         n =n+1
181
182     # Maximum speed
183     if float(leitura[2]) > U_max:
184         U_max = float(leitura[2])
185 file.close()
186
187 # Relative frequency by sector
188 Freq_total= sum(Freq_by_sector)
189 Ficheiros_totais=Freq_total+Freq_descartados
190 for i in range(0,12):
191     Freq_by_sector_rel[i]=(Freq_by_sector[i]/Freq_total)*100.0;
192
193 # Average velocity
194 Velocidade_media = Velocidade_total/n
195
196 # »»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»
197 # Weibull parameters
198
199 result_1 = Weibull_parameters_A_k(data_array_U)
200 result_2 = Weibull_frequencies(data_array_U)
201
202 A_final = result_1[0]
203 k_final = result_1[1]
204 Weibull_U = result_2[0]
205 Weibull_f = result_2[1]
206
207 # »»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»
208 # Relative frequency by velocity
209
210 U_total_ms_cont= sum(Freq_by_velocity)
211 for i in range(0,20):
212     Freq_by_velocity_rel[i]=(Freq_by_velocity[i]/U_total_ms_cont)*100.0;
213
214 # »»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»
215 # Average speed + turbulent intensity
216
217 U_total = np.zeros(12)
218 U_media = np.zeros(12)
219 cont = np.zeros(12)
220 TI_total = np.zeros(12)
221 TKE_total = np.zeros(12)
222 TI_cont = np.zeros(12)
223 TKE_cont = np.zeros(12)
224 TI_medio = np.zeros(12)
225 delta_TI_medio = np.zeros(12)
226 TKE_medio = np.zeros(12)
227 delta_s = np.zeros(12)
228
229 Vel_plot = []
230 TI_plot = []
231 TI_scatter_total = []
232 TI_scatter_cont = []
233 TI_scatter_media = []
234 Vel_scatter = []
235
236 TI_calc = 0
237 TKE_calc = 0
238 contador = 0
239 i = 0
240 j = 0
241 s = 0
242 index = 0
243
244 # Fill the array inicially
245 for j in np.arange(5.0,30.1,0.1):
246     j = round(j,1)
247     TI_scatter_total.append(0)
248     TI_scatter_cont.append(0)
249     TI_scatter_media.append(0)
250     Vel_scatter.append(j)
251
252 # Read wind data file
253 file= open(file_name, "r")
254 validos = 0
255 TI_general_total = 0
256 TI_general_cont = 0
```

```
257 TI_general_avg = 0
258 Energy_total = 0
259 Energy_cont = 0
260 Energy_avg = 0
261
262 # Average 10 minutes velocity
263 for line in file.readlines():
264     leitura=line.split(",")
265     if float(leitura[2])!=99.99 and float(leitura[1])!=999.0:
266         validos+=1
267         index=get_index_degree(float(leitura[1]))
268         U_total[index]+= float(leitura[2])
269         cont[index]+=1
270         if float(leitura[2])>5.0:
271             TI_calc=(float(leitura[5])/float(leitura[2]))*100.0
272             TI_total[index]+=TI_calc
273             TI_general_total+=TI_calc
274             TI_cont[index]+=1
275             TI_general_cont+=1
276             TKE_calc=(3/2)*pow(TI_calc/100.0,2)*pow(float(leitura[2]),2)
277             TKE_total[index]+=TKE_calc
278             TKE_cont[index]+=1
279
280         # Calculates turbulent intensity (TI)
281         if float(leitura[2])>=5.0:
282             s=s+1
283             Vel_plot.append(float(leitura[2]))
284             TI_plot.append(TI_calc)
285             Vel_aux= round(float(leitura[2]),1)
286             index= int((Vel_aux-5)*10.0)
287             TI_scatter_total[index]=TI_scatter_total[index]+TI_calc
288             TI_scatter_cont[index]=TI_scatter_cont[index]+1
289
290         Energy_total += 0.5*1.239*float(leitura[2])**3
291         Energy_cont += 1
292
293     contador=contador+1
294 file.close()
295
296 diametro_rotor= 160.0
297 area_rotor = np.pi/4*pow(diametro_rotor,2)
298
299 TI_general_avg = TI_general_total/TI_general_cont
300 Energy_avg = Energy_total / Energy_cont
301
302 # »»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»
303 # Calculates the average turbulent intensity by velocity range
304
305 for a in range(0,250):
306     if TI_scatter_cont[a]>0:
307         TI_scatter_media[a]=TI_scatter_total[a]/TI_scatter_cont[a]
308     else:
309         TI_scatter_media[a]=TI_scatter_total[a]
310
311 # Removes values of TI < 0 from the scatter plot
312 for a in range(0,250):
313     if TI_scatter_media[a]<=0 or Vel_scatter[a]>22.0:
314         TI_scatter_media[a]=-10
315
316 # »»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»
317 # Calculates the average velocity
318
319 for i in range(0,12):
320     if cont[i]>0:
321         U_media[i]=U_total[i]/cont[i]
322     else:
323         U_media[i]=0
324     # SPEED-UP
325     delta_s[i] = (U_media[i])/U_ref
326
327 # »»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»
328 # Relative frequency for turbulent intensity by sector
329
330 for i in range(0,12):
331     TI_medio[i]=TI_total[i]/TI_cont[i];
332     TKE_medio[i]=3/2*pow(U_media[i],2)*pow(TI_medio[i]/100.0,2)
333     TKE_div_U2_medio[i]= TKE_medio[i] / pow(U_media[i],2)
334     delta_TI_medio[i] = (TI_medio[i]-TI_ref) / TI_ref
335
336 # »»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»
337 # Wind energy
338
339 Energia = np.zeros(12)
340 Energia_rel = np.zeros(12)
341 Energia_total = 0
```

```
342 i = 0
343 j = 0
344 k = 0
345 x = 0
346
347 for i in range(0,12):
348     Energia[i]=U_media[i]**(3)*Freq_by_sector[i]
349 Energia_total = sum(Energia)
350
351 # »»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»
352 # Energy contribution by sector
353
354 Energia_W_m2 = 0
355 for j in range(0,12):
356     Energia_rel[j]=Energia[j]/Energia_total*100.0
357
358 for i in range (0,len(Energia_rel)):
359     print(Energia_rel[i])
```

B.2 Synthesis_analysis.py

This algorithm reads the OpenFOAM results for a given SET of simulations that contains the 12 simulations corresponding to each wind direction, and then runs a post-processing algorithm that begins with a three-dimensional scan of the numerical mesh to calculate the desired quantities in all points, followed by three-dimensional interpolations to the wind turbines and mast i690 coordinates, to finally apply the *Synthesis* algorithm to estimate the wind farm's annual energy production and to calculate harmful parameters in the wind turbines at hub height of 166 m AGL.

In terms of procedure, it is necessary to previously configure some parameters that are presented at the beginning of the code, which impact the operations that the algorithm performs. Then, for each simulated direction, the algorithm executes the following processing routine:

(i) Open the folder corresponding to the simulation directory and read the three-dimensional mesh to obtain xyz coordinates, U and k, for each control volume.

(ii) Open the `preproc.cfg` file to extract the rotation applied to the mesh (parameter `rot`), and the coordinates of its geographic center (parameters `xcentre` and `ycentre`).

(iii) Open the `windie.cfg` file to extract the number of control volumes configured for the numerical mesh in the x and y directions (parameters `nig` and `njg`).

(iv) Open the `_preprocDict` file to extract the number of control volumes configured for the numerical mesh in the z direction (parameters `nz`), and canopy related parameters such us canopy height h_{cano} (parameter `hcano`), number of control volumes in the high resolution area below the canopy nz_{cano} (parameter `nzcano`), and also the configuration that defines the introduction of the forest model in the simulations (parameter `icano`).

(v) Perform a three-dimensional scan on the numerical mesh, and for each control volume executes the following procedure:

- Invert the rotation on the mesh introduced by OpenFOAM: converts the coordinates of the point [i,j,k] to polar coordinates, applies a rotation symmetric to the initially applied rotation, and converts it back into Cartesian coordinates again.
- Adds the geographic coordinates of the mesh center to obtain the geographic coordinates of the point.
- Calculate the magnitude of velocity based on its horizontal and vertical components.
- Calculate the wind direction from the velocity horizontal and vertical components.
- Calculate the wind shear factor α and flow inclination γ.
- Perform a three-dimensional interpolation for the wind turbines and mast i690 coordinates to extract all the quantities at these points.
- Calculate the speed-up ratio $\Delta S = \widetilde{U}_h^t / \widetilde{U}_h^{\oplus}$ for each turbine.

- Stores all processed data in an individual structure for each wind direction.

(vi) Read the files corresponding to the wind data and apply the *Synthesis* algorithm to transport the time series measured at mast i690 to each wind turbines.

(vii) Calculate the combined annual energy production of the wind farm and other additional quantities to perform a site assessment.

The source code of `Synthesis_analysis.py` file is presented below.

```
# »»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»
# Libraries

import math as math
import numpy as np
import pandas as pd
import os
import matplotlib.pyplot as plt
from scipy.optimize import fsolve
from matplotlib import cm
from fluidfoam import readmesh
from fluidfoam import readvector
from fluidfoam import readscalar
from dataclasses import dataclass
from scipy.special import gamma
from scipy.interpolate import griddata
from scipy.interpolate import interp1d
from scipy.interpolate import LinearNDInterpolator
from scipy.interpolate import RegularGridInterpolator
import time
import warnings
from ipywidgets import IntProgress,Label
from IPython.display import display,HTML
warnings.filterwarnings("ignore", category=DeprecationWarning)

# »»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»
# General parameters
path = "F:/DPEST/MORKNASSKOGEN/_FINAL/214.FOREST_250x250x90_CanoMESH/"
ref_SET = '214'

sinthesys_mastro = 0
type_mastro = 1
newCanoMesh = 1
s_inicial = 0
s_final = 12
blade_diameter = 160.0

# Analysis on a grid of points
dxn = 10000.0
dyn = 10000.0
zagl_grid = 166.0

cells_x_grid = 25
cells_y_grid = 25

# Coordinates for mast and turbines
if sinthesys_mastro == 1:
    if type_mastro == 1:
        coord_Mastro = [573295.0, 7019569.0, 120]
        coord_WT_01  = [573295.0, 7019569.0, 82]
        coord_WT_02  = [573295.0, 7019569.0, 82]
        coord_WT_03  = [573295.0, 7019569.0, 48]
        coord_WT_04  = [573295.0, 7019569.0, 48]
    if type_mastro == 2:
        coord_Mastro = [573295.0, 7019569.0, 82]
        coord_WT_01  = [573295.0, 7019569.0, 120]
        coord_WT_02  = [573295.0, 7019569.0, 120]
        coord_WT_03  = [573295.0, 7019569.0, 48]
        coord_WT_04  = [573295.0, 7019569.0, 48]
    if type_mastro == 3:
        coord_Mastro = [573295.0, 7019569.0, 48]
        coord_WT_01  = [573295.0, 7019569.0, 120]
        coord_WT_02  = [573295.0, 7019569.0, 120]
        coord_WT_03  = [573295.0, 7019569.0, 82]
        coord_WT_04  = [573295.0, 7019569.0, 82]
else:
```

```
67      coord_Mastro = [573295.0, 7019569.0, 120]
68      coord_WT_01  = [575616.7,7018255.5,166]
69      coord_WT_02  = [575774.7,7017685.0,166]
70      coord_WT_03  = [576125.2,7017393.5,166]
71      coord_WT_04  = [576652.6,7017091.0,166]
72
73  # »»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»
74  # Grid of points
75
76  # Control volumes
77  nx_points = cells_x_grid+1
78  ny_points = cells_y_grid+1
79
80  # High resolution central area
81  cof_dxn = dxn/cells_x_grid
82  cof_dyn = dyn/cells_y_grid
83
84  # Variabels
85  dim = nx_points*ny_points
86  Grid_coordinate_x = np.zeros(dim)
87  Grid_coordinate_y = np.zeros(dim)
88  Grid_U_avg_Synthesis = np.zeros(dim)
89  Grid_A_weibull_Synthesis = np.zeros(dim)
90  Grid_k_weibull_Synthesis = np.zeros(dim)
91  Grid_AEP_Synthesis = np.zeros(dim)
92  Grid_I_Synthesis = np.zeros(dim)
93
94  # Coordinates for points
95  for i in range (0,dim):
96      Grid_coordinate_x[i] = -(dxn/2.0) + cof_dxn*(i-nx_points*np.trunc(i/nx_points))
97      Grid_coordinate_y[i] = -(dyn/2.0) + cof_dyn*np.trunc(i/nx_points)
98
99  # »»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»
100 # Global structure for to save data for every wind direction
101
102 class wind_data:
103     def __init__(self,Dir,x,y,z,zm,U,Ux,Uy,k,U_Vh,SF,FI,
104                  wind_shear_WT_01,wind_shear_WT_02,wind_shear_WT_03,wind_shear_WT_04,
105                  flow_inclination_WT_01,flow_inclination_WT_02,flow_inclination_WT_03,
106                  flow_inclination_WT_04,TI,RC,RCx,RCy,delta_TKE,delta_TI,
107                  int_U,int_Ux,int_Uy,int_TI,perfil_mastro_U,perfil_mastro_Uh,
108                  perfil_mastro_Uh_star,perfil_mastro_k,perfil_mastro_k_star,
109                  perfil_mastro_TI,perfil_mastro_TI_star,perfil_mastro_zagl,
110                  Grid_interpolated_Ux,Grid_interpolated_Uy,Grid_interpolated_U,
111                  Grid_interpolated_k,Grid_interpolated_TI,Grid_RCx,Grid_RCy,Grid_RC):
112         self.Dir = Dir
113         self.x_geographic = x
114         self.y_geographic = y
115         self.z_agl = z
116         self.zm = zm
117         self.U = U
118         self.Ux = Ux
119         self.Uy = Uy
120         self.k = k
121         self.U_Vh = U_Vh
122         self.SF = SF
123         self.FI = FI
124         self.wind_shear_WT_01 = wind_shear_WT_01
125         self.wind_shear_WT_02 = wind_shear_WT_02
126         self.wind_shear_WT_03 = wind_shear_WT_03
127         self.wind_shear_WT_04 = wind_shear_WT_04
128         self.flow_inclination_WT_01 = flow_inclination_WT_01
129         self.flow_inclination_WT_02 = flow_inclination_WT_02
130         self.flow_inclination_WT_03 = flow_inclination_WT_03
131         self.flow_inclination_WT_04 = flow_inclination_WT_04
132         self.TI = TI
133         self.RC = RC
134         self.RCx = RCx
135         self.RCy = RCy
136         self.delta_TKE = delta_TKE
137         self.delta_TI = delta_TI
138         self.int_U = int_U
139         self.int_Ux = int_Ux
140         self.int_Uy = int_Uy
141         self.int_TI = int_TI
142         self.perfil_mastro_U = perfil_mastro_U
143         self.perfil_mastro_Uh = perfil_mastro_Uh
144         self.perfil_mastro_Uh_star = perfil_mastro_Uh_star
145         self.perfil_mastro_k = perfil_mastro_k
146         self.perfil_mastro_k_star = perfil_mastro_k_star
147         self.perfil_mastro_TI = perfil_mastro_TI
148         self.perfil_mastro_TI_star = perfil_mastro_TI_star
149         self.perfil_mastro_zagl = perfil_mastro_zagl
150         self.Grid_interpolated_Ux = Grid_interpolated_Ux
151         self.Grid_interpolated_Uy = Grid_interpolated_Uy
```

```
        self.Grid_interpolated_U  = Grid_interpolated_U
        self.Grid_interpolated_k = Grid_interpolated_k
        self.Grid_interpolated_TI = Grid_interpolated_TI
        self.Grid_RCx = Grid_RCx
        self.Grid_RCy = Grid_RCy
        self.Grid_RC = Grid_RC

# »»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»
# Functions

# Weibull parameters A, k
def Weibull_parameters_A_k(input_array_U):

    cont = 0
    array_U = []
    for i in range(0,len(input_array_U)):
        if input_array_U[i] != 99.99:
            array_U.append(input_array_U[i])
            cont=cont+1

    U_avg = np.mean(array_U)
    print("U_avg =",U_avg)
    Somamin = pow(1,12)
    A_min = round(U_avg,0)-0.4* round(U_avg,0)
    A_max = round(U_avg,0)+0.4* round(U_avg,0)
    contPot = 0
    contSupMed = 0

    for i in range(0,len(array_U)):
        contPot = contPot + pow(array_U[i],3)
        if array_U[i] > U_avg:
            contSupMed = contSupMed + 1

    for A in np.arange(A_min,A_max,0.01):
        for k in np.arange(0.01,15.0,0.01):
            x = 1+(3/k)
            Gama=gamma(x)*(A**3)
            Fw=math.exp(-((U_avg/A)**k))
            Erro_1= abs((contSupMed/cont)-Fw)/(contSupMed/cont)*100.0
            Erro_2= abs((contPot/cont)-Gama)/(contPot/cont)*100.0
            Soma=Erro_1+Erro_2
            if Soma<Somamin:
                Somamin=Soma
                A_final=A
                k_final=k
    return(A_final,k_final)

# Weibull frequencies
def Weibull_frequencies(input_array_U):

    U_interval = np.zeros(20)
    Freq_by_velocity = np.zeros(20)
    Freq_by_velocity_rel = np.zeros(20)

    for i in range(0,len(input_array_U)):
        posicao = math.trunc(input_array_U[i])
        if input_array_U[i] < 20:
            Freq_by_velocity[posicao] += 1
    Freq_by_velocity_total = sum(Freq_by_velocity)

    for i in range(0,20):
        U_interval[i] = i + 1
        Freq_by_velocity_rel[i]=(Freq_by_velocity[i]/Freq_by_velocity_total)*100.0;
    return(U_interval,Freq_by_velocity_rel)

# Turbulence by speed ranges
def TI_by_speed_ranges(input_array_U,input_array_TI):

    U_by_ranges = np.arange(3, 34, 2)
    TI_by_ranges = np.zeros(16)
    TI_by_ranges_AVG = np.zeros(16)
    records = np.zeros(16)
    Frequency = np.zeros(16)

    for i in range(0,len(input_array_U)):
        for j in range(0,len(TI_by_ranges)):
            if (input_array_U[i] >= (U_by_ranges[j]-1.0) and
                input_array_U[i] < (U_by_ranges[j]+1.0)) and input_array_TI[i] > 0.0:
                TI_by_ranges[j] = TI_by_ranges[j] + input_array_TI[i]
                records[j] += 1
                break

    total_records = np.sum(records)

    for j in range(0,len(TI_by_ranges)):
```

```
237         if records[j] > 0:
238             TI_by_ranges_AVG[j] = TI_by_ranges[j] / records[j]
239             Frequency[j] = records[j] / total_records * 100.0
240         else:
241             TI_by_ranges_AVG[j] = 0.0
242             Frequency[j] = 0.0
243 
244     return(TI_by_ranges_AVG,Frequency)
245 
246 
247 # Convert coordinates to polar coordinates
248 def cart_to_polar(x, y):
249     rho = np.sqrt(x**2 + y**2)
250     theta = np.arctan2(y, x)
251     return(rho, theta)
252 
253 # Convert polar to cartesian coordinates
254 def polar_to_cart(rho, theta):
255     x = rho * np.cos(theta)
256     y = rho * np.sin(theta)
257     return(x, y)
258 
259 # Add the rotation of mast and turbines
260 def Add_rotation(coordenadas):
261     # Converts cartesian coordinates to polar coordinates to undo the mesh rotation
262     r,theta_i = cart_to_polar(coordenadas[0],coordenadas[1])
263 
264     # Adds the rotation to the mesh
265     rot_inc = rot * np.pi / 180.0
266     theta_f = theta_i - rot_inc - (90 * np.pi / 180.0)
267 
268     # Convert polar to cartesian coordinates
269     cota_x_rot,cota_y_rot = polar_to_cart(r,theta_f)
270 
271     output = [cota_x_rot,cota_y_rot,coordenadas[2]]
272     return(output)
273 
274 # Reverse the rotation of mast and turbines
275 def Invert_rotation(coordenadas):
276     # Converts cartesian coordinates to polar coordinates to undo the mesh rotation
277     r,theta_i = cart_to_polar(coordenadas[0],coordenadas[1])
278 
279     # Adds the rotation to the mesh
280     rot_inc = (rot+90) * np.pi / 180.0
281     theta_f = theta_i + rot_inc
282 
283     # Convert polar to cartesian coordinates
284     cota_x_rot,cota_y_rot = polar_to_cart(r,theta_f)
285 
286     output = [cota_x_rot,cota_y_rot,coordenadas[2]]
287     return(output)
288 
289 # 3D interpolation
290 def interpolate_wind(coordenadas,U,xm,ym,z_agl):
291     x = coordenadas[0]
292     y = coordenadas[1]
293     z = coordenadas[2]
294     i_low = 0
295     i_high = 0
296     j_low = 0
297     j_high = 0
298     k_low = 0
299     k_high = 0
300 
301     # "x" coordinate
302     for i in range(0,nx-1):
303         if x >= xm[i,0,0] and x < xm[i+1,0,0]:
304             i_low = i
305             i_high = i+1
306             break
307 
308     # "y" coordinate
309     for j in range(0,ny-1):
310         if y >= ym[0,j,0] and y < ym[0,j+1,0]:
311             j_low = j
312             j_high = j+1
313             break
314 
315     # "z" coordinate
316     for k in range(0,nz-1):
317         if z >= z_agl[0,0,k] and z < z_agl[0,0,k+1]:
318             k_low = k-2
319             k_high = k+3
320             break
321 
```

```
322     # Interpolação da velocidade
323     points_aux = []
324     values_aux = []
325 
326     for i in range(i_low,i_high+1):
327         for j in range(j_low,j_high+1):
328             for k in range(k_low,k_high+1):
329             # for k in range(0,nz-1):
330                 points_aux.append([xm[i,j,k],ym[i,j,k],z_agl[i,j,k]])
331                 values_aux.append(U[i,j,k])
332     result = griddata(points_aux,values_aux,coordenadas,method='linear')
333     return(result)
334 
335 # Rotate velocity vector
336 def Rotate_velocity_vector(Ux,Uy):
337     Ux_rot = Ux
338     Uy_rot = Uy
339     for l in range(0,len(Ux)):
340          # Converts cartesian coordinates to polar coordinates to undo the mesh rotation
341         r,theta_i = cart_to_polar(Ux[l],Uy[l])
342 
343         # Adds the rotation to the mesh
344         rot_inc = -(rot+90) * np.pi / 180.0
345         theta_f = theta_i + rot_inc
346 
347         # Convert polar to cartesian coordinates
348         Ux_rot[l],Uy_rot[l] = polar_to_cart(r,theta_f)
349 
350     output = Ux_rot,Uy_rot
351     return(output)
352 
353 # Interpolation to obtain flow inclination at hub height/rotor coordinates
354 def interpolate_to_point(x_ponto,y_ponto,z_ponto):
355     coordenadas = [x_ponto,y_ponto,z_ponto]
356     i_low = 0; i_high = 0; j_low = 0; j_high = 0; k_low = 0; k_high = 0
357 
358     # "x" coordinate
359     for i in range(0,nx-1):
360         if x_ponto >= (xm[i,0,0]) and x_ponto < (xm[i+1,0,0]):
361             i_low = i
362             i_high = i+1
363             break
364 
365     # "y" coordinate
366     for j in range(0,ny-1):
367         if y_ponto >= (ym[0,j,0]) and y_ponto < (ym[0,j+1,0]):
368             j_low = j
369             j_high = j+1
370             break
371 
372     # "z" coordinate
373     for k in range(0,nz-1):
374         if z_ponto >= (z_agl[0,0,k]) and z_ponto < (z_agl[0,0,k+1]):
375             k_low = k-2
376             k_high = k+1+2
377             break
378 
379     points_x_y_z = []
380     values_Vh_rot = []
381     values_Uz = []
382 
383     for i in range(i_low,i_high+1):
384         for j in range(j_low,j_high+1):
385             for k in range(k_low,k_high+1):
386                 points_x_y_z.append([xm[i,j,k],ym[i,j,k],z_agl[i,j,k]])
387                 values_Vh_rot.append(Magnitude_Vh_rot[i,j,k])
388                 values_Uz.append(Uz[i,j,k])
389 
390     result_Vh_rot = griddata(points_x_y_z,values_Vh_rot,coordenadas,method='linear')
391     result_Uz = griddata(points_x_y_z,values_Uz,coordenadas,method='linear')
392     return(result_Vh_rot[0],result_Uz[0])
393 
394 # Interpolation for each wind rose degree
395 def interpolate_for_each_degree(vector,struct,Dir):
396     for i in range(0,360):
397         if i >=0 and i <30:
398             vector[i]=struct[0]-(((struct[0]-struct[1])*(0-i))/(0-30))
399         if i >=Dir[1] and i <Dir[2]:
400             vector[i]=struct[1]-(((struct[1]-struct[2])*(Dir[1]-i))/(Dir[1]-Dir[2]))
401         if i >=Dir[2] and i <Dir[3]:
402             vector[i]=struct[2]-(((struct[2]-struct[3])*(Dir[2]-i))/(Dir[2]-Dir[3]))
403         if i >=Dir[3] and i <Dir[4]:
404             vector[i]=struct[3]-(((struct[3]-struct[4])*(Dir[3]-i))/(Dir[3]-Dir[4]))
405         if i >=Dir[4] and i <Dir[5]:
406             vector[i]=struct[4]-(((struct[4]-struct[5])*(Dir[4]-i))/(Dir[4]-Dir[5]))
```

```
407         if i >=Dir[5] and i <Dir[6]:
408             vector[i]=struct[5]-(((struct[5]-struct[6])*(Dir[5]-i))/(Dir[5]-Dir[6]))
409         if i >=Dir[6] and i <Dir[7]:
410             vector[i]=struct[6]-(((struct[6]-struct[7])*(Dir[6]-i))/(Dir[6]-Dir[7]))
411         if i >=Dir[7] and i <Dir[8]:
412             vector[i]=struct[7]-(((struct[7]-struct[8])*(Dir[7]-i))/(Dir[7]-Dir[8]))
413         if i >=Dir[8] and i <Dir[9]:
414             vector[i]=struct[8]-(((struct[8]-struct[9])*(Dir[8]-i))/(Dir[8]-Dir[9]))
415         if i >=Dir[9] and i <Dir[10]:
416             vector[i]=struct[9]-(((struct[9]-struct[10])*(Dir[9]-i))/(Dir[9]-Dir[10]))
417         if i >=Dir[10] and i <Dir[11]:
418             vector[i]=struct[10]-(((struct[10]-struct[11])*(Dir[10]-i))/(Dir[10]-Dir[11]))
419         if i >=330 and i <360:
420             vector[i]=struct[11]-(((struct[11]-struct[0])*(330-i))/(330-360))
421     return(vector)
422 
423 # Converts string to char
424 def split(word):
425     return [char for char in word]
426 
427 # Converts datetime to string
428 def DT_to_string(array_char):
429     ano = array_char[0] + array_char[1] + array_char[2] + array_char[3]
430     mes = array_char[4] + array_char[5]
431     dia = array_char[6] + array_char[7]
432     result = ano + '-' + mes + '-' + dia
433     return(result)
434 
435 # Calculates wind flow direction using velocity components
436 def get_flow_direction(Ux,Uy):
437     Flow_dir = 0.0
438     # 1ž quadrant
439     if Ux <= 0.0 and Uy < 0.0:
440         Flow_dir = np.arctan2(-Ux,-Uy)*180.0/np.pi
441     # 2ž quadrant
442     if Ux > 0.0 and Uy <= 0.0:
443         Flow_dir = 180.0-np.arctan2(-Ux,Uy)*180.0/np.pi
444     # 3ž quadrant
445     if Ux >= 0.0 and Uy > 0.0:
446         Flow_dir = np.arctan2(-Ux,-Uy)*180.0/np.pi+360.0
447     # 4ž quadrant
448     if Ux < 0.0 and Uy >= 0.0:
449         Flow_dir = np.arctan2(-Ux,-Uy)*180.0/np.pi
450     return(Flow_dir)
451 
452 # Converts number of days between two dates
453 def days_between(d1, d2):
454     d1 = datetime.strptime(d1, "%Y-%m-%d")
455     d2 = datetime.strptime(d2, "%Y-%m-%d")
456     return abs((d2 - d1).days)
457 
458 # Interpolates nominal power for a given wind speed
459 def interpolate_power(velocidade,WindSpeed,PowerOutput):
460     index = 0
461     for i in range(0,cont-1):
462         if velocidade >= WindSpeed[i] and velocidade < WindSpeed[i+1]:
463             index = i
464     result =
465     PowerOutput[index]-((PowerOutput[index]-PowerOutput[index+1])/
466     (WindSpeed[index]-WindSpeed[index+1]))*(WindSpeed[index]-velocidade)
467     return(result)
468 
469 # »»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»
470 # Synthesis algorithm
471 
472 # Fluidfoam mesh reader
473 start = time.time()
474 
475 folders = ['000','030','060','090','120','150','180','210','240','270','300','330']
476 diretorios = ['','','','','','','','','','','','']
477 
478 for i in range(0,len(diretorios)):
479     diretorios[i]=path+folders[i]
480 
481 icano = 0
482 hcano = 0
483 nzcano = 0
484 
485 for s in range(s_inicial,s_final,1):
486 
487     start_parcial = time.time()
488     resultados = 'res_final'
489     diretorio = diretorios[s]
490     print("------------------------------------------------------")
491     print("Diretório em processamento: ",diretorio)
```

```
492     print("")
493     x, y, z = readmesh(diretorio)
494     vel = readvector(diretorio, resultados, 'U', False)
495     k = readscalar(diretorio, resultados, 'k', False)
496 
497     # Looks for rotation, x_enter and ycenter
498     a = ''
499     file = open(diretorio + "/_preproc/_gsurf_WINDIE/" + "preproc.cfg", "r")
500 
501     for line in file.readlines():
502         a = file.readline()
503         leitura = line.split()
504         if 'rot' in leitura:
505             rot = float(leitura[2])
506         if 'xcentre' in leitura:
507             xcenter = float(leitura[2])
508         if 'ycentre' in leitura:
509             ycenter = float(leitura[2])
510     file.close()
511 
512     # Looks for nig and njg
513     a = ''
514     file = open(diretorio + "/_preproc/_gsurf_WINDIE/" + "windie.cfg", "r")
515 
516     for line in file.readlines():
517         a = file.readline()
518         leitura = line.split()
519         if 'nig' in leitura:
520             nx = int(leitura[2])-2
521         if 'njg' in leitura:
522             ny = int(leitura[2])-2
523     file.close()
524 
525     # Looks for nz and canopy parameters
526     a = ''
527     file = open(diretorio + "/_preproc/" + "_preprocDict", "r")
528 
529     for line in file.readlines():
530         a = file.readline()
531         leitura = line.split()
532         if 'nz' in leitura:
533             nz = int(leitura[2])
534         if 'icano' in leitura:
535             icano = int(leitura[2])
536         if 'hcano' in leitura:
537             hcano = float(leitura[2])
538         if 'nzcano' in leitura:
539             nzcano = int(leitura[2])
540     file.close()
541 
542     print("nx,ny,nz =",nx,"x",ny,"x",nz)
543     print("icano =",icano)
544     print("hcano =",hcano)
545     print("nzcano =",nzcano)
546 
547     # Matrix operations to reorder matrices
548     if newCanoMesh == 1:
549         npoints = nx*ny*nz
550         np_below_cano = nx*ny*nzcano
551         np_above_cano = npoints-np_below_cano
552 
553         # "x" coordinate
554         xm_below_cano = np.reshape(x[0:np_below_cano], (nx,ny,nzcano))
555         xm_above_cano = np.reshape(x[np_below_cano:npoints], (nx,ny,nz-nzcano))
556         xm = np.concatenate((xm_below_cano,xm_above_cano),axis=2)
557         xm = np.rot90(np.flip(xm))
558 
559         # "y" coordinate
560         ym_below_cano = np.reshape(y[0:np_below_cano], (nx,ny,nzcano))
561         ym_above_cano = np.reshape(y[np_below_cano:npoints], (nx,ny,nz-nzcano))
562         ym = np.concatenate((ym_below_cano,ym_above_cano),axis=2)
563         ym = np.rot90(np.flip(ym),k=-1)
564 
565         # "z" coordinate
566         zm_below_cano = np.reshape(z[0:np_below_cano], (nx,ny,nzcano))
567         zm_above_cano = np.reshape(z[np_below_cano:npoints], (nx,ny,nz-nzcano))
568         zm = np.concatenate((zm_below_cano,zm_above_cano),axis=2)
569         zm = np.array([np.transpose(zm[:, :, i]) for i in range(zm.shape[2])])
570         zm = zm.transpose(1, 2, 0)
571 
572         # Velocity component "Ux"
573         Ux_below_cano = np.reshape(vel[0,0:np_below_cano], (nx,ny,nzcano))
574         Ux_above_cano = np.reshape(vel[0,np_below_cano:npoints], (nx,ny,nz-nzcano))
575         Ux = np.concatenate((Ux_below_cano,Ux_above_cano),axis=2)
576         Ux = np.array([np.transpose(Ux[:, :, i]) for i in range(Ux.shape[2])])
```

```
577         Ux = Ux.transpose(1, 2, 0)
578 
579         # Velocity component "Uy"
580         Uy_below_cano = np.reshape(vel[1,0:np_below_cano], (nx,ny,nzcano))
581         Uy_above_cano = np.reshape(vel[1,np_below_cano:npoints], (nx,ny,nz-nzcano))
582         Uy = np.concatenate((Uy_below_cano,Uy_above_cano),axis=2)
583         Uy = np.array([np.transpose(Uy[:, :, i]) for i in range(Uy.shape[2])])
584         Uy = Uy.transpose(1, 2, 0)
585 
586         # Velocity component "Uz"
587         Uz_below_cano = np.reshape(vel[2,0:np_below_cano], (nx,ny,nzcano))
588         Uz_above_cano = np.reshape(vel[2,np_below_cano:npoints], (nx,ny,nz-nzcano))
589         Uz = np.concatenate((Uz_below_cano,Uz_above_cano),axis=2)
590         Uz = np.array([np.transpose(Uz[:, :, i]) for i in range(Uz.shape[2])])
591         Uz = Uz.transpose(1, 2, 0)
592 
593         # k
594         ki_below_cano = np.reshape(k[0:np_below_cano], (nx,ny,nzcano))
595         ki_above_cano = np.reshape(k[np_below_cano:npoints], (nx,ny,nz-nzcano))
596         ki = np.concatenate((ki_below_cano,ki_above_cano),axis=2)
597         ki = np.array([np.transpose(ki[:, :, i]) for i in range(ki.shape[2])])
598         ki = ki.transpose(1, 2, 0)
599     else:
600         xm = np.reshape(x, (nx,ny,nz))
601         ym = np.reshape(y, (nx,ny,nz))
602         zm = np.reshape(z, (nx,ny,nz))
603         Ux = np.reshape(vel[0,:], (nx,ny,nz))
604         Uy = np.reshape(vel[1,:], (nx,ny,nz))
605         Uz = np.reshape(vel[2,:], (nx,ny,nz))
606         ki = np.reshape(k, (nx,ny,nz))
607 
608     # Declaration of arrays
609     xm_rot = np.array(xm)
610     ym_rot = np.array(ym)
611     xm_geographic = np.array(xm)
612     ym_geographic = np.array(ym)
613     z_agl = np.array(zm)
614     Ux_rot = np.array(Ux)
615     Uy_rot = np.array(Uy)
616     Magnitude_U = np.array(Ux)
617     Magnitude_Vh = np.array(Ux)
618     Magnitude_Vh_rot = np.array(Ux)
619     Dir_vento = np.array(Ux)
620     SF = np.array(Ux)
621     FI = np.array(Ux)
622     TI = np.array(Ux)
623     points = np.array(Ux)
624     coord_Mastro_m = np.array(coord_Mastro)
625     coord_WT_01_m = np.array(coord_WT_01)
626     coord_WT_02_m = np.array(coord_WT_02)
627     coord_WT_03_m = np.array(coord_WT_03)
628     coord_WT_04_m = np.array(coord_WT_04)
629 
630     # Relative coordinates (not geographic)
631     coord_Mastro_m[0]=coord_Mastro[0]-xcenter
632     coord_WT_01_m[0]=coord_WT_01[0]-xcenter
633     coord_WT_02_m[0]=coord_WT_02[0]-xcenter
634     coord_WT_03_m[0]=coord_WT_03[0]-xcenter
635     coord_WT_04_m[0]=coord_WT_04[0]-xcenter
636 
637     coord_Mastro_m[1]=coord_Mastro[1]-ycenter
638     coord_WT_01_m[1]=coord_WT_01[1]-ycenter
639     coord_WT_02_m[1]=coord_WT_02[1]-ycenter
640     coord_WT_03_m[1]=coord_WT_03[1]-ycenter
641     coord_WT_04_m[1]=coord_WT_04[1]-ycenter
642 
643     coord_Mastro_m_rot = Invert_rotation(coord_Mastro_m)
644     coord_WT_01_m_rot = Invert_rotation(coord_WT_01_m)
645     coord_WT_02_m_rot = Invert_rotation(coord_WT_02_m)
646     coord_WT_03_m_rot = Invert_rotation(coord_WT_03_m)
647     coord_WT_04_m_rot = Invert_rotation(coord_WT_04_m)
648 
649     # Post-processing
650     print("Mesh calculations...")
651     cont = 0
652     i = 0
653     j = 0
654     k = 0
655 
656     for i in range(0,nx):
657         for j in range(0,ny):
658             for k in range(0,nz):
659 
660                 # Reads coord
661                 cota_x = xm[i,j,k]
```

```
                cota_y = ym[i,j,k]
                z_agl[i,j,k] = float(zm[i,j,k]) - float(zm[i,j,0])

                # Convert cartesian into polar coordinates to undo the rotation
                r,theta_i = cart_to_polar(cota_x,cota_y)

                # Adds the rotation
                rot_inc = rot * np.pi / 180.0
                theta_f = theta_i - rot_inc - (90 * np.pi / 180.0)

                # Convert polar into cartesian coordinates
                cota_x_rot,cota_y_rot = polar_to_cart(r,theta_f)

                # Saves the corrected coordinates in the new matrix
                xm_rot[i,j,k] = cota_x_rot
                ym_rot[i,j,k] = cota_y_rot

                # Adds geographic coordinates base on the center of the computacional domain
                xm_geographic[i,j,k] = xm_rot[i,j,k] + xcenter
                ym_geographic[i,j,k] = ym_rot[i,j,k] + ycenter

                # Calculates velocity magnitude
                Ux_2 = pow(Ux[i,j,k],2)
                Uy_2 = pow(Uy[i,j,k],2)
                Uz_2 = pow(Uz[i,j,k],2)
                result = np.sqrt(Ux_2 + Uy_2 + Uz_2)
                Magnitude_U[i,j,k] = result
                Magnitude_Vh[i,j,k] = np.sqrt(Ux_2 + Uy_2)

                # Calculates the horizontal velocity after reverse the mesh rotation
                r_U,theta_i_U = cart_to_polar(Ux[i,j,k],Uy[i,j,k])
                theta_f_U = theta_i_U - rot_inc - (90 * np.pi / 180.0)
                Ux_f,Uy_f = polar_to_cart(r_U,theta_f_U)
                Ux_rot[i,j,k] = Ux_f
                Uy_rot[i,j,k] = Uy_f
                Ux_2_rot = pow(Ux_rot[i,j,k],2)
                Uy_2_rot = pow(Uy_rot[i,j,k],2)
                Magnitude_Vh_rot[i,j,k] = np.sqrt(Ux_2_rot + Uy_2_rot)

                # Calculates wind direction
                Ux_temp = - (Ux_rot[i,j,k])
                Uy_temp = - (Uy_rot[i,j,k])
                r_temp,alpha = cart_to_polar(Ux_temp,Uy_temp)
                Dir_vento[i,j,k] = 90.0 - (alpha * 180.0 / np.pi)

                # Calculates wind sehar factor --> 0 < SF < 0.2
                if k != (nz-1):
                    delta_U = Magnitude_Vh_rot[i,j,k+1] / Magnitude_Vh_rot[i,j,k]
                    delta_z = zm[i,j,k+1] / zm[i,j,k]
                    SF[i,j,k] = np.log(delta_U) / np.log(delta_z)
                else:
                    SF[i,j,k] = 0.0

                # Calculates flow inclination
                phi = np.arctan2(Uz[i,j,k], Magnitude_Vh_rot[i,j,k])
                FI[i,j,k] = phi * 180.0 / np.pi

                # Calculates TI Assuming isotropy (equal entropy in all directions)
                TI[i,j,k] = np.sqrt((2/3)*ki[i,j,k])/Magnitude_Vh_rot[i,j,k]*100.0

                cont = cont + 1

    # Interpolations to mast and turbine coordinates
    print("Interpolations...")

    interpolate_U = np.zeros(5)
    interpolate_Ux = np.zeros(5)
    interpolate_Uy = np.zeros(5)
    interpolate_Ux_aux = np.zeros(5)
    interpolate_Uy_aux = np.zeros(5)
    interpolate_TKE = np.zeros(5)
    interpolate_TI = np.zeros(5)
    interpolate_perfil_mastro_U = np.zeros(nz)
    interpolate_perfil_mastro_Uh = np.zeros(nz)
    interpolate_perfil_mastro_Uh_star = np.zeros(nz)
    interpolate_perfil_mastro_k = np.zeros(nz)
    interpolate_perfil_mastro_k_star = np.zeros(nz)
    interpolate_perfil_mastro_TI = np.zeros(nz)
    interpolate_perfil_mastro_TI_star = np.zeros(nz)
    interpolate_perfil_mastro_zagl = np.zeros(nz)

    RC = np.zeros(5)
    RCx = np.zeros(5)
    RCy = np.zeros(5)
    delta_TKE = np.zeros(5)
```

```
747     delta_TI = np.zeros(5)
748     transported_TI = np.zeros(5)
749 
750     # 0 : MASTRO
751     # 1 : TURBINA WT_01
752     # 2 : TURBINA WT_02
753     # 3 : TURBINA WT_03
754     # 4 : TURBINA WT_04
755 
756     # Ux
757     interpolate_Ux_aux[0] = interpolate_wind(coord_Mastro_m_rot,Ux,xm,ym,z_agl)
758     interpolate_Ux_aux[1] = interpolate_wind(coord_WT_01_m_rot,Ux,xm,ym,z_agl)
759     interpolate_Ux_aux[2] = interpolate_wind(coord_WT_02_m_rot,Ux,xm,ym,z_agl)
760     interpolate_Ux_aux[3] = interpolate_wind(coord_WT_03_m_rot,Ux,xm,ym,z_agl)
761     interpolate_Ux_aux[4] = interpolate_wind(coord_WT_04_m_rot,Ux,xm,ym,z_agl)
762 
763     # Uy
764     interpolate_Uy_aux[0] = interpolate_wind(coord_Mastro_m_rot,Uy,xm,ym,z_agl)
765     interpolate_Uy_aux[1] = interpolate_wind(coord_WT_01_m_rot,Uy,xm,ym,z_agl)
766     interpolate_Uy_aux[2] = interpolate_wind(coord_WT_02_m_rot,Uy,xm,ym,z_agl)
767     interpolate_Uy_aux[3] = interpolate_wind(coord_WT_03_m_rot,Uy,xm,ym,z_agl)
768     interpolate_Uy_aux[4] = interpolate_wind(coord_WT_04_m_rot,Uy,xm,ym,z_agl)
769 
770     # TKE
771     interpolate_TKE[0] = interpolate_wind(coord_Mastro_m_rot,ki,xm,ym,z_agl)
772     interpolate_TKE[1] = interpolate_wind(coord_WT_01_m_rot,ki,xm,ym,z_agl)
773     interpolate_TKE[2] = interpolate_wind(coord_WT_02_m_rot,ki,xm,ym,z_agl)
774     interpolate_TKE[3] = interpolate_wind(coord_WT_03_m_rot,ki,xm,ym,z_agl)
775     interpolate_TKE[4] = interpolate_wind(coord_WT_04_m_rot,ki,xm,ym,z_agl)
776 
777      # TI (Turbulence intensity)
778     interpolate_TI[0] = interpolate_wind(coord_Mastro_m_rot,TI,xm,ym,z_agl)
779     interpolate_TI[1] = interpolate_wind(coord_WT_01_m_rot,TI,xm,ym,z_agl)
780     interpolate_TI[2] = interpolate_wind(coord_WT_02_m_rot,TI,xm,ym,z_agl)
781     interpolate_TI[3] = interpolate_wind(coord_WT_03_m_rot,TI,xm,ym,z_agl)
782     interpolate_TI[4] = interpolate_wind(coord_WT_04_m_rot,TI,xm,ym,z_agl)
783 
784     # Interpolate to GRID OF POINTS
785     Grid_interpolated_Ux_aux = np.zeros(dim)
786     Grid_interpolated_Uy_aux = np.zeros(dim)
787     Grid_interpolated_Ux = np.zeros(dim)
788     Grid_interpolated_Uy = np.zeros(dim)
789     Grid_interpolated_U  = np.zeros(dim)
790     Grid_interpolated_k  = np.zeros(dim)
791     Grid_interpolated_TI = np.zeros(dim)
792     Grid_RCx = np.zeros(dim)
793     Grid_RCy = np.zeros(dim)
794     Grid_RC = np.zeros(dim)
795 
796     for i in range (0,dim):
797         coordinates = [Grid_coordinate_x[i],Grid_coordinate_y[i],zagl_grid]
798         Grid_interpolated_Ux_aux[i] = interpolate_wind(coordinates,Ux,xm,ym,z_agl)
799         Grid_interpolated_Uy_aux[i] = interpolate_wind(coordinates,Uy,xm,ym,z_agl)
800         Grid_interpolated_k[i]  = interpolate_wind(coordinates,ki,xm,ym,z_agl)
801         Grid_interpolated_TI[i]   = interpolate_wind(coordinates,TI,xm,ym,z_agl)
802 
803     interpolate_Ux,interpolate_Uy =
804     Rotate_velocity_vector(interpolate_Ux_aux,interpolate_Uy_aux)
805 
806     Grid_interpolated_Ux,Grid_interpolated_Uy =
807     Rotate_velocity_vector(Grid_interpolated_Ux_aux,Grid_interpolated_Uy_aux)
808 
809     for i in range (0,dim):
810         Grid_interpolated_U[i]  =
811         np.sqrt(pow(Grid_interpolated_Ux[i],2)+pow(Grid_interpolated_Uy[i],2))
812 
813     # U
814     interpolate_U[0] = np.sqrt(pow(interpolate_Ux[0],2)+pow(interpolate_Uy[0],2))
815     interpolate_U[1] = np.sqrt(pow(interpolate_Ux[1],2)+pow(interpolate_Uy[1],2))
816     interpolate_U[2] = np.sqrt(pow(interpolate_Ux[2],2)+pow(interpolate_Uy[2],2))
817     interpolate_U[3] = np.sqrt(pow(interpolate_Ux[3],2)+pow(interpolate_Uy[3],2))
818     interpolate_U[4] = np.sqrt(pow(interpolate_Ux[4],2)+pow(interpolate_Uy[4],2))
819 
820     # Vertical profiles at mast coordinates
821     x_mastro = coord_Mastro[0]
822     y_mastro = coord_Mastro[1]
823     i_low = 0; i_high = 0; j_low = 0; j_high = 0
824 
825     # "x" coordinate
826     for i in range(0,nx-1):
827         if x_mastro >= (xm[i,0,0]+xcenter) and x_mastro < (xm[i+1,0,0]+xcenter):
828             i_low = i
829             i_high = i+1
830             break
831     # "y" coordinate
```

```
    for j in range(0,ny-1):
        if y_mastro >= (ym[0,j,0]+ycenter) and y_mastro < (ym[0,j+1,0]+ycenter):
            j_low = j
            j_high = j+1
            break

    # Interpolate to each index in "z" direction
    for k in range(0,nz):
        points_perfil = []
        values_perfil_U = []
        values_perfil_Uh = []
        values_perfil_k = []
        values_perfil_TI = []
        values_perfil_zagl = []

        for i in range(i_low,i_high+1):
            for j in range(j_low,j_high+1):
                points_perfil.append([xm[i,j,k]+xcenter,ym[i,j,k]+ycenter])
                values_perfil_U.append(Magnitude_U[i,j,k])
                values_perfil_Uh.append(Magnitude_Vh[i,j,k])
                values_perfil_k.append(ki[i,j,k])
                values_perfil_TI.append(TI[i,j,k])
                values_perfil_zagl.append(z_agl[i,j,k])

        coordenadas = [coord_Mastro[0],coord_Mastro[1]]

        result_U = griddata(points_perfil,values_perfil_U,coordenadas,method='linear')
        result_Uh = griddata(points_perfil,values_perfil_Uh,coordenadas,method='linear')
        result_k = griddata(points_perfil,values_perfil_k,coordenadas,method='linear')
        result_TI = griddata(points_perfil,values_perfil_TI,coordenadas,method='linear')
        result_z_agl = griddata(points_perfil,values_perfil_zagl,coordenadas,method='linear')


        interpolate_perfil_mastro_U[k] = result_U
        interpolate_perfil_mastro_Uh[k] = result_Uh
        interpolate_perfil_mastro_k[k] = result_k
        interpolate_perfil_mastro_TI[k] = result_TI
        interpolate_perfil_mastro_zagl[k] = result_z_agl

    # Auxiliary point to non-dimensional profile
    new_zagl = 82

    index = 0
    for i in range(0,len(interpolate_perfil_mastro_zagl)):
        if int(interpolate_perfil_mastro_zagl[i]) >= 82:
            index = i
            break

    for i in range(0,len(interpolate_perfil_mastro_Uh_star)):
        interpolate_perfil_mastro_Uh_star[i] =
        interpolate_perfil_mastro_Uh[i]/interpolate_perfil_mastro_Uh[index]
        interpolate_perfil_mastro_k_star[i] =
        interpolate_perfil_mastro_k[i]/interpolate_perfil_mastro_k[index]
        interpolate_perfil_mastro_TI_star[i] =
        interpolate_perfil_mastro_TI[i]/interpolate_perfil_mastro_TI[index]

    Rt = 0.5*blade_diameter

    # Interpolataes to points
    if sinthesys_mastro == 1:
        data_point_WT_01_rotor           = [0.0,0.0]
        data_point_WT_01_top_of_blade    = [0.0,0.0]
        data_point_WT_01_bottom_of_blade = [0.0,0.0]

        data_point_WT_02_rotor           = [0.0,0.0]
        data_point_WT_02_top_of_blade    = [0.0,0.0]
        data_point_WT_02_bottom_of_blade = [0.0,0.0]

        data_point_WT_03_rotor           = [0.0,0.0]
        data_point_WT_03_top_of_blade    = [0.0,0.0]
        data_point_WT_03_bottom_of_blade = [0.0,0.0]

        data_point_WT_04_rotor           = [0.0,0.0]
        data_point_WT_04_top_of_blade    = [0.0,0.0]
        data_point_WT_04_bottom_of_blade = [0.0,0.0]
    else:
        data_point_WT_01_rotor           =
        interpolate_to_point(coord_WT_01_m_rot[0],coord_WT_01_m_rot[1],coord_WT_01_m_rot[2])
        data_point_WT_01_top_of_blade    =
        interpolate_to_point(coord_WT_01_m_rot[0],coord_WT_01_m_rot[1],coord_WT_01_m_rot[2]+Rt)
        data_point_WT_01_bottom_of_blade =
        interpolate_to_point(coord_WT_01_m_rot[0],coord_WT_01_m_rot[1],coord_WT_01_m_rot[2]-Rt)

        data_point_WT_02_rotor           =
        interpolate_to_point(coord_WT_02_m_rot[0],coord_WT_02_m_rot[1],coord_WT_02_m_rot[2])
```

```
917         data_point_WT_02_top_of_blade    =
918         interpolate_to_point(coord_WT_02_m_rot[0],coord_WT_02_m_rot[1],coord_WT_02_m_rot[2]+Rt)
919         data_point_WT_02_bottom_of_blade =
920         interpolate_to_point(coord_WT_02_m_rot[0],coord_WT_02_m_rot[1],coord_WT_02_m_rot[2]-Rt)
921
922         data_point_WT_03_rotor           =
923         interpolate_to_point(coord_WT_03_m_rot[0],coord_WT_03_m_rot[1],coord_WT_03_m_rot[2])
924         data_point_WT_03_top_of_blade    =
925         interpolate_to_point(coord_WT_03_m_rot[0],coord_WT_03_m_rot[1],coord_WT_03_m_rot[2]+Rt)
926         data_point_WT_03_bottom_of_blade =
927         interpolate_to_point(coord_WT_03_m_rot[0],coord_WT_03_m_rot[1],coord_WT_03_m_rot[2]-Rt)
928
929         data_point_WT_04_rotor           =
930         interpolate_to_point(coord_WT_04_m_rot[0],coord_WT_04_m_rot[1],coord_WT_04_m_rot[2])
931         data_point_WT_04_top_of_blade    =
932         interpolate_to_point(coord_WT_04_m_rot[0],coord_WT_04_m_rot[1],coord_WT_04_m_rot[2]+Rt)
933         data_point_WT_04_bottom_of_blade =
934         interpolate_to_point(coord_WT_04_m_rot[0],coord_WT_04_m_rot[1],coord_WT_04_m_rot[2]-Rt)
935
936     # Calculates wind shear between top and bottom of the blades
937     if sinthesys_mastro == 0:
938         ratio_Magnitude_Vh_rot =
939         data_point_WT_01_top_of_blade[0] / data_point_WT_01_bottom_of_blade[0]
940         ratio_z = (coord_WT_01_m_rot[2]+Rt)/(coord_WT_01_m_rot[2]-Rt)
941         wind_shear_WT_01 = np.log(ratio_Magnitude_Vh_rot) / np.log(ratio_z)
942
943         ratio_Magnitude_Vh_rot =
944         data_point_WT_02_top_of_blade[0] / data_point_WT_02_bottom_of_blade[0]
945         ratio_z = (coord_WT_02_m_rot[2]+Rt)/(coord_WT_02_m_rot[2]-Rt)
946         wind_shear_WT_02 = np.log(ratio_Magnitude_Vh_rot) / np.log(ratio_z)
947
948         ratio_Magnitude_Vh_rot =
949         data_point_WT_03_top_of_blade[0] / data_point_WT_03_bottom_of_blade[0]
950         ratio_z = (coord_WT_03_m_rot[2]+Rt)/(coord_WT_03_m_rot[2]-Rt)
951         wind_shear_WT_03 = np.log(ratio_Magnitude_Vh_rot) / np.log(ratio_z)
952
953         ratio_Magnitude_Vh_rot =
954         data_point_WT_04_top_of_blade[0] / data_point_WT_04_bottom_of_blade[0]
955         ratio_z = (coord_WT_04_m_rot[2]+Rt)/(coord_WT_04_m_rot[2]-Rt)
956         wind_shear_WT_04 = np.log(ratio_Magnitude_Vh_rot) / np.log(ratio_z)
957     else:
958         wind_shear_WT_01 = 0.0
959         wind_shear_WT_02 = 0.0
960         wind_shear_WT_03 = 0.0
961         wind_shear_WT_04 = 0.0
962
963     # Flow inclination across the rotor --> -8ž < FI < +8ž
964     phi_ponto = np.arctan2(data_point_WT_01_rotor[1], data_point_WT_01_rotor[0])
965     flow_inclination_WT_01 = phi_ponto * 180.0 / np.pi
966
967     phi_ponto = np.arctan2(data_point_WT_02_rotor[1], data_point_WT_02_rotor[0])
968     flow_inclination_WT_02 = phi_ponto * 180.0 / np.pi
969
970     phi_ponto = np.arctan2(data_point_WT_03_rotor[1], data_point_WT_03_rotor[0])
971     flow_inclination_WT_03 = phi_ponto * 180.0 / np.pi
972
973     phi_ponto = np.arctan2(data_point_WT_04_rotor[1], data_point_WT_04_rotor[0])
974     flow_inclination_WT_04 = phi_ponto * 180.0 / np.pi
975
976     # Speed-ups between mast and turbines
977     for i in range(0,5):
978         RC[i] = interpolate_U[i] / interpolate_U[0]
979         RCx[i] = interpolate_Ux[i] / interpolate_Ux[0]
980         RCy[i] = interpolate_Uy[i] / interpolate_Uy[0]
981         delta_TKE[i] = interpolate_TKE[0] - interpolate_TKE[i]
982         delta_TI[i] = interpolate_TI[0] - interpolate_TI[i] # MEASURED - INTERPOLATED
983
984     # Grid of points
985     for i in range (0,dim):
986         Grid_RCx[i] = Grid_interpolated_Ux[i] / interpolate_Ux[0]
987         Grid_RCy[i] = Grid_interpolated_Uy[i] / interpolate_Uy[0]
988         Grid_RC[i]  = Grid_interpolated_U[i]  / interpolate_U[0]
989
990
991     # Saves all value in the corresponding structure
992     if s == 0:
993         dir_000 = wind_data(Dir_vento,xm_geographic,ym_geographic,z_agl,zm,Magnitude_U,
994                             Ux_rot,Uy_rot,ki,Magnitude_Vh_rot,SF,FI,
995                             wind_shear_WT_01,wind_shear_WT_02,wind_shear_WT_03,wind_shear_WT_04,
996                             flow_inclination_WT_01,flow_inclination_WT_02,flow_inclination_WT_03,
997                             flow_inclination_WT_04,TI,RC,RCx,RCy,delta_TKE,delta_TI,
998                             interpolate_U,interpolate_Ux,interpolate_Uy,interpolate_TI,
999                             interpolate_perfil_mastro_U,interpolate_perfil_mastro_Uh,
1000                            interpolate_perfil_mastro_Uh_star,interpolate_perfil_mastro_k,
1001                            interpolate_perfil_mastro_k_star,interpolate_perfil_mastro_TI,
```

```
                            interpolate_perfil_mastro_TI_star,interpolate_perfil_mastro_zagl,
                            Grid_interpolated_Ux,Grid_interpolated_Uy,Grid_interpolated_U,
                            Grid_interpolated_k,Grid_interpolated_TI,Grid_RCx,Grid_RCy,Grid_RC)

    if s == 1:
        dir_030 = wind_data(Dir_vento,xm_geographic,ym_geographic,z_agl,zm,Magnitude_U,
                            Ux_rot,Uy_rot,ki,Magnitude_Vh_rot,SF,FI,
                            wind_shear_WT_01,wind_shear_WT_02,wind_shear_WT_03,wind_shear_WT_04,
                            flow_inclination_WT_01,flow_inclination_WT_02,flow_inclination_WT_03,
                            flow_inclination_WT_04,TI,RC,RCx,RCy,delta_TKE,delta_TI,
                            interpolate_U,interpolate_Ux,interpolate_Uy,interpolate_TI,
                            interpolate_perfil_mastro_U,interpolate_perfil_mastro_Uh,
                            interpolate_perfil_mastro_Uh_star,interpolate_perfil_mastro_k,
                            interpolate_perfil_mastro_k_star,interpolate_perfil_mastro_TI,
                            interpolate_perfil_mastro_TI_star,interpolate_perfil_mastro_zagl,
                            Grid_interpolated_Ux,Grid_interpolated_Uy,Grid_interpolated_U,
                            Grid_interpolated_k,Grid_interpolated_TI,Grid_RCx,Grid_RCy,Grid_RC)

    if s == 2:
        dir_060 = wind_data(Dir_vento,xm_geographic,ym_geographic,z_agl,zm,Magnitude_U,
                            Ux_rot,Uy_rot,ki,Magnitude_Vh_rot,SF,FI,
                            wind_shear_WT_01,wind_shear_WT_02,wind_shear_WT_03,wind_shear_WT_04,
                            flow_inclination_WT_01,flow_inclination_WT_02,flow_inclination_WT_03,
                            flow_inclination_WT_04,TI,RC,RCx,RCy,delta_TKE,delta_TI,
                            interpolate_U,interpolate_Ux,interpolate_Uy,interpolate_TI,
                            interpolate_perfil_mastro_U,interpolate_perfil_mastro_Uh,
                            interpolate_perfil_mastro_Uh_star,interpolate_perfil_mastro_k,
                            interpolate_perfil_mastro_k_star,interpolate_perfil_mastro_TI,
                            interpolate_perfil_mastro_TI_star,interpolate_perfil_mastro_zagl,
                            Grid_interpolated_Ux,Grid_interpolated_Uy,Grid_interpolated_U,
                            Grid_interpolated_k,Grid_interpolated_TI,Grid_RCx,Grid_RCy,Grid_RC)

    if s == 3:
        dir_090 = wind_data(Dir_vento,xm_geographic,ym_geographic,z_agl,zm,Magnitude_U,
                            Ux_rot,Uy_rot,ki,Magnitude_Vh_rot,SF,FI,
                            wind_shear_WT_01,wind_shear_WT_02,wind_shear_WT_03,wind_shear_WT_04,
                            flow_inclination_WT_01,flow_inclination_WT_02,flow_inclination_WT_03,
                            flow_inclination_WT_04,TI,RC,RCx,RCy,delta_TKE,delta_TI,
                            interpolate_U,interpolate_Ux,interpolate_Uy,interpolate_TI,
                            interpolate_perfil_mastro_U,interpolate_perfil_mastro_Uh,
                            interpolate_perfil_mastro_Uh_star,interpolate_perfil_mastro_k,
                            interpolate_perfil_mastro_k_star,interpolate_perfil_mastro_TI,
                            interpolate_perfil_mastro_TI_star,interpolate_perfil_mastro_zagl,
                            Grid_interpolated_Ux,Grid_interpolated_Uy,Grid_interpolated_U,
                            Grid_interpolated_k,Grid_interpolated_TI,Grid_RCx,Grid_RCy,Grid_RC)

    if s == 4:
        dir_120 = wind_data(Dir_vento,xm_geographic,ym_geographic,z_agl,zm,Magnitude_U,
                            Ux_rot,Uy_rot,ki,Magnitude_Vh_rot,SF,FI,
                            wind_shear_WT_01,wind_shear_WT_02,wind_shear_WT_03,wind_shear_WT_04,
                            flow_inclination_WT_01,flow_inclination_WT_02,flow_inclination_WT_03,
                            flow_inclination_WT_04,TI,RC,RCx,RCy,delta_TKE,delta_TI,
                            interpolate_U,interpolate_Ux,interpolate_Uy,interpolate_TI,
                            interpolate_perfil_mastro_U,interpolate_perfil_mastro_Uh,
                            interpolate_perfil_mastro_Uh_star,interpolate_perfil_mastro_k,
                            interpolate_perfil_mastro_k_star,interpolate_perfil_mastro_TI,
                            interpolate_perfil_mastro_TI_star,interpolate_perfil_mastro_zagl,
                            Grid_interpolated_Ux,Grid_interpolated_Uy,Grid_interpolated_U,
                            Grid_interpolated_k,Grid_interpolated_TI,Grid_RCx,Grid_RCy,Grid_RC)

    if s == 5:
        dir_150 = wind_data(Dir_vento,xm_geographic,ym_geographic,z_agl,zm,Magnitude_U,
                            Ux_rot,Uy_rot,ki,Magnitude_Vh_rot,SF,FI,
                            wind_shear_WT_01,wind_shear_WT_02,wind_shear_WT_03,wind_shear_WT_04,
                            flow_inclination_WT_01,flow_inclination_WT_02,flow_inclination_WT_03,
                            flow_inclination_WT_04,TI,RC,RCx,RCy,delta_TKE,delta_TI,
                            interpolate_U,interpolate_Ux,interpolate_Uy,interpolate_TI,
                            interpolate_perfil_mastro_U,interpolate_perfil_mastro_Uh,
                            interpolate_perfil_mastro_Uh_star,interpolate_perfil_mastro_k,
                            interpolate_perfil_mastro_k_star,interpolate_perfil_mastro_TI,
                            interpolate_perfil_mastro_TI_star,interpolate_perfil_mastro_zagl,
                            Grid_interpolated_Ux,Grid_interpolated_Uy,Grid_interpolated_U,
                            Grid_interpolated_k,Grid_interpolated_TI,Grid_RCx,Grid_RCy,Grid_RC)

    if s == 6:
        dir_180 = wind_data(Dir_vento,xm_geographic,ym_geographic,z_agl,zm,Magnitude_U,
                            Ux_rot,Uy_rot,ki,Magnitude_Vh_rot,SF,FI,
                            wind_shear_WT_01,wind_shear_WT_02,wind_shear_WT_03,wind_shear_WT_04,
                            flow_inclination_WT_01,flow_inclination_WT_02,flow_inclination_WT_03,
                            flow_inclination_WT_04,TI,RC,RCx,RCy,delta_TKE,delta_TI,
                            interpolate_U,interpolate_Ux,interpolate_Uy,interpolate_TI,
                            interpolate_perfil_mastro_U,interpolate_perfil_mastro_Uh,
                            interpolate_perfil_mastro_Uh_star,interpolate_perfil_mastro_k,
                            interpolate_perfil_mastro_k_star,interpolate_perfil_mastro_TI,
                            interpolate_perfil_mastro_TI_star,interpolate_perfil_mastro_zagl,
```

```
                            Grid_interpolated_Ux,Grid_interpolated_Uy,Grid_interpolated_U,
                            Grid_interpolated_k,Grid_interpolated_TI,Grid_RCx,Grid_RCy,Grid_RC)

    if s == 7:
        dir_210 = wind_data(Dir_vento,xm_geographic,ym_geographic,z_agl,zm,Magnitude_U,
                            Ux_rot,Uy_rot,ki,Magnitude_Vh_rot,SF,FI,
                            wind_shear_WT_01,wind_shear_WT_02,wind_shear_WT_03,wind_shear_WT_04,
                            flow_inclination_WT_01,flow_inclination_WT_02,flow_inclination_WT_03,
                            flow_inclination_WT_04,TI,RC,RCx,RCy,delta_TKE,delta_TI,
                            interpolate_U,interpolate_Ux,interpolate_Uy,interpolate_TI,
                            interpolate_perfil_mastro_U,interpolate_perfil_mastro_Uh,
                            interpolate_perfil_mastro_Uh_star,interpolate_perfil_mastro_k,
                            interpolate_perfil_mastro_k_star,interpolate_perfil_mastro_TI,
                            interpolate_perfil_mastro_TI_star,interpolate_perfil_mastro_zagl,
                            Grid_interpolated_Ux,Grid_interpolated_Uy,Grid_interpolated_U,
                            Grid_interpolated_k,Grid_interpolated_TI,Grid_RCx,Grid_RCy,Grid_RC)

    if s == 8:
        dir_240 = wind_data(Dir_vento,xm_geographic,ym_geographic,z_agl,zm,Magnitude_U,
                            Ux_rot,Uy_rot,ki,Magnitude_Vh_rot,SF,FI,
                            wind_shear_WT_01,wind_shear_WT_02,wind_shear_WT_03,wind_shear_WT_04,
                            flow_inclination_WT_01,flow_inclination_WT_02,flow_inclination_WT_03,
                            flow_inclination_WT_04,TI,RC,RCx,RCy,delta_TKE,delta_TI,
                            interpolate_U,interpolate_Ux,interpolate_Uy,interpolate_TI,
                            interpolate_perfil_mastro_U,interpolate_perfil_mastro_Uh,
                            interpolate_perfil_mastro_Uh_star,interpolate_perfil_mastro_k,
                            interpolate_perfil_mastro_k_star,interpolate_perfil_mastro_TI,
                            interpolate_perfil_mastro_TI_star,interpolate_perfil_mastro_zagl,
                            Grid_interpolated_Ux,Grid_interpolated_Uy,Grid_interpolated_U,
                            Grid_interpolated_k,Grid_interpolated_TI,Grid_RCx,Grid_RCy,Grid_RC)

    if s == 9:
        dir_270 = wind_data(Dir_vento,xm_geographic,ym_geographic,z_agl,zm,Magnitude_U,
                            Ux_rot,Uy_rot,ki,Magnitude_Vh_rot,SF,FI,
                            wind_shear_WT_01,wind_shear_WT_02,wind_shear_WT_03,wind_shear_WT_04,
                            flow_inclination_WT_01,flow_inclination_WT_02,flow_inclination_WT_03,
                            flow_inclination_WT_04,TI,RC,RCx,RCy,delta_TKE,delta_TI,
                            interpolate_U,interpolate_Ux,interpolate_Uy,interpolate_TI,
                            interpolate_perfil_mastro_U,interpolate_perfil_mastro_Uh,
                            interpolate_perfil_mastro_Uh_star,interpolate_perfil_mastro_k,
                            interpolate_perfil_mastro_k_star,interpolate_perfil_mastro_TI,
                            interpolate_perfil_mastro_TI_star,interpolate_perfil_mastro_zagl,
                            Grid_interpolated_Ux,Grid_interpolated_Uy,Grid_interpolated_U,
                            Grid_interpolated_k,Grid_interpolated_TI,Grid_RCx,Grid_RCy,Grid_RC)

    if s == 10:
        dir_300 = wind_data(Dir_vento,xm_geographic,ym_geographic,z_agl,zm,Magnitude_U,
                            Ux_rot,Uy_rot,ki,Magnitude_Vh_rot,SF,FI,
                            wind_shear_WT_01,wind_shear_WT_02,wind_shear_WT_03,wind_shear_WT_04,
                            flow_inclination_WT_01,flow_inclination_WT_02,flow_inclination_WT_03,
                            flow_inclination_WT_04,TI,RC,RCx,RCy,delta_TKE,delta_TI,
                            interpolate_U,interpolate_Ux,interpolate_Uy,interpolate_TI,
                            interpolate_perfil_mastro_U,interpolate_perfil_mastro_Uh,
                            interpolate_perfil_mastro_Uh_star,interpolate_perfil_mastro_k,
                            interpolate_perfil_mastro_k_star,interpolate_perfil_mastro_TI,
                            interpolate_perfil_mastro_TI_star,interpolate_perfil_mastro_zagl,
                            Grid_interpolated_Ux,Grid_interpolated_Uy,Grid_interpolated_U,
                            Grid_interpolated_k,Grid_interpolated_TI,Grid_RCx,Grid_RCy,Grid_RC)

    if s == 11:
        dir_330 = wind_data(Dir_vento,xm_geographic,ym_geographic,z_agl,zm,Magnitude_U,
                            Ux_rot,Uy_rot,ki,Magnitude_Vh_rot,SF,FI,
                            wind_shear_WT_01,wind_shear_WT_02,wind_shear_WT_03,wind_shear_WT_04,
                            flow_inclination_WT_01,flow_inclination_WT_02,flow_inclination_WT_03,
                            flow_inclination_WT_04,TI,RC,RCx,RCy,delta_TKE,delta_TI,
                            interpolate_U,interpolate_Ux,interpolate_Uy,interpolate_TI,
                            interpolate_perfil_mastro_U,interpolate_perfil_mastro_Uh,
                            interpolate_perfil_mastro_Uh_star,interpolate_perfil_mastro_k,
                            interpolate_perfil_mastro_k_star,interpolate_perfil_mastro_TI,
                            interpolate_perfil_mastro_TI_star,interpolate_perfil_mastro_zagl,
                            Grid_interpolated_Ux,Grid_interpolated_Uy,Grid_interpolated_U,
                            Grid_interpolated_k,Grid_interpolated_TI,Grid_RCx,Grid_RCy,Grid_RC)

    end_parcial = time.time()

# »»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»
# Reads velocity at turbine coordinates and combines peed-up ratios

ID = ['00','01','02','03','04','05','06','07','08','09','10','11','12']
Orientacao = ['N','NNE','ENE','E','ESE','SSE','S','SSW','WSW','W','WNW','NNW']
Graus = ['0','30','60','90','120','150','180','210','240','270','300','330']

RC_WT_00=[dir_000.RC[0],dir_030.RC[0],dir_060.RC[0],dir_090.RC[0],dir_120.RC[0],dir_150.RC[0],
          dir_180.RC[0],dir_210.RC[0],dir_240.RC[0],dir_270.RC[0],dir_300.RC[0],dir_330.RC[0]]
RC_WT_01=[dir_000.RC[1],dir_030.RC[1],dir_060.RC[1],dir_090.RC[1],dir_120.RC[1],dir_150.RC[1],
```

```
          dir_180.RC[1],dir_210.RC[1],dir_240.RC[1],dir_270.RC[1],dir_300.RC[1],dir_330.RC[1]]
RC_WT_02=[dir_000.RC[2],dir_030.RC[2],dir_060.RC[2],dir_090.RC[2],dir_120.RC[2],dir_150.RC[2],
          dir_180.RC[2],dir_210.RC[2],dir_240.RC[2],dir_270.RC[2],dir_300.RC[2],dir_330.RC[2]]
RC_WT_03=[dir_000.RC[3],dir_030.RC[3],dir_060.RC[3],dir_090.RC[3],dir_120.RC[3],dir_150.RC[3],
          dir_180.RC[3],dir_210.RC[3],dir_240.RC[3],dir_270.RC[3],dir_300.RC[3],dir_330.RC[3]]
RC_WT_04=[dir_000.RC[4],dir_030.RC[4],dir_060.RC[4],dir_090.RC[4],dir_120.RC[4],dir_150.RC[4],
          dir_180.RC[4],dir_210.RC[4],dir_240.RC[4],dir_270.RC[4],dir_300.RC[4],dir_330.RC[4]]

RCx_WT_01=[dir_000.RCx[1],dir_030.RCx[1],dir_060.RCx[1],dir_090.RCx[1],dir_120.RCx[1],
           dir_150.RCx[1],dir_180.RCx[1],dir_210.RCx[1],dir_240.RCx[1],dir_270.RCx[1],
           dir_300.RCx[1],dir_330.RCx[1]]
RCx_WT_02=[dir_000.RCx[2],dir_030.RCx[2],dir_060.RCx[2],dir_090.RCx[2],dir_120.RCx[2],
           dir_150.RCx[2],dir_180.RCx[2],dir_210.RCx[2],dir_240.RCx[2],dir_270.RCx[2],
           dir_300.RCx[2],dir_330.RCx[2]]
RCx_WT_03=[dir_000.RCx[3],dir_030.RCx[3],dir_060.RCx[3],dir_090.RCx[3],dir_120.RCx[3],
           dir_150.RCx[3],dir_180.RCx[3],dir_210.RCx[3],dir_240.RCx[3],dir_270.RCx[3],
           dir_300.RCx[3],dir_330.RCx[3]]
RCx_WT_04=[dir_000.RCx[4],dir_030.RCx[4],dir_060.RCx[4],dir_090.RCx[4],dir_120.RCx[4],
           dir_150.RCx[4],dir_180.RCx[4],dir_210.RCx[4],dir_240.RCx[4],dir_270.RCx[4],
           dir_300.RCx[4],dir_330.RCx[4]]

RCy_WT_01=[dir_000.RCy[1],dir_030.RCy[1],dir_060.RCy[1],dir_090.RCy[1],dir_120.RCy[1],
           dir_150.RCy[1],dir_180.RCy[1],dir_210.RCy[1],dir_240.RCy[1],dir_270.RCy[1],
           dir_300.RCy[1],dir_330.RCy[1]]
RCy_WT_02=[dir_000.RCy[2],dir_030.RCy[2],dir_060.RCy[2],dir_090.RCy[2],dir_120.RCy[2],
           dir_150.RCy[2],dir_180.RCy[2],dir_210.RCy[2],dir_240.RCy[2],dir_270.RCy[2],
           dir_300.RCy[2],dir_330.RCy[2]]
RCy_WT_03=[dir_000.RCy[3],dir_030.RCy[3],dir_060.RCy[3],dir_090.RCy[3],dir_120.RCy[3],
           dir_150.RCy[3],dir_180.RCy[3],dir_210.RCy[3],dir_240.RCy[3],dir_270.RCy[3],
           dir_300.RCy[3],dir_330.RCy[3]]
RCy_WT_04=[dir_000.RCy[4],dir_030.RCy[4],dir_060.RCy[4],dir_090.RCy[4],dir_120.RCy[4],
           dir_150.RCy[4],dir_180.RCy[4],dir_210.RCy[4],dir_240.RCy[4],dir_270.RCy[4],
           dir_300.RCy[4],dir_330.RCy[4]]

U_WT_00=[dir_000.int_U[0],dir_030.int_U[0],dir_060.int_U[0],dir_090.int_U[0],dir_120.int_U[0],
         dir_150.int_U[0],dir_180.int_U[0],dir_210.int_U[0],dir_240.int_U[0],dir_270.int_U[0],
         dir_300.int_U[0],dir_330.int_U[0]]
U_WT_01=[dir_000.int_U[1],dir_030.int_U[1],dir_060.int_U[1],dir_090.int_U[1],dir_120.int_U[1],
         dir_150.int_U[1],dir_180.int_U[1],dir_210.int_U[1],dir_240.int_U[1],dir_270.int_U[1],
         dir_300.int_U[1],dir_330.int_U[1]]
U_WT_02=[dir_000.int_U[2],dir_030.int_U[2],dir_060.int_U[2],dir_090.int_U[2],dir_120.int_U[2],
         dir_150.int_U[2],dir_180.int_U[2],dir_210.int_U[2],dir_240.int_U[2],dir_270.int_U[2],
         dir_300.int_U[2],dir_330.int_U[2]]
U_WT_03=[dir_000.int_U[3],dir_030.int_U[3],dir_060.int_U[3],dir_090.int_U[3],dir_120.int_U[3],
         dir_150.int_U[3],dir_180.int_U[3],dir_210.int_U[3],dir_240.int_U[3],dir_270.int_U[3],
         dir_300.int_U[3],dir_330.int_U[3]]
U_WT_04=[dir_000.int_U[4],dir_030.int_U[4],dir_060.int_U[4],dir_090.int_U[4],dir_120.int_U[4],
         dir_150.int_U[4],dir_180.int_U[4],dir_210.int_U[4],dir_240.int_U[4],dir_270.int_U[4],
         dir_300.int_U[4],dir_330.int_U[4]]

Ux_WT_01=[dir_000.int_Ux[1],dir_030.int_Ux[1],dir_060.int_Ux[1],dir_090.int_Ux[1],
          dir_120.int_Ux[1],dir_150.int_Ux[1],dir_180.int_Ux[1],dir_210.int_Ux[1],
          dir_240.int_Ux[1],dir_270.int_Ux[1],dir_300.int_Ux[1],dir_330.int_Ux[1]]
Ux_WT_02=[dir_000.int_Ux[2],dir_030.int_Ux[2],dir_060.int_Ux[2],dir_090.int_Ux[2],
          dir_120.int_Ux[2],dir_150.int_Ux[2],dir_180.int_Ux[2],dir_210.int_Ux[2],
          dir_240.int_Ux[2],dir_270.int_Ux[2],dir_300.int_Ux[2],dir_330.int_Ux[2]]
Ux_WT_03=[dir_000.int_Ux[3],dir_030.int_Ux[3],dir_060.int_Ux[3],dir_090.int_Ux[3],
          dir_120.int_Ux[3],dir_150.int_Ux[3],dir_180.int_Ux[3],dir_210.int_Ux[3],
          dir_240.int_Ux[3],dir_270.int_Ux[3],dir_300.int_Ux[3],dir_330.int_Ux[3]]
Ux_WT_04=[dir_000.int_Ux[4],dir_030.int_Ux[4],dir_060.int_Ux[4],dir_090.int_Ux[4],
          dir_120.int_Ux[4],dir_150.int_Ux[4],dir_180.int_Ux[4],dir_210.int_Ux[4],
          dir_240.int_Ux[4],dir_270.int_Ux[4],dir_300.int_Ux[4],dir_330.int_Ux[4]]

Uy_WT_01=[dir_000.int_Uy[1],dir_030.int_Uy[1],dir_060.int_Uy[1],dir_090.int_Uy[1],
          dir_120.int_Uy[1],dir_150.int_Uy[1],dir_180.int_Uy[1],dir_210.int_Uy[1],
          dir_240.int_Uy[1],dir_270.int_Uy[1],dir_300.int_Uy[1],dir_330.int_Uy[1]]
Uy_WT_02=[dir_000.int_Uy[2],dir_030.int_Uy[2],dir_060.int_Uy[2],dir_090.int_Uy[2],
          dir_120.int_Uy[2],dir_150.int_Uy[2],dir_180.int_Uy[2],dir_210.int_Uy[2],
          dir_240.int_Uy[2],dir_270.int_Uy[2],dir_300.int_Uy[2],dir_330.int_Uy[2]]
Uy_WT_03=[dir_000.int_Uy[3],dir_030.int_Uy[3],dir_060.int_Uy[3],dir_090.int_Uy[3],
          dir_120.int_Uy[3],dir_150.int_Uy[3],dir_180.int_Uy[3],dir_210.int_Uy[3],
          dir_240.int_Uy[3],dir_270.int_Uy[3],dir_300.int_Uy[3],dir_330.int_Uy[3]]
Uy_WT_04=[dir_000.int_Uy[4],dir_030.int_Uy[4],dir_060.int_Uy[4],dir_090.int_Uy[4],
          dir_120.int_Uy[4],dir_150.int_Uy[4],dir_180.int_Uy[4],dir_210.int_Uy[4],
          dir_240.int_Uy[4],dir_270.int_Uy[4],dir_300.int_Uy[4],dir_330.int_Uy[4]]

RC_WT_01_plot=[RC_WT_01[3],RC_WT_01[2],RC_WT_01[1],RC_WT_01[0],
               RC_WT_01[11],RC_WT_01[10],RC_WT_01[9],RC_WT_01[8],
               RC_WT_01[7],RC_WT_01[6],RC_WT_01[5],RC_WT_01[4]]

RC_WT_02_plot=[RC_WT_02[3],RC_WT_02[2],RC_WT_02[1],RC_WT_02[0],
               RC_WT_02[11],RC_WT_02[10],RC_WT_02[9],RC_WT_02[8],
               RC_WT_02[7],RC_WT_02[6],RC_WT_02[5],RC_WT_02[4]]

RC_WT_03_plot=[RC_WT_03[3],RC_WT_03[2],RC_WT_03[1],RC_WT_03[0],
```

```
                RC_WT_03[11],RC_WT_03[10],RC_WT_03[9],RC_WT_03[8],
                RC_WT_03[7],RC_WT_03[6],RC_WT_03[5],RC_WT_03[4]]

RC_WT_04_plot=[RC_WT_04[3],RC_WT_04[2],RC_WT_04[1],RC_WT_04[0],
                RC_WT_04[11],RC_WT_04[10],RC_WT_04[9],RC_WT_04[8],
                RC_WT_04[7],RC_WT_04[6],RC_WT_04[5],RC_WT_04[4]]

# Flow inclination
flow_inclination_group_WT_01 =
[dir_000.flow_inclination_WT_01,dir_030.flow_inclination_WT_01,dir_060.flow_inclination_WT_01,
 dir_090.flow_inclination_WT_01,dir_120.flow_inclination_WT_01,dir_150.flow_inclination_WT_01,
 dir_180.flow_inclination_WT_01,dir_210.flow_inclination_WT_01,dir_240.flow_inclination_WT_01,
 dir_270.flow_inclination_WT_01,dir_300.flow_inclination_WT_01,dir_330.flow_inclination_WT_01]

flow_inclination_group_WT_02 =
[dir_000.flow_inclination_WT_02,dir_030.flow_inclination_WT_02,dir_060.flow_inclination_WT_02,
 dir_090.flow_inclination_WT_02,dir_120.flow_inclination_WT_02,dir_150.flow_inclination_WT_02,
 dir_180.flow_inclination_WT_02,dir_210.flow_inclination_WT_02,dir_240.flow_inclination_WT_02,
 dir_270.flow_inclination_WT_02,dir_300.flow_inclination_WT_02,dir_330.flow_inclination_WT_02]

flow_inclination_group_WT_03 =
[dir_000.flow_inclination_WT_03,dir_030.flow_inclination_WT_03,dir_060.flow_inclination_WT_03,
 dir_090.flow_inclination_WT_03,dir_120.flow_inclination_WT_03,dir_150.flow_inclination_WT_03,
 dir_180.flow_inclination_WT_03,dir_210.flow_inclination_WT_03,dir_240.flow_inclination_WT_03,
 dir_270.flow_inclination_WT_03,dir_300.flow_inclination_WT_03,dir_330.flow_inclination_WT_03]

flow_inclination_group_WT_04 =
[dir_000.flow_inclination_WT_04,dir_030.flow_inclination_WT_04,dir_060.flow_inclination_WT_04,
 dir_090.flow_inclination_WT_04,dir_120.flow_inclination_WT_04,dir_150.flow_inclination_WT_04,
 dir_180.flow_inclination_WT_04,dir_210.flow_inclination_WT_04,dir_240.flow_inclination_WT_04,
 dir_270.flow_inclination_WT_04,dir_300.flow_inclination_WT_04,dir_330.flow_inclination_WT_04]

# Wind shear
wind_shear_group_WT_01 =
[dir_000.wind_shear_WT_01,dir_030.wind_shear_WT_01,dir_060.wind_shear_WT_01,
 dir_090.wind_shear_WT_01,dir_120.wind_shear_WT_01,dir_150.wind_shear_WT_01,
 dir_180.wind_shear_WT_01,dir_210.wind_shear_WT_01,dir_240.wind_shear_WT_01,
 dir_270.wind_shear_WT_01,dir_300.wind_shear_WT_01,dir_330.wind_shear_WT_01]

wind_shear_group_WT_02 =
[dir_000.wind_shear_WT_02,dir_030.wind_shear_WT_02,dir_060.wind_shear_WT_02,
 dir_090.wind_shear_WT_02,dir_120.wind_shear_WT_02,dir_150.wind_shear_WT_02,
 dir_180.wind_shear_WT_02,dir_210.wind_shear_WT_02,dir_240.wind_shear_WT_02,
 dir_270.wind_shear_WT_02,dir_300.wind_shear_WT_02,dir_330.wind_shear_WT_02]

wind_shear_group_WT_03 =
[dir_000.wind_shear_WT_03,dir_030.wind_shear_WT_03,dir_060.wind_shear_WT_03,
 dir_090.wind_shear_WT_03,dir_120.wind_shear_WT_03,dir_150.wind_shear_WT_03,
 dir_180.wind_shear_WT_03,dir_210.wind_shear_WT_03,dir_240.wind_shear_WT_03,
 dir_270.wind_shear_WT_03,dir_300.wind_shear_WT_03,dir_330.wind_shear_WT_03]

wind_shear_group_WT_04 =
[dir_000.wind_shear_WT_04,dir_030.wind_shear_WT_04,dir_060.wind_shear_WT_04,
 dir_090.wind_shear_WT_04,dir_120.wind_shear_WT_04,dir_150.wind_shear_WT_04,
 dir_180.wind_shear_WT_04,dir_210.wind_shear_WT_04,dir_240.wind_shear_WT_04,
 dir_270.wind_shear_WT_04,dir_300.wind_shear_WT_04,dir_330.wind_shear_WT_04]

# Turbulence
delta_TKE_WT_00=[dir_000.delta_TKE[0],dir_030.delta_TKE[0],dir_060.delta_TKE[0],
                 dir_090.delta_TKE[0],dir_120.delta_TKE[0],dir_150.delta_TKE[0],
                 dir_180.delta_TKE[0],dir_210.delta_TKE[0],dir_240.delta_TKE[0],
                 dir_270.delta_TKE[0],dir_300.delta_TKE[0],dir_330.delta_TKE[0]]
delta_TKE_WT_01=[dir_000.delta_TKE[1],dir_030.delta_TKE[1],dir_060.delta_TKE[1],
                 dir_090.delta_TKE[1],dir_120.delta_TKE[1],dir_150.delta_TKE[1],
                 dir_180.delta_TKE[1],dir_210.delta_TKE[1],dir_240.delta_TKE[1],
                 dir_270.delta_TKE[1],dir_300.delta_TKE[1],dir_330.delta_TKE[1]]
delta_TKE_WT_02=[dir_000.delta_TKE[2],dir_030.delta_TKE[2],dir_060.delta_TKE[2],
                 dir_090.delta_TKE[2],dir_120.delta_TKE[2],dir_150.delta_TKE[2],
                 dir_180.delta_TKE[2],dir_210.delta_TKE[2],dir_240.delta_TKE[2],
                 dir_270.delta_TKE[2],dir_300.delta_TKE[2],dir_330.delta_TKE[2]]
delta_TKE_WT_03=[dir_000.delta_TKE[3],dir_030.delta_TKE[3],dir_060.delta_TKE[3],
                 dir_090.delta_TKE[3],dir_120.delta_TKE[3],dir_150.delta_TKE[3],
                 dir_180.delta_TKE[3],dir_210.delta_TKE[3],dir_240.delta_TKE[3],
                 dir_270.delta_TKE[3],dir_300.delta_TKE[3],dir_330.delta_TKE[3]]
delta_TKE_WT_04=[dir_000.delta_TKE[4],dir_030.delta_TKE[4],dir_060.delta_TKE[4],
                 dir_090.delta_TKE[4],dir_120.delta_TKE[4],dir_150.delta_TKE[4],
                 dir_180.delta_TKE[4],dir_210.delta_TKE[4],dir_240.delta_TKE[4],
                 dir_270.delta_TKE[4],dir_300.delta_TKE[4],dir_330.delta_TKE[4]]

delta_TI_WT_00=[dir_000.delta_TI[0],dir_030.delta_TI[0],dir_060.delta_TI[0],
                dir_090.delta_TI[0],dir_120.delta_TI[0],dir_150.delta_TI[0],
                dir_180.delta_TI[0],dir_210.delta_TI[0],dir_240.delta_TI[0],
                dir_270.delta_TI[0],dir_300.delta_TI[0],dir_330.delta_TI[0]]
delta_TI_WT_01=[dir_000.delta_TI[1],dir_030.delta_TI[1],dir_060.delta_TI[1],
                dir_090.delta_TI[1],dir_120.delta_TI[1],dir_150.delta_TI[1],
```

```
                dir_180.delta_TI[1],dir_210.delta_TI[1],dir_240.delta_TI[1],
                dir_270.delta_TI[1],dir_300.delta_TI[1],dir_330.delta_TI[1]]
delta_TI_WT_02=[dir_000.delta_TI[2],dir_030.delta_TI[2],dir_060.delta_TI[2],
                dir_090.delta_TI[2],dir_120.delta_TI[2],dir_150.delta_TI[2],
                dir_180.delta_TI[2],dir_210.delta_TI[2],dir_240.delta_TI[2],
                dir_270.delta_TI[2],dir_300.delta_TI[2],dir_330.delta_TI[2]]
delta_TI_WT_03=[dir_000.delta_TI[3],dir_030.delta_TI[3],dir_060.delta_TI[3],
                dir_090.delta_TI[3],dir_120.delta_TI[3],dir_150.delta_TI[3],
                dir_180.delta_TI[3],dir_210.delta_TI[3],dir_240.delta_TI[3],
                dir_270.delta_TI[3],dir_300.delta_TI[3],dir_330.delta_TI[3]]
delta_TI_WT_04=[dir_000.delta_TI[4],dir_030.delta_TI[4],dir_060.delta_TI[4],
                dir_090.delta_TI[4],dir_120.delta_TI[4],dir_150.delta_TI[4],
                dir_180.delta_TI[4],dir_210.delta_TI[4],dir_240.delta_TI[4],
                dir_270.delta_TI[4],dir_300.delta_TI[4],dir_330.delta_TI[4]]

TI_WT_00=[dir_000.int_TI[0],dir_030.int_TI[0],dir_060.int_TI[0],dir_090.int_TI[0],
          dir_120.int_TI[0],dir_150.int_TI[0],dir_180.int_TI[0],dir_210.int_TI[0],
          dir_240.int_TI[0],dir_270.int_TI[0],dir_300.int_TI[0],dir_330.int_TI[0]]
TI_WT_01=[dir_000.int_TI[1],dir_030.int_TI[1],dir_060.int_TI[1],dir_090.int_TI[1],
          dir_120.int_TI[1],dir_150.int_TI[1],dir_180.int_TI[1],dir_210.int_TI[1],
          dir_240.int_TI[1],dir_270.int_TI[1],dir_300.int_TI[1],dir_330.int_TI[1]]
TI_WT_02=[dir_000.int_TI[2],dir_030.int_TI[2],dir_060.int_TI[2],dir_090.int_TI[2],
          dir_120.int_TI[2],dir_150.int_TI[2],dir_180.int_TI[2],dir_210.int_TI[2],
          dir_240.int_TI[2],dir_270.int_TI[2],dir_300.int_TI[2],dir_330.int_TI[2]]
TI_WT_03=[dir_000.int_TI[3],dir_030.int_TI[3],dir_060.int_TI[3],dir_090.int_TI[3],
          dir_120.int_TI[3],dir_150.int_TI[3],dir_180.int_TI[3],dir_210.int_TI[3],
          dir_240.int_TI[3],dir_270.int_TI[3],dir_300.int_TI[3],dir_330.int_TI[3]]
TI_WT_04=[dir_000.int_TI[4],dir_030.int_TI[4],dir_060.int_TI[4],dir_090.int_TI[4],
          dir_120.int_TI[4],dir_150.int_TI[4],dir_180.int_TI[4],dir_210.int_TI[4],
          dir_240.int_TI[4],dir_270.int_TI[4],dir_300.int_TI[4],dir_330.int_TI[4]]

# GRID OF POINTS
RC_GRID_POINTS  = [[0] * 12 for _ in range(dim)]
RCx_GRID_POINTS = [[0] * 12 for _ in range(dim)]
RCy_GRID_POINTS = [[0] * 12 for _ in range(dim)]
TI_GRID_POINTS  = [[0] * 12 for _ in range(dim)]

for i in range (0,dim):
    RC_GRID_POINTS[i] = [dir_000.Grid_RC[i],
                         dir_030.Grid_RC[i],
                         dir_060.Grid_RC[i],
                         dir_090.Grid_RC[i],
                         dir_120.Grid_RC[i],
                         dir_150.Grid_RC[i],
                         dir_180.Grid_RC[i],
                         dir_210.Grid_RC[i],
                         dir_240.Grid_RC[i],
                         dir_270.Grid_RC[i],
                         dir_300.Grid_RC[i],
                         dir_330.Grid_RC[i]]

    RCx_GRID_POINTS[i] =[dir_000.Grid_RCx[i],
                         dir_030.Grid_RCx[i],
                         dir_060.Grid_RCx[i],
                         dir_090.Grid_RCx[i],
                         dir_120.Grid_RCx[i],
                         dir_150.Grid_RCx[i],
                         dir_180.Grid_RCx[i],
                         dir_210.Grid_RCx[i],
                         dir_240.Grid_RCx[i],
                         dir_270.Grid_RCx[i],
                         dir_300.Grid_RCx[i],
                         dir_330.Grid_RCx[i]]

    RCy_GRID_POINTS[i] =[dir_000.Grid_RCy[i],
                         dir_030.Grid_RCy[i],
                         dir_060.Grid_RCy[i],
                         dir_090.Grid_RCy[i],
                         dir_120.Grid_RCy[i],
                         dir_150.Grid_RCy[i],
                         dir_180.Grid_RCy[i],
                         dir_210.Grid_RCy[i],
                         dir_240.Grid_RCy[i],
                         dir_270.Grid_RCy[i],
                         dir_300.Grid_RCy[i],
                         dir_330.Grid_RCy[i]]

    TI_GRID_POINTS[i]  =[dir_000.Grid_interpolated_TI[i],
                         dir_030.Grid_interpolated_TI[i],
                         dir_060.Grid_interpolated_TI[i],
                         dir_090.Grid_interpolated_TI[i],
                         dir_120.Grid_interpolated_TI[i],
                         dir_150.Grid_interpolated_TI[i],
                         dir_180.Grid_interpolated_TI[i],
                         dir_210.Grid_interpolated_TI[i],
```

```
                          dir_240.Grid_interpolated_TI[i],
                          dir_270.Grid_interpolated_TI[i],
                          dir_300.Grid_interpolated_TI[i],
                          dir_330.Grid_interpolated_TI[i]]

print(RC_GRID_POINTS[0])

flow_inclination_group_WT_01_AVG = np.mean(flow_inclination_group_WT_01)
flow_inclination_group_WT_02_AVG = np.mean(flow_inclination_group_WT_02)
flow_inclination_group_WT_03_AVG = np.mean(flow_inclination_group_WT_03)
flow_inclination_group_WT_04_AVG = np.mean(flow_inclination_group_WT_04)

wind_shear_group_WT_01_AVG = np.mean(wind_shear_group_WT_01)
wind_shear_group_WT_02_AVG = np.mean(wind_shear_group_WT_02)
wind_shear_group_WT_03_AVG = np.mean(wind_shear_group_WT_03)
wind_shear_group_WT_04_AVG = np.mean(wind_shear_group_WT_04)

# »»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»
# Calculates directions at mast coordinates

ID=[1,2,3,4,5,6,7,8,9,10,11,12]
Orientacao=['N','NNE','ENE','E','ESE','SSE','S','SSW','WSW','W','WNW','NNW']
Graus=['0','30','60','90','120','150','180','210','240','270','300','330']

Ux_mastro=[dir_000.int_Ux[0],dir_030.int_Ux[0],dir_060.int_Ux[0],dir_090.int_Ux[0],
           dir_120.int_Ux[0],dir_150.int_Ux[0],dir_180.int_Ux[0],dir_210.int_Ux[0],
           dir_240.int_Ux[0],dir_270.int_Ux[0],dir_300.int_Ux[0],dir_330.int_Ux[0]]
Uy_mastro=[dir_000.int_Uy[0],dir_030.int_Uy[0],dir_060.int_Uy[0],dir_090.int_Uy[0],
           dir_120.int_Uy[0],dir_150.int_Uy[0],dir_180.int_Uy[0],dir_210.int_Uy[0],
           dir_240.int_Uy[0],dir_270.int_Uy[0],dir_300.int_Uy[0],dir_330.int_Uy[0]]
Dir_mastro=[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0]
Desvio_Dir_mastro=[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0]
Desvio_Dir_mastro_percent = [0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0]

# Directions
for i in range(0,12):
    # 1ž quadrant
    if Ux_mastro[i]<=0.0 and Uy_mastro[i]<0.0:
        Dir_mastro[i]=np.arctan2(-Ux_mastro[i],-Uy_mastro[i])*180.0/np.pi
    # 2ž quadrant
    if Ux_mastro[i]>0.0 and Uy_mastro[i]<=0.0:
        Dir_mastro[i]=180.0-np.arctan2(-Ux_mastro[i],Uy_mastro[i])*180.0/np.pi
    # 3ž quadrant
    if Ux_mastro[i]>=0.0 and Uy_mastro[i]>0.0:
        Dir_mastro[i]=np.arctan2(-Ux_mastro[i],-Uy_mastro[i])*180.0/np.pi+360.0
    # 4ž quadrant
    if Ux_mastro[i]<0.0 and Uy_mastro[i]>=0.0:
        Dir_mastro[i]=np.arctan2(-Ux_mastro[i],-Uy_mastro[i])*180.0/np.pi

# Deviations
for i in range(0,12):
    if i ==0:
        if Dir_mastro[i]<300.0:
            Desvio_Dir_mastro[i]=Dir_mastro[i]
        else:
            Desvio_Dir_mastro[i]=360.0-Dir_mastro[i]
    else:
        Desvio_Dir_mastro[i]= float(Graus[i])-Dir_mastro[i]

# »»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»
# Interpolates speed-ups for each wind rose direction

RC_WIND_ROSE_WT_01=[]
RC_WIND_ROSE_WT_02=[]
RC_WIND_ROSE_WT_03=[]
RC_WIND_ROSE_WT_04=[]

RC_WIND_ROSE_WT_01_plot = []
RC_WIND_ROSE_WT_02_plot=[]
RC_WIND_ROSE_WT_03_plot=[]
RC_WIND_ROSE_WT_04_plot = []

RCx_WIND_ROSE_WT_01 = []
RCx_WIND_ROSE_WT_02=[]
RCx_WIND_ROSE_WT_03=[]
RCx_WIND_ROSE_WT_04 = []

RCy_WIND_ROSE_WT_01 = []
RCy_WIND_ROSE_WT_02=[]
RCy_WIND_ROSE_WT_03=[]
RCy_WIND_ROSE_WT_04=[]

# Grid of points
RC_WIND_ROSE_GRID  = [[0] * 360 for _ in range(dim)]
RCx_WIND_ROSE_GRID = [[0] * 360 for _ in range(dim)]
```

```
RCy_WIND_ROSE_GRID = [[0] * 360 for _ in range(dim)]
TI_WIND_ROSE_GRID  = [[0] * 360 for _ in range(dim)]

eixo_dir = []

deltaTI_WIND_ROSE_WT_01 = []
deltaTI_WIND_ROSE_WT_02 = []
deltaTI_WIND_ROSE_WT_03 = []
deltaTI_WIND_ROSE_WT_04 = []

TI_WIND_ROSE_WT_00 = []
TI_WIND_ROSE_WT_01 = []
TI_WIND_ROSE_WT_02 = []
TI_WIND_ROSE_WT_03 = []
TI_WIND_ROSE_WT_04 = []

# Fills arrays with inicial values
for i in range(0,360):
    RC_WIND_ROSE_WT_01.append(0.0)
    RC_WIND_ROSE_WT_02.append(0.0)
    RC_WIND_ROSE_WT_03.append(0.0)
    RC_WIND_ROSE_WT_04.append(0.0)
    RC_WIND_ROSE_WT_01_plot.append(0.0)
    RC_WIND_ROSE_WT_02_plot.append(0.0)
    RC_WIND_ROSE_WT_03_plot.append(0.0)
    RC_WIND_ROSE_WT_04_plot.append(0.0)
    RCx_WIND_ROSE_WT_01.append(0.0)
    RCx_WIND_ROSE_WT_02.append(0.0)
    RCx_WIND_ROSE_WT_03.append(0.0)
    RCx_WIND_ROSE_WT_04.append(0.0)
    RCy_WIND_ROSE_WT_01.append(0.0)
    RCy_WIND_ROSE_WT_02.append(0.0)
    RCy_WIND_ROSE_WT_03.append(0.0)
    RCy_WIND_ROSE_WT_04.append(0.0)
    deltaTI_WIND_ROSE_WT_01.append(0.0)
    deltaTI_WIND_ROSE_WT_02.append(0.0)
    deltaTI_WIND_ROSE_WT_03.append(0.0)
    deltaTI_WIND_ROSE_WT_04.append(0.0)
    TI_WIND_ROSE_WT_00.append(0.0)
    TI_WIND_ROSE_WT_01.append(0.0)
    TI_WIND_ROSE_WT_02.append(0.0)
    TI_WIND_ROSE_WT_03.append(0.0)
    TI_WIND_ROSE_WT_04.append(0.0)
    eixo_dir.append(i)

# Interpolates for each wind turbine
Dir_mastro=[0,30,60,90,120,150,180,210,240,270,300,330]

RC_WIND_ROSE_WT_01 = interpolate_for_each_degree(RC_WIND_ROSE_WT_01,RC_WT_01,Dir_mastro)
RC_WIND_ROSE_WT_02 = interpolate_for_each_degree(RC_WIND_ROSE_WT_02,RC_WT_02,Dir_mastro)
RC_WIND_ROSE_WT_03 = interpolate_for_each_degree(RC_WIND_ROSE_WT_03,RC_WT_03,Dir_mastro)
RC_WIND_ROSE_WT_04 = interpolate_for_each_degree(RC_WIND_ROSE_WT_04,RC_WT_04,Dir_mastro)

RCx_WIND_ROSE_WT_01 = interpolate_for_each_degree(RCx_WIND_ROSE_WT_01,RCx_WT_01,Dir_mastro)
RCx_WIND_ROSE_WT_02 = interpolate_for_each_degree(RCx_WIND_ROSE_WT_02,RCx_WT_02,Dir_mastro)
RCx_WIND_ROSE_WT_03 = interpolate_for_each_degree(RCx_WIND_ROSE_WT_03,RCx_WT_03,Dir_mastro)
RCx_WIND_ROSE_WT_04 = interpolate_for_each_degree(RCx_WIND_ROSE_WT_04,RCx_WT_04,Dir_mastro)

RCy_WIND_ROSE_WT_01 = interpolate_for_each_degree(RCy_WIND_ROSE_WT_01,RCy_WT_01,Dir_mastro)
RCy_WIND_ROSE_WT_02 = interpolate_for_each_degree(RCy_WIND_ROSE_WT_02,RCy_WT_02,Dir_mastro)
RCy_WIND_ROSE_WT_03 = interpolate_for_each_degree(RCy_WIND_ROSE_WT_03,RCy_WT_03,Dir_mastro)
RCy_WIND_ROSE_WT_04 = interpolate_for_each_degree(RCy_WIND_ROSE_WT_04,RCy_WT_04,Dir_mastro)

# Grid of points
for i in range(0,dim):
    RC_WIND_ROSE_GRID[i] =
    interpolate_for_each_degree(RC_WIND_ROSE_GRID[i],RC_GRID_POINTS[i],Dir_mastro)
    RCx_WIND_ROSE_GRID[i] =
    interpolate_for_each_degree(RCx_WIND_ROSE_GRID[i],RCx_GRID_POINTS[i],Dir_mastro)
    RCy_WIND_ROSE_GRID[i] =
    interpolate_for_each_degree(RCy_WIND_ROSE_GRID[i],RCy_GRID_POINTS[i],Dir_mastro)
    TI_WIND_ROSE_GRID[i] =
    interpolate_for_each_degree(TI_WIND_ROSE_GRID[i],TI_GRID_POINTS[i],Dir_mastro)

RC_WIND_ROSE_WT_01_plot =
interpolate_for_each_degree(RC_WIND_ROSE_WT_01_plot,RC_WT_01_plot,Dir_mastro)
RC_WIND_ROSE_WT_02_plot =
interpolate_for_each_degree(RC_WIND_ROSE_WT_02_plot,RC_WT_02_plot,Dir_mastro)
RC_WIND_ROSE_WT_03_plot =
interpolate_for_each_degree(RC_WIND_ROSE_WT_03_plot,RC_WT_03_plot,Dir_mastro)
RC_WIND_ROSE_WT_04_plot =
interpolate_for_each_degree(RC_WIND_ROSE_WT_04_plot,RC_WT_04_plot,Dir_mastro)

deltaTI_WIND_ROSE_WT_01 =
interpolate_for_each_degree(deltaTI_WIND_ROSE_WT_01,delta_TI_WT_01,Dir_mastro)
```

```
deltaTI_WIND_ROSE_WT_02 =
interpolate_for_each_degree(deltaTI_WIND_ROSE_WT_02,delta_TI_WT_02,Dir_mastro)
deltaTI_WIND_ROSE_WT_03 =
interpolate_for_each_degree(deltaTI_WIND_ROSE_WT_03,delta_TI_WT_03,Dir_mastro)
deltaTI_WIND_ROSE_WT_04 =
interpolate_for_each_degree(deltaTI_WIND_ROSE_WT_04,delta_TI_WT_04,Dir_mastro)

TI_WIND_ROSE_WT_00 = interpolate_for_each_degree(TI_WIND_ROSE_WT_00,TI_WT_00,Dir_mastro)
TI_WIND_ROSE_WT_01 = interpolate_for_each_degree(TI_WIND_ROSE_WT_01,TI_WT_01,Dir_mastro)
TI_WIND_ROSE_WT_02 = interpolate_for_each_degree(TI_WIND_ROSE_WT_02,TI_WT_02,Dir_mastro)
TI_WIND_ROSE_WT_03 = interpolate_for_each_degree(TI_WIND_ROSE_WT_03,TI_WT_03,Dir_mastro)
TI_WIND_ROSE_WT_04 = interpolate_for_each_degree(TI_WIND_ROSE_WT_04,TI_WT_04,Dir_mastro)

# »»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»
# Mathematical model validation

DataLog_SIM_48_U=[]
DataLog_SIM_82_U=[]
DataLog_SIM_120_U=[]
DataLog_SIM_48_TI=[]
DataLog_SIM_82_TI=[]
DataLog_SIM_120_TI=[]

# Reads the wind data files"DataLog_48m_SIM.txt" e saves all data
cont_a=0
file= open(path+"DataLog_48m_SIM.txt", "r")

for line in file.readlines():
    leitura=line.split(",")
    DataLog_SIM_48_U.append(float(leitura[2]))
    if float(leitura[2])>5.0:
        DataLog_SIM_48_TI.append(float(leitura[5])/float(leitura[2])*100.0)
    cont_a += 1
file.close()

# Reads the wind data files"DataLog_82m_SIM.txt" e saves all data
cont_b = 0
file= open(path+"DataLog_82m_SIM.txt", "r")

for line in file.readlines():
    leitura=line.split(",")
    DataLog_SIM_82_U.append(float(leitura[2]))
    if float(leitura[2])>5.0:
        DataLog_SIM_82_TI.append(float(leitura[5])/float(leitura[2])*100.0)
    cont_b += 1
file.close()

# Reads the wind data files"DataLog_120m_SIM.txt" e saves all data
cont_c = 0
file= open(path+"DataLog_120m_SIM.txt", "r")

for line in file.readlines():
    leitura=line.split(",")
    DataLog_SIM_120_U.append(float(leitura[2]))
    if float(leitura[2])>5.0:
        DataLog_SIM_120_TI.append(float(leitura[5])/float(leitura[2])*100.0)
    cont_c+=1
file.close()

# Calculates the average velocity for each time series
DataLog_SIM_48_U_avg  = np.mean(DataLog_SIM_48_U)
DataLog_SIM_82_U_avg  = np.mean(DataLog_SIM_82_U)
DataLog_SIM_120_U_avg = np.mean(DataLog_SIM_120_U)
DataLog_SIM_48_TI_avg = np.mean(DataLog_SIM_48_TI)
DataLog_SIM_82_TI_avg = np.mean(DataLog_SIM_82_TI)
DataLog_SIM_120_TI_avg= np.mean(DataLog_SIM_120_TI)

# »»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»
# Transport the time series

from datetime import datetime

DataLog_Dir = []
DataLog_U = []
DataLog_Ux = []
DataLog_Uy = []
DataLog_DesvPad = []
DataLog_DateTime = []
DataLog_TI = []

# Opens wind data files
j = 0
cont_TI = 0

if Altura_medicao_mastro == 48:
```

```
    file = open(path+"DataLog_48m_SIM.txt", "r")

if Altura_medicao_mastro == 82:
    file = open(path+"DataLog_82m_SIM.txt", "r")

if Altura_medicao_mastro == 120:
    file = open(path+"DataLog_120m_SIM.txt", "r")

for line in file.readlines():
    leitura = line.split(",")

    if float(leitura[2]) != 99.99:
        DataLog_Dir.append(int(leitura[1]))
        DataLog_U.append(float(leitura[2]))
        DataLog_Ux.append(0.0)
        DataLog_Uy.append(0.0)
        DataLog_DesvPad.append(float(leitura[5]))
        DataLog_DateTime.append(leitura[6])
        j = j + 1
        if float(leitura[2]) > 5.0:
            DataLog_TI.append(float(leitura[5])/float(leitura[2])*100.0)
        else:
            DataLog_TI.append(0.0)
file.close()

# Calculates "Ux" and "Uy" components
for i in range(0,j):
    theta = 0
    # 3º quadrant
    if DataLog_Dir[i] >= 0.0 and DataLog_Dir[i] < 90.0:
        DataLog_Ux[i] = -DataLog_U[i] * np.sin(DataLog_Dir[i]*np.pi/180.0)
        DataLog_Uy[i] = -DataLog_U[i] * np.cos(DataLog_Dir[i]*np.pi/180.0)
    # 2º quadrant
    if DataLog_Dir[i] >= 90.0 and DataLog_Dir[i] < 180.0:
        theta = 180.0-DataLog_Dir[i]
        DataLog_Ux[i] = -DataLog_U[i] * np.sin(theta*np.pi/180.0)
        DataLog_Uy[i] = DataLog_U[i] * np.cos(theta*np.pi/180.0)
    # 1º quadrant
    if DataLog_Dir[i] >= 180.0 and DataLog_Dir[i] < 270.0:
        theta_ = DataLog_Dir[i]-180.0
        DataLog_Ux[i] = DataLog_U[i] * np.sin(theta*np.pi/180.0)
        DataLog_Uy[i] = DataLog_U[i] * np.cos(theta*np.pi/180.0)
    # 4º quadrant
    if DataLog_Dir[i] >= 270.0 and DataLog_Dir[i] < 360.0:
        theta_ = 360.0-DataLog_Dir[i]
        DataLog_Ux[i] = DataLog_U[i] * np.sin(theta*np.pi/180.0)
        DataLog_Uy[i] = -DataLog_U[i] * np.cos(theta*np.pi/180.0)

# Calculates the time interval corresponding to the measurement campaign
DT_array_i = split(DataLog_DateTime[0])
DT_array_f = split(DataLog_DateTime[j-1])

DT_i = DT_to_string(DT_array_i)
DT_f = DT_to_string(DT_array_f)

delta_days = days_between(DT_i,DT_f)

# Write speed-up ratio in the in a virtual time series
time_series_speed_up_01 = []
time_series_speed_up_02 = []
time_series_speed_up_03 = []
time_series_speed_up_04 = []

time_series_speed_up_x_01 = []
time_series_speed_up_x_02 = []
time_series_speed_up_x_03 = []
time_series_speed_up_x_04 = []

time_series_speed_up_y_01 = []
time_series_speed_up_y_02 = []
time_series_speed_up_y_03 = []
time_series_speed_up_y_04 = []

time_series_deltaTI_01 = []
time_series_deltaTI_02 = []
time_series_deltaTI_03 = []
time_series_deltaTI_04 = []

time = []

time_series_U_01 = []
time_series_U_02 = []
time_series_U_03 = []
time_series_U_04 = []

```

```
1767 time_series_Ux_01 = []
1768 time_series_Ux_02 = []
1769 time_series_Ux_03 = []
1770 time_series_Ux_04 = []
1771
1772 time_series_Uy_01 = []
1773 time_series_Uy_02 = []
1774 time_series_Uy_03 = []
1775 time_series_Uy_04 = []
1776
1777 # GRID OF POINTS
1778 TIME_SERIES_GRID_RC  = [[] for _ in range(dim)]
1779 TIME_SERIES_GRID_RCx = [[] for _ in range(dim)]
1780 TIME_SERIES_GRID_RCy = [[] for _ in range(dim)]
1781
1782 TIME_SERIES_GRID_U  = [[] for _ in range(dim)]
1783 TIME_SERIES_GRID_Ux = [[] for _ in range(dim)]
1784 TIME_SERIES_GRID_Uy = [[] for _ in range(dim)]
1785
1786 TIME_SERIES_GRID_P = [[] for _ in range(dim)]
1787 TIME_SERIES_GRID_E = [[] for _ in range(dim)]
1788
1789 time_series_dir_01 = []
1790 time_series_dir_02 = []
1791 time_series_dir_03 = []
1792 time_series_dir_04 = []
1793
1794 time_series_P_01 = []
1795 time_series_P_02 = []
1796 time_series_P_03 = []
1797 time_series_P_04 = []
1798
1799 time_series_E_01 = []
1800 time_series_E_02 = []
1801 time_series_E_03 = []
1802 time_series_E_04 = []
1803
1804 time_series_TI_mastro_calc = []
1805 time_series_TI_01_calc = []
1806 time_series_TI_02_calc = []
1807 time_series_TI_03_calc = []
1808 time_series_TI_04_calc = []
1809
1810 time_series_TI_01 = []
1811 time_series_TI_02 = []
1812 time_series_TI_03 = []
1813 time_series_TI_04 = []
1814
1815
1816 # Transport the time series to all wind turbines
1817 cont = 0
1818 for i in DataLog_Dir:
1819     if i > 359.5:
1820         index = 0
1821     else:
1822         index = int(i)
1823
1824     # Speed-ups
1825     time_series_speed_up_01.append(RC_WIND_ROSE_WT_01[index])
1826     time_series_speed_up_02.append(RC_WIND_ROSE_WT_02[index])
1827     time_series_speed_up_03.append(RC_WIND_ROSE_WT_03[index])
1828     time_series_speed_up_04.append(RC_WIND_ROSE_WT_04[index])
1829
1830     time_series_speed_up_x_01.append(RCx_WIND_ROSE_WT_01[index])
1831     time_series_speed_up_x_02.append(RCx_WIND_ROSE_WT_02[index])
1832     time_series_speed_up_x_03.append(RCx_WIND_ROSE_WT_03[index])
1833     time_series_speed_up_x_04.append(RCx_WIND_ROSE_WT_04[index])
1834
1835     time_series_speed_up_y_01.append(RCy_WIND_ROSE_WT_01[index])
1836     time_series_speed_up_y_02.append(RCy_WIND_ROSE_WT_02[index])
1837     time_series_speed_up_y_03.append(RCy_WIND_ROSE_WT_03[index])
1838     time_series_speed_up_y_04.append(RCy_WIND_ROSE_WT_04[index])
1839
1840     # Grid of points
1841     for record in range(0,dim):
1842         TIME_SERIES_GRID_RC[record].append(RC_WIND_ROSE_GRID[record][index])
1843         TIME_SERIES_GRID_RCx[record].append(RCx_WIND_ROSE_GRID[record][index])
1844         TIME_SERIES_GRID_RCy[record].append(RCy_WIND_ROSE_GRID[record][index])
1845
1846         TIME_SERIES_GRID_U[record].append(0.0)
1847         TIME_SERIES_GRID_Ux[record].append(0.0)
1848         TIME_SERIES_GRID_Uy[record].append(0.0)
1849
1850         TIME_SERIES_GRID_P[record].append(0.0)
1851         TIME_SERIES_GRID_E[record].append(0.0)
```

```
1852          #TIME_SERIES_GRID_I[record].append(0.0)
1853
1854      # Delta TI
1855      time_series_deltaTI_01.append(DataLog_TI[cont] - TI_WIND_ROSE_WT_00[index])
1856      time_series_deltaTI_02.append(DataLog_TI[cont] - TI_WIND_ROSE_WT_00[index])
1857      time_series_deltaTI_03.append(DataLog_TI[cont] - TI_WIND_ROSE_WT_00[index])
1858      time_series_deltaTI_04.append(DataLog_TI[cont] - TI_WIND_ROSE_WT_00[index])
1859
1860      time_series_TI_mastro_calc.append(TI_WIND_ROSE_WT_00[index])
1861      time_series_TI_01_calc.append(TI_WIND_ROSE_WT_01[index])
1862      time_series_TI_02_calc.append(TI_WIND_ROSE_WT_02[index])
1863      time_series_TI_03_calc.append(TI_WIND_ROSE_WT_03[index])
1864      time_series_TI_04_calc.append(TI_WIND_ROSE_WT_04[index])
1865
1866      time_series_TI_01.append(0.0)
1867      time_series_TI_02.append(0.0)
1868      time_series_TI_03.append(0.0)
1869      time_series_TI_04.append(0.0)
1870
1871      # Wind speed
1872      time_series_U_01.append(0.0)
1873      time_series_U_02.append(0.0)
1874      time_series_U_03.append(0.0)
1875      time_series_U_04.append(0.0)
1876
1877      time_series_Ux_01.append(0.0)
1878      time_series_Ux_02.append(0.0)
1879      time_series_Ux_03.append(0.0)
1880      time_series_Ux_04.append(0.0)
1881
1882      time_series_Uy_01.append(0.0)
1883      time_series_Uy_02.append(0.0)
1884      time_series_Uy_03.append(0.0)
1885      time_series_Uy_04.append(0.0)
1886
1887      # Flow direction
1888      time_series_dir_01.append(0.0)
1889      time_series_dir_02.append(0.0)
1890      time_series_dir_03.append(0.0)
1891      time_series_dir_04.append(0.0)
1892
1893      # Power + energy
1894      time_series_P_01.append(0.0)
1895      time_series_P_02.append(0.0)
1896      time_series_P_03.append(0.0)
1897      time_series_P_04.append(0.0)
1898
1899      time_series_E_01.append(0.0)
1900      time_series_E_02.append(0.0)
1901      time_series_E_03.append(0.0)
1902      time_series_E_04.append(0.0)
1903
1904      time.append(cont)
1905      cont = cont + 1
1906
1907  # Writes the final transported time series
1908  for i in range(0,cont-1):
1909
1910      # Wind speed
1911      time_series_U_01[i] = time_series_speed_up_01[i]*DataLog_U[i]
1912      time_series_U_02[i] = time_series_speed_up_02[i]*DataLog_U[i]
1913      time_series_U_03[i] = time_series_speed_up_03[i]*DataLog_U[i]
1914      time_series_U_04[i] = time_series_speed_up_04[i]*DataLog_U[i]
1915
1916      time_series_Ux_01[i] = time_series_speed_up_x_01[i]*DataLog_U[i]
1917      time_series_Ux_02[i] = time_series_speed_up_x_02[i]*DataLog_U[i]
1918      time_series_Ux_03[i] = time_series_speed_up_x_03[i]*DataLog_U[i]
1919      time_series_Ux_04[i] = time_series_speed_up_x_04[i]*DataLog_U[i]
1920
1921      time_series_Uy_01[i] = time_series_speed_up_y_01[i]*DataLog_U[i]
1922      time_series_Uy_02[i] = time_series_speed_up_y_02[i]*DataLog_U[i]
1923      time_series_Uy_03[i] = time_series_speed_up_y_03[i]*DataLog_U[i]
1924      time_series_Uy_04[i] = time_series_speed_up_y_04[i]*DataLog_U[i]
1925
1926      # Turbulent intensity
1927      if DataLog_U[i] > 5.0:
1928          time_series_TI_01[i] =
1929          time_series_TI_01_calc[i] + time_series_deltaTI_01[i]/(time_series_speed_up_01[i])
1930          time_series_TI_02[i] =
1931          time_series_TI_02_calc[i] + time_series_deltaTI_02[i]/(time_series_speed_up_02[i])
1932          time_series_TI_03[i] =
1933          time_series_TI_03_calc[i] + time_series_deltaTI_03[i]/(time_series_speed_up_03[i])
1934          time_series_TI_04[i] =
1935          time_series_TI_04_calc[i] + time_series_deltaTI_04[i]/(time_series_speed_up_04[i])
1936      else:
```

```
1937            time_series_TI_01[i] = 0.0
1938            time_series_TI_02[i] = 0.0
1939            time_series_TI_03[i] = 0.0
1940            time_series_TI_04[i] = 0.0
1941
1942        # Wind flow direction
1943        time_series_dir_01[i] = get_flow_direction(time_series_Ux_01[i],time_series_Uy_01[i])
1944        time_series_dir_02[i] = get_flow_direction(time_series_Ux_02[i],time_series_Uy_02[i])
1945        time_series_dir_03[i] = get_flow_direction(time_series_Ux_03[i],time_series_Uy_03[i])
1946        time_series_dir_04[i] = get_flow_direction(time_series_Ux_04[i],time_series_Uy_04[i])
1947
1948    # For the grid of points
1949    for record in range(0,dim):
1950        for i in range(0,cont-1):
1951            TIME_SERIES_GRID_U[record][i]  = TIME_SERIES_GRID_RC[record][i]*DataLog_U[i]
1952            TIME_SERIES_GRID_Ux[record][i] = TIME_SERIES_GRID_RCx[record][i]*DataLog_U[i]
1953            TIME_SERIES_GRID_Uy[record][i] = TIME_SERIES_GRID_RCy[record][i]*DataLog_U[i]
1954
1955    time_series_TI_01 = np.array(time_series_TI_01)
1956    time_series_TI_02 = np.array(time_series_TI_02)
1957    time_series_TI_03 = np.array(time_series_TI_03)
1958    time_series_TI_04 = np.array(time_series_TI_04)
1959
1960
1961    # »»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»
1962    # Writes the files to the used by Weibull algorithm
1963
1964    time_series_dir_01_aux=[]
1965    time_series_dir_02_aux=[]
1966    time_series_dir_03_aux=[]
1967    time_series_dir_04_aux=[]
1968    time_series_U_01_aux=[]
1969    time_series_U_02_aux=[]
1970    time_series_U_03_aux=[]
1971    time_series_U_04_aux=[]
1972
1973    for i in range(0,j):
1974        time_series_dir_01_aux.append('x')
1975        time_series_dir_02_aux.append('x')
1976        time_series_dir_03_aux.append('x')
1977        time_series_dir_04_aux.append('x')
1978        time_series_U_01_aux.append('x')
1979        time_series_U_02_aux.append('x')
1980        time_series_U_03_aux.append('x')
1981        time_series_U_04_aux.append('x')
1982
1983    for i in range(0,j):
1984        time_series_dir_01_aux[i]= str(time_series_dir_01[i])
1985        time_series_dir_02_aux[i]= str(time_series_dir_02[i])
1986        time_series_dir_03_aux[i]= str(time_series_dir_03[i])
1987        time_series_dir_04_aux[i]= str(time_series_dir_04[i])
1988        time_series_U_01_aux[i]= str(time_series_U_01[i])
1989        time_series_U_02_aux[i]= str(time_series_U_02[i])
1990        time_series_U_03_aux[i]= str(time_series_U_03[i])
1991        time_series_U_04_aux[i]= str(time_series_U_04[i])
1992
1993    with open(path+'DataLog_z0_roug_120_WT_01.txt', 'w') as f:
1994        for i in range(0,j):
1995            f.write('readme,'+time_series_dir_01_aux[i]+','+time_series_U_01_aux[i])
1996            f.write('\n')
1997    f.close()
1998
1999    with open(path+'DataLog_z0_roug_120_WT_02.txt', 'w') as f:
2000        for i in range(0,j):
2001            f.write('readme,'+time_series_dir_02_aux[i]+','+time_series_U_02_aux[i])
2002            f.write('\n')
2003    f.close()
2004
2005    with open(path+'DataLog_z0_roug_120_WT_03.txt', 'w') as f:
2006        for i in range(0,j):
2007            f.write('readme,'+time_series_dir_03_aux[i]+','+time_series_U_03_aux[i])
2008            f.write('\n')
2009    f.close()
2010
2011    with open(path+'DataLog_z0_roug_120_WT_04.txt', 'w') as f:
2012        for i in range(0,j):
2013            f.write('readme,'+time_series_dir_04_aux[i]+','+time_series_U_04_aux[i])
2014            f.write('\n')
2015    f.close()
2016
2017    # »»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»
2018    # Wind farm estimate energy production
2019
2020    WindSpeed = []
2021    PowerOutput = []
```

```
Energia_total_parque = [0.0,0.0,0.0,0.0]
Energia_total_parque_ano = [0.0,0.0,0.0,0.0]

# Reads the file corresponding to turbine power curve "TURBINEPOWERCURVE.TXT"
cont = 0
file = open(path+"TurbinePowerCurve.txt", "r")

for line in file.readlines():
    leitura = line.split(",")

    WindSpeed.append(float(leitura[0]))
    PowerOutput.append(float(leitura[1])/1000.0)
    cont = cont + 1
file.close()

# Calculates the power for each record
for i in range(0,j-1):
    time_series_P_01[i] = interpolate_power(time_series_U_01[i],WindSpeed,PowerOutput) # kW
    time_series_P_02[i] = interpolate_power(time_series_U_02[i],WindSpeed,PowerOutput) # kW
    time_series_P_03[i] = interpolate_power(time_series_U_03[i],WindSpeed,PowerOutput) # kW
    time_series_P_04[i] = interpolate_power(time_series_U_04[i],WindSpeed,PowerOutput) # kW
    time_series_E_01[i] = time_series_P_01[i]*(10.0/60.0) # kWh
    time_series_E_02[i] = time_series_P_02[i]*(10.0/60.0) # kWh
    time_series_E_03[i] = time_series_P_03[i]*(10.0/60.0) # kWh
    time_series_E_04[i] = time_series_P_04[i]*(10.0/60.0) # kWh

    # Grid of points
    for record in range(0,dim):
        TIME_SERIES_GRID_P[record][i] =
        interpolate_power(TIME_SERIES_GRID_U[record][i],WindSpeed,PowerOutput) #kW
        TIME_SERIES_GRID_E[record][i] =
        TIME_SERIES_GRID_P[record][i]*(10.0/60.0) # kWh

# Energy
Energia_total_parque[0] = sum(time_series_E_01)/1000.0 # MWh
Energia_total_parque[1] = sum(time_series_E_02)/1000.0 # MWh
Energia_total_parque[2] = sum(time_series_E_03)/1000.0 # MWh
Energia_total_parque[3] = sum(time_series_E_04)/1000.0 # MWh

for i in range(0,4):
    Energia_total_parque_ano[i] = 365.0 * Energia_total_parque[i] / delta_days

Energia_total = sum(Energia_total_parque_ano)

# Grid of points
ENERGIA_TOTAL_GRID = np.zeros(dim)
ENERGIA_TOTAL_PARQUE_GRID = np.zeros(dim)

for record in range(0,dim):
    ENERGIA_TOTAL_GRID[record] = sum(TIME_SERIES_GRID_E[record])/1000.0 # MWh
    ENERGIA_TOTAL_PARQUE_GRID[record] = 365.0 * ENERGIA_TOTAL_GRID[record] / delta_days

# Capacity factor
producao_max = (4.6*4)*365*24
fator_cap = (Energia_total/producao_max)*100.0
print("Fator de capacidade =", round(fator_cap,2), "%")

# Full load hours
horas_plena_carga = Energia_total/(4.6*4)
print("HORAS EM PLENA CARGA =", round(horas_plena_carga,2), "horas")

# >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
# Calculates the value for the grid of points contour plot

for i in range(0,dim):
    Grid_U_avg_Synthesis[i] = round(np.mean(TIME_SERIES_GRID_U[i]),3)
    Grid_AEP_Synthesis[i] = round(ENERGIA_TOTAL_PARQUE_GRID[i],3)
```

B.3 Generate_canopy_vtk.py

This algorithm is responsible to generate a topographic map that includes the canopy height. In terms of procedure, it consists in a mathematical manipulation that is performed using both the topography map (`topo_LOCAL.vtk`) and roughness map (`rough_LOCAL.vtk`).

Essentially, the concept involves reading the coordinates of both maps, and at points where the roughness map presents aerodynamic roughness length values greater than or equal to 0.7 m, the height of the trees is added to the topography map at those same coordinates. The result is a manipulated topography map that has an offset in z direction corresponding to the forest (see Figure 5.2). The source code of `Generate_canopy_vtk.py` file is presented below.

```
# »»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»
# Variables

x_topo_vals = []
y_topo_vals = []
z_topo_vals = []

x_roug_vals = []
y_roug_vals = []
z_roug_vals = []

z_canopy_vals = []

# »»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»
# Looks for canopy height in "_preprocDict" configuration file

print('Reading _preprocDict ...')
_preprocDict_data = [lt.strip().split() for lt in open("./../_preprocDict").readlines()]
h_canopy = float(_preprocDict_data[33][2])
print("hcano = ",h_canopy)

# »»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»
# Read maps "topo_LOCAL.vtk" and "rough_LOCAL.vtk"

print('Reading files topo_LOCAL.vtk and rough_LOCAL.vtk ...')
topo_datContent = [lt.strip().split() for lt in open("./topo_LOCAL.vtk").readlines()]
roug_datContent = [lr.strip().split() for lr in open("./rough_LOCAL.vtk").readlines()]

Nx_text = topo_datContent[4][1]
Ny_text = topo_datContent[4][2]
Nz_text = float(1)
points_text = topo_datContent[5][1]

Nx = int(Nx_text)
Ny = int(Ny_text)
Nz = int(Nz_text)
points = Nx * Ny

for i in range(0,len(topo_datContent)):
    lines_topo = topo_datContent[i]
    if i >= 6 and i < (6+points):
        x_topo_vals.append(float(lines_topo[0]))
        y_topo_vals.append(float(lines_topo[1]))
        z_topo_vals.append(float(lines_topo[2]))
        z_canopy_vals.append(0.0)

for i in range(0,len(roug_datContent)):
    lines_roug = roug_datContent[i]
    if i >= 6 and i < (6+points):
        x_roug_vals.append(float(lines_roug[0]))
        y_roug_vals.append(float(lines_roug[1]))
        z_roug_vals.append(float(lines_roug[2]))

for i in range(0,len(z_canopy_vals)):
    if z_roug_vals[i] >= 0.7:
        z_canopy_vals[i] = z_topo_vals[i] + h_canopy
    else:
        z_canopy_vals[i] = z_topo_vals[i]

# »»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»
# Write "canopy_LOCAL.vtk" file

```

```
print('\nWriting canopy_LOCAL.vtk ...')

f = open('canopy_LOCAL.vtk','w')
f.write('# vtk DataFile Version 1.0\n')
f.write('CANOPY\n')
f.write('ASCII\n')
f.write('DATASET STRUCTURED_GRID\n')
f.write('DIMENSIONS %.0f %.0f %.0f\n' % (float(Nx),float(Ny),float(Nz)))
f.write('POINTS %.0f FLOAT\n' % float(points))

for i in range(0,len(x_topo_vals)):
    f.write('   %.8e    %.9e    %.8e\n'
    % (float(x_topo_vals[i]),float(y_topo_vals[i]),float(z_canopy_vals[i])))

f.write('\nPOINT_DATA %.0f\n\n' % float(points))
f.write('SCALARS Z FLOAT 1\n')
f.write('LOOKUP_TABLE default\n')

for i in range(0,len(z_topo_vals)):
    f.write('   %.8f\n' % (float(z_canopy_vals[i])))

f.close()

print('Program execution completed!')
```

B.4 vtk_to_stl.py

This algorithm is responsible to generate the STL file with the top canopy surface. In terms of procedure, the code reads the manipulated topography map with forest `canopy_LOCAL.vtk` and converts the three-dimensional structure into a polygon format, whose geometry is then written to the file `canopy_LOCAL.stl`. The source code of `vtk_to_stl.py` file is presented below.

```
# »»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»
# Variables

import sys
sys.path.append('~/.local/lib/python3.6/site-packages')
import vtk

# »»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»
# Load VTK file "canopy_LOCAL.vtk"

input_vtk_file = "./canopy_LOCAL.vtk"
reader = vtk.vtkStructuredGridReader()
reader.SetFileName(input_vtk_file)
reader.Update()

# »»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»
# Extract STL surface

surface_filter = vtk.vtkDataSetSurfaceFilter()
surface_filter.SetInputConnection(reader.GetOutputPort())
surface_filter.Update()

# »»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»
# Convert the surface to polygons

poly_data = vtk.vtkPolyData()
poly_data.ShallowCopy(surface_filter.GetOutput())

# »»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»
# Save in STL format

output_stl_file = "./canopy_LOCAL.stl"
stl_writer = vtk.vtkSTLWriter()
stl_writer.SetFileName(output_stl_file)
stl_writer.SetInputData(poly_data)
stl_writer.Write()

print('\nFile canopy_LOCAL.stl created!')
```

Appendix C

Purpose built scripts to streamline workflow

In this Appendix is presented the set of scripts developed to optimise the process of launching simulations in the INCD cloud environment, referring to the FCT advanced computing project 2023.10427.CPCA.A0.

In terms of procedure, simulations are submitted for processing with the *sbatch* command, so a script was created to execute this command for the 12 simulations, later called `AllSubmit.sh`, which is exemplified below for SET_{212}. Essentially it runs a for loop to execute the *sbatch* command for each wind direction, whose value is placed in the variable `i`.

```
#!/bin/bash
for i in ???
do
    cd $i
    sbatch SET_212_$i.sh
    cd ..
done
```

As an example for a wind direction of 0°, the command *sbatch* is applied to the script `SET_212_000.sh`, described below, which has the instructions for submitting the job. At this point, the number of nodes and tasks per node can be configured as desired, as well as other settings such as the job name. In the code, it is necessary to include the OpenFOAM module so that it can be loaded and used when running any native OpenFOAM command.

```
#!/bin/bash
#SBATCH --job-name=Teste_MORKNASKOGEN
#SBATCH --time=0:10:0
#SBATCH --partition=fct
#SBATCH --qos=cpca104272023
#SBATCH --output=%x.o%j
#SBATCH --error=%x.o%j
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1

### Prepare the environment
module purge
module load intel/openfoam/2306

echo hostname
./AllrunAll
```

In this case, and after defining all configurations and loading the necessary modules, the final purpose of the code `SET_212_000.sh` is to execute the script

`AllrunAll.sh`, which will proceed to execute all commands associated with mesh generation and to submit *simpleFoam* solver. Its contents is described below for the case of a simulation with a forest model, in particular corresponding to SET_{212}.

```
#!/bin/sh
cd ${0%/*} || exit 1

. $WM_PROJECT_DIR/bin/tools/RunFunctions

# PREPROC

cd _preproc/_gsurf_WINDIE
./gsrf3
./groug
cd ..

./write_blockMeshDictCano

cd ..
runApplication blockMesh
runApplication topoSet

cd _preproc
./write_z0
./write_bCsCano
./write_turbulenceProperties

cd vtk/
python3 Generate_canopy_vtk.py
python3 vtk_to_stl.py
cd ../../
cp -r ./_preproc/vtk/canopy_LOCAL.stl ./constant/triSurface/canopy_LOCAL.stl

# RUN
# OPÇÃO 1. CORRIDA EM PARALELO (4 procs, mudar decomposeParDict para usar outro número de procs)
#foamDictionary -entry "method" -set "hierarchical" system/decomposeParDict
#runApplication -o decomposePar
#runParallel -o $(getApplication)
runApplication simpleFoam
cp -r $(foamListTimes -latestTime) res_final
cp -r $(foamListTimes -latestTime) $(foamListTimes -latestTime)_
touch visu.foam

# OPÇÃO 1.1. RECONSTRUIR APENAS A ITERAÇÃO FINAL
#runApplication -o reconstructPar -constant -latestTime

# OPÇÃO 1.2. RECONSTRUIR VÁRIAS ITERAÇÕES (SELEÇÃO)
#runApplication -o reconstructPar -constant -t

# OPÇÃO 2. CORRIDA EM SÉRIE (apagar log.simpleFoam se existir, de outra forma não corre)
```

Appendix D

Significantly modified FORTRAN codes for pre-processing

In this Appendix are presented the source codes of the executable files *write_blockMeshDict* and *write_bCs* that were significantly changed to include simulations with a forest model. Both codes were inherited from precursor works in the field, as previously described in Appendix A.

D.1 `write_blockMeshDictCano.f90`

As described in Section 5.4, *write_blockMeshDictCano* script was modified to include the new mesh type `canopyHD`, which includes in the mesh a high resolution area below the canopy, in order to make this region better defined. These changes were performed in the respective source code file `write_blockMeshDictCano.f90` programmed in FORTRAN, whose modified code is presented below.

```
1  program write_blockMeshDict2
2
3    implicit none
4
5    !!$ calc
6
7    integer :: xdim,ydim,zdim,ndim,ni,nj,nk
8    integer :: n1,n2,n3,n4,li,le
9    integer :: i,j,k,l,m,n,o,p
10   real :: xmin,ymin,zmin,xmax,ymax,zmax
11
12   real :: hcano
13   integer :: nzcano, icano
14
15   real, allocatable, dimension(:) :: xvec,yvec,zvec
16   real, allocatable, dimension(:,:) :: xmat,ymat,zmat
17   integer, allocatable, dimension(:) :: block,ground,skyvec,inlet,outlet,front,back
18
19   !!$ reading engine
20
21   character*1 :: ichar,equal
22   character*100 :: leitura
23
24   real :: sky,R
25   integer :: nz
26
27   !!$ outros
28
29   real :: t1,t2
30
31   !!$ start
32
33   call cpu_time(t1)
34
35   !!$  call system('clear')
36
37   !!$ ###################################
38   !!$ Leitura do ficheiro de configuração
39   !!$ ###################################
```

```
40
41    open(1,file='_preprocDict')
42
43    !! $ Procura de 'sky'
44
45    p=1
46
47    do while (p.eq.1)
48      read(1,121) leitura
49      if(leitura(1:3).eq.'sky') then
50          read(leitura(4:100),*) equal,sky
51          p=2
52          rewind 1
53      endif
54    enddo
55
56    !! $ Procura de 'R'
57
58    p=1
59
60    do while (p.eq.1)
61      read(1,121) leitura
62      if(leitura(1:1).eq.'R') then
63          read(leitura(2:100),*) equal,R
64          p=2
65          rewind 1
66      endif
67    enddo
68
69    !! $ Procura de 'nz'
70
71    p=1
72
73    do while (p.eq.1)
74      read(1,121) leitura
75      if(leitura(1:2).eq.'nz') then
76          read(leitura(3:100),*) equal,nz
77          p=2
78          rewind 1
79      endif
80    enddo
81
82    !! $ Procura de 'icano'
83
84    p=1
85
86    do while (p.eq.1)
87      read(1,121) leitura
88      if(leitura(1:5).eq.'icano') then
89          read(leitura(6:100),*) equal,icano
90          p=2
91          rewind 1
92      endif
93    enddo
94
95    if (icano.eq.1) then
96
97    !! $ Procura de 'nzcano'
98
99    p=1
100
101   do while (p.eq.1)
102     read(1,121) leitura
103     if(leitura(1:6).eq.'nzcano') then
104         read(leitura(7:100),*) equal,nzcano
105         p=2
106         rewind 1
107     endif
108   enddo
109
110   !! $ Procura de 'hcano'
111
112   p=1
113
114   do while (p.eq.1)
115     read(1,121) leitura
116     if(leitura(1:5).eq.'hcano') then
117         read(leitura(6:100),*) equal,hcano
118         p=2
119         rewind 1
120     endif
121   enddo
122   else
123     hcano = 0.0
124     nzcano = 0
125   endif
```

```
126   !! $ Fim do motor de busca
127
128   close(1)
129
130   121  format(a100)
131
132   print*,''
133   print*,'-------------------------------------------------'
134   print*,'- GERAÇÃO DE FICHEIRO blockMeshDict PARA OpenFOAM -'
135   print*,'-------------------------------------------------'
136   print*,'-'
137   print*,'- Cota de "céu" [m]              : ',sky
138   print*,'- Nž de células em z             : ',nz
139   print*,'- Fator de expansão R            : ',R
140   print*,'- Flag de floresta               : ',icano
141   print*,'- Nž de células abaixo de hcano  : ',nzcano
142   print*,'- Altura da canópia              : ',hcano
143
144   !!$ ########################################
145   !!$ Leitura do ficheiro VTK criado pelo GSURF
146   !!$ ########################################
147
148   print*,'- '
149   print*,'-----------------------------------------'
150   print*,'- '
151   print*,'- A ler ficheiro ".vtk"...'
152
153   open(1,file='vtk/topo_LOCAL.vtk')
154
155   !! $ Procura de 'DIMENSIONS'
156
157   p=1
158
159   j=1
160
161   do while (p.eq.1)
162     read(1,121) leitura
163     if(leitura(j:j+10).eq.'DIMENSIONS') then
164       read(leitura(12:50),*) xdim,ydim,zdim
165       ni = xdim - 1
166       nj = ydim - 1
167       nk = zdim - 1
168       p=2
169     endif
170   enddo
171
172   !!$ Leitura das coordenadas
173
174   read(1,121) leitura    !Salta linha
175
176   allocate(xvec(xdim*ydim),yvec(xdim*ydim),zvec(xdim*ydim))
177
178   do i=1,xdim*ydim
179           read(1,*) xvec(i),yvec(i),zvec(i)
180   enddo
181
182   close(1)
183
184   !!$ Resumo de dados
185
186   xmin = minval(xvec)
187   ymin = minval(yvec)
188   zmin = minval(zvec)
189
190   !!$ Transformação de vetores em matrizes
191
192   allocate(xmat(xdim,ydim),ymat(xdim,ydim),zmat(xdim,ydim))
193
194   xmat = reshape(xvec,(/xdim,ydim/))
195
196   ymat = reshape(yvec,(/xdim,ydim/))
197
198   zmat = reshape(zvec,(/xdim,ydim/))
199
200   !!$ ########################################
201   !!$ Escrita de ficheiro blockMeshDict
202   !!$ ########################################
203
204   !!$ A escrever cabeçalho
205
206   print*,'- '
207   print*,'- A escrever ficheiro "blockMeshDict"...'
208
209   open(1,file='blockMeshDict')
210
```

```
211   200  format('*---------------------------------*- C++ -*----------------------------------*')
212   201  format('| =========                 |                                                 |')
213   202  format('| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |')
214   203  format('|  \\    /   O peration     | Version:  2.0.0                                 |')
215   204  format('|   \\  /    A nd           | Web:      www.OpenFOAM.com                      |')
216   205  format('|    \\/     M anipulation  |                                                 |')
217   206  format('\*-------------------------------------------------------------------------*/')
218   207  format('FoamFile')
219   208  format('{')
220   209  format('    version     2.0;')
221   210  format('    format      ascii;')
222   211  format('    class       dictionary;')
223   212  format('    object      blockMeshDict;')
224   213  format('}')
225   214  format('// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //')
226
227
228   write(1,200)
229   write(1,201)
230   write(1,202)
231   write(1,203)
232   write(1,204)
233   write(1,205)
234   write(1,206)
235   write(1,207)
236   write(1,208)
237   write(1,209)
238   write(1,210)
239   write(1,211)
240   write(1,212)
241   write(1,213)
242   write(1,214)
243   write(1,*) ''
244
245   !!$ !!$ ######################################### Escrita de VERTICES
246
247   print*,'- '
248   print*,'- A escrever vértices...'
249
250   215  format('scale 1;')
251   write(1,215)
252   write(1,*) ''
253   216  format('vertices')
254   write(1,216)
255   217  format('(')
256   write(1,217)
257
258   !!$ Write vertices sweeping i, j then k
259   !!$ Ground
260   do j = 1, ydim
261     do i = 1, xdim
262       write(1,218) xmat(i,j),ymat(i,j),zmat(i,j)
263     end do
264   end do
265
266  !!$ Sky
267   do j = 1, ydim
268     do i = 1, xdim
269       write(1,218) xmat(i,j),ymat(i,j),sky
270     end do
271   end do
272
273  !!$ Top of forest if present
274   if (icano.eq.1) then
275     do j = 1, ydim
276       do i = 1, xdim
277         write(1,218) xmat(i,j),ymat(i,j),zmat(i,j)+hcano
278       end do
279     end do
280   endif
281
282   218  format('(',f12.2,1x,f12.2,1x,f12.2,' )')
283   219  format(');')
284   write(1,219)
285   write(1,*) ''
286
287   !!$ !!$ !!$ ######################################### Escrita de BLOCKS
288
289   allocate(block(8))
290
291   print*,'- '
292   print*,'- A escrever blocos...'
293
294   220  format('blocks')
295   write(1,220)
```

```
296   221  format('(')
297   write(1,221)
298
299   !!$ ---------- Escrita de blocos (ordenados por i, j)
300
301   if (icano.eq.0) then
302
303     ndim = xdim * ydim
304     do j = 1, nj
305       do i = 1, ni
306         block(1) = i+(j-1)*xdim-1
307         block(2) = block(1)+1
308         block(3) = block(2)+xdim
309         block(4) = block(1)+xdim
310         block(5) = block(1)+ndim
311         block(6) = block(2)+ndim
312         block(7) = block(3)+ndim
313         block(8) = block(4)+ndim
314
315         write(1,223) block,nz,R
316       end do
317     end do
318   else
319     ndim = xdim * ydim
320     !!$ Blocks inside forest
321     do j = 1, nj
322       do i = 1, ni
323         block(1) = i+(j-1)*xdim-1
324         block(2) = block(1)+1
325         block(3) = block(2)+xdim
326         block(4) = block(1)+xdim
327         block(5) = block(1)+2*ndim
328         block(6) = block(2)+2*ndim
329         block(7) = block(3)+2*ndim
330         block(8) = block(4)+2*ndim
331
332         write(1,223) block,nzcano,1.0
333       end do
334     end do
335
336     !!$ Blocks above forest
337     do j = 1, nj
338       do i = 1, ni
339         block(1) = i+(j-1)*xdim-1 + 2*ndim
340         block(2) = block(1)+1
341         block(3) = block(2)+xdim
342         block(4) = block(1)+xdim
343         block(5) = block(1)-ndim
344         block(6) = block(2)-ndim
345         block(7) = block(3)-ndim
346         block(8) = block(4)-ndim
347
348         write(1,223) block,(nz-nzcano),R
349
350       end do
351     end do
352   end if
353
354   223  format('    hex ( ',i9,i9,i9,i9,i9,i9,i9,i9,' ) (1 1 ',i6,') simplegrading (1 1 ',f8.4,')')
355   224  format(');')
356   write(1,224)
357   write(1,*) ''
358
359   !!$ !!$ !!$ ######################################### Escrita de BOUNDARY
360
361   print*,'- '
362   print*,'- A escrever fronteiras...'
363
364   allocate(ground(4),skyvec(4),inlet(4),outlet(4),front(4),back(4))
365
366   225  format('boundary')
367   write(1,225)
368   226  format('(')
369   write(1,226)
370
371   !!$ ---------- ## ground ##
372
373   227  format('    ground')
374   write(1,227)
375   228  format('    {')
376   write(1,228)
377   229  format('        type wall;')
378   write(1,229)
379   230  format('        faces')
380   write(1,230)
```

```
231  format('        (')
write(1,231)

do j = 1, nj
   do i = 1, ni
      ground(1) = i+(j-1)*xdim - 1
      ground(2) = ground(1)+xdim
      ground(3) = ground(2) + 1
      ground(4) = ground(1) + 1

      write(1,234) ground
   end do
end do

234  format('        ( ',i9,i9,i9,i9,' )')

235  format('        );')
write(1,235)
236  format('    }')
write(1,236)

!!$ ---------- ## sky ##

237  format('    sky')
write(1,237)
238  format('    {')
write(1,238)
239  format('        type symmetryPlane;')
write(1,239)
240  format('        faces')
write(1,240)
241  format('        (')
write(1,241)

do j = 1, nj
  do i = 1, ni
     skyvec(1) = i+(j-1)*xdim + ndim - 1
     skyvec(2) = skyvec(1) + 1
     skyvec(3) = skyvec(2) + xdim
     skyvec(4) = skyvec(3) - 1

     write(1,234) skyvec
  end do
end do

243  format('        );')
write(1,243)
244  format('    }')
write(1,244)

!!$ ---------- ## inlet ##

245  format('    inlet')
write(1,245)
246  format('    {')
write(1,246)
247  format('        type patch;')
write(1,247)
248  format('        faces')
write(1,248)
249  format('        (')
write(1,249)

if (icano.eq.0) then

  do j = 1, nj
       inlet(1) = 1 + (j-1)*xdim - 1
       inlet(2) = inlet(1) + ndim
       inlet(3) = inlet(2) + xdim
       inlet(4) = inlet(1) + xdim

       write(1,234) inlet
  end do

else

  !!$ inlet faces below forest height
  do j = 1, nj
       inlet(1) = 1 + (j-1)*xdim - 1
       inlet(2) = inlet(1) + 2*ndim
       inlet(3) = inlet(2) + xdim
       inlet(4) = inlet(1) + xdim

       write(1,234) inlet
  end do
```

```
466
467       !!$ inlet faces above forest height
468       do j = 1, nj
469            inlet(1) = 1 + (j-1)*xdim + 2*ndim - 1
470            inlet(2) = inlet(1) - ndim
471            inlet(3) = inlet(2) + xdim
472            inlet(4) = inlet(1) + xdim
473
474            write(1,234) inlet
475       end do
476     end if
477
478     252  format('        );')
479     write(1,252)
480     253  format('    }')
481     write(1,253)
482
483     !!$ ---------- ## outlet ##
484
485     254  format('    outlet')
486     write(1,254)
487     255  format('    {')
488     write(1,255)
489     256  format('        type patch;')
490     write(1,256)
491     257  format('        faces')
492     write(1,257)
493     258  format('        (')
494     write(1,258)
495
496     if (icano.eq.0) then
497
498       do j = 1, nj
499            outlet(1) = xdim + (j-1)*xdim - 1
500            outlet(2) = outlet(1) + xdim
501            outlet(3) = outlet(2) + ndim
502            outlet(4) = outlet(1) + ndim
503
504            write(1,234) outlet
505       end do
506
507     else
508
509       !!$ inlet faces below forest height
510       do j = 1, nj
511            outlet(1) = xdim + (j-1)*xdim - 1
512            outlet(2) = outlet(1) + xdim
513            outlet(3) = outlet(2) + 2*ndim
514            outlet(4) = outlet(1) + 2*ndim
515
516            write(1,234) outlet
517       end do
518
519       !!$ inlet faces above forest height
520       do j = 1, nj
521           outlet(1) = xdim + (j-1)*xdim + 2*ndim - 1
522            outlet(2) = outlet(1) + xdim
523            outlet(3) = outlet(2) - ndim
524            outlet(4) = outlet(1) - ndim
525
526            write(1,234) outlet
527       end do
528
529     end if
530
531     259  format('        );')
532     write(1,259)
533     260  format('    }')
534     write(1,260)
535
536    !!$ ---------- ## front ##
537
538     261  format('    front')
539     write(1,261)
540     262  format('    {')
541     write(1,262)
542     263  format('        type symmetryPlane;')
543     write(1,263)
544     264  format('        faces')
545     write(1,264)
546     265  format('        (')
547     write(1,265)
548
549     if (icano.eq.0) then
550
551       do i = 1, ni
```

```
552           front(1) = i - 1
553           front(2) = front(1) + 1
554           front(3) = front(2) + ndim
555           front(4) = front(1) + ndim
556
557         write(1,234) front
558       end do
559
560     else
561
562       !!$ front faces below forest height
563       do i = 1, ni
564           front(1) = i - 1
565           front(2) = front(1) + 1
566           front(3) = front(2) + 2*ndim
567           front(4) = front(1) + 2*ndim
568
569           write(1,234) front
570       end do
571
572       !!$ front faces above forest height
573       do i = 1, ni
574           front(1) = i + 2*ndim - 1
575           front(2) = front(1) + 1
576           front(3) = front(2) - ndim
577           front(4) = front(1) - ndim
578
579           write(1,234) front
580       end do
581
582     end if
583
584     267  format('        );')
585     write(1,267)
586     268  format('    }')
587     write(1,268)
588
589     !!$ ---------- ## back ##
590
591     269  format('    back')
592     write(1,269)
593     270  format('    {')
594     write(1,270)
595     271  format('        type symmetryPlane;')
596     write(1,271)
597     272  format('        faces')
598     write(1,272)
599     273  format('        (')
600     write(1,273)
601
602     if (icano.eq.0) then
603
604       do i = 1, ni
605            back(1) = ndim - xdim + i - 1
606            back(2) = back(1) + ndim
607            back(3) = back(2) + 1
608            back(4) = back(1) + 1
609
610            write(1,234) back
611       end do
612
613     else
614
615       !!$ inlet faces below forest height
616       do i = 1, ni
617           back(1) = ndim - xdim + i - 1
618           back(2) = back(1) + 2*ndim
619           back(3) = back(2) + 1
620           back(4) = back(1) + 1
621
622           write(1,234) back
623       end do
624
625       !!$ inlet faces above forest height
626       do i = 1, ni
627           back(1) = ndim - xdim + i - 1 + 2*ndim
628           back(2) = back(1) - ndim
629           back(3) = back(2) + 1
630           back(4) = back(1) + 1
631
632           write(1,234) back
633       end do
634
635     end if
636
```

```
637    274  format('        );')
638    write(1,274)
639    275  format('    }')
640    write(1,275)
641
642    276  format(');')
643    write(1,276)
644    277  format('')
645    write(1,277)
646
647    !!$ Escrita de MERGEPATCHPAIRS
648
649    278  format('mergePatchPairs')
650    write(1,278)
651    279  format('(')
652    write(1,279)
653    280  format(');')
654    write(1,280)
655    281  format('')
656    write(1,281)
657
658    !!$ ###
659    !!$ FIM
660    !!$ ###
661
662    close(1)
663
664    call system('mv -f blockMeshDict ../system')
665
666    !!$ stop
667
668    12 format(' - Terminado em ',f6.2,' s')
669
670    call cpu_time(t2)
671
672    print*,'-'
673    write(*,12) t2-t1
674    print*,'-'
675    print*,'---------------------------------------------'
676    print*,''
677
678  end program write_blockMeshDict2
```

D.2 `write_bCsCano.f90`

As a consequence of the new mesh generation type `canopyHD` that was added to the *write_blockMeshDictCano* executable, it was necessary to modify the boundary conditions writing strategy so that *write_bCs* could handle two hexahedrons for each column of control volumes. These changes were performed in the respective source code file `write_bCsCano.f90` programmed in FORTRAN, whose modified code is presented below.

```
program write_bCs_Cano

  implicit none

  !!$ calc

  integer :: nfaces,npoints,startface
  integer :: i,j,ii,p,k,l,m,ni,nj,nk,nig,njg,nz,nic,njc,niv,njv,nkv,xdim,ydim,zdim

  integer, allocatable, dimension(:,:) :: faces
  real, allocatable, dimension(:) :: x,y,z
  real, allocatable, dimension(:) :: xp,yp,zp,zoff,zgroundp
  real, allocatable, dimension(:) :: xvec,yvec,zvec,yinlet,zinlet
  real, allocatable, dimension(:) :: u,kappa,ep,omega
  real, allocatable, dimension(:,:) :: zs
  real, allocatable, dimension(:,:) :: xmat,ymat,zmat
  real :: kint,kw,epsint,epsw,omegaint,omegaw
  real :: xmin,ymin,zmin,xmax,ymax,zmax

  !!$ reading engine

  character*1 :: ichar,equal
  character*100 :: leitura
  real :: ustar,z0,delta,kv,cmu,zrs

  integer :: istat,nlines,ierror
  character (len=70), dimension(100) :: line
  character name*100

  !!$ outros

  real :: t1,t2

  !!$ Functions
  real :: interp1D,stringparser

  !!$ start

  call cpu_time(t1)

  !!$  call system('clear')

  !!$ ################################################
  !!$ Leitura do ficheiro de configuração _preprocDict
  !!$ ################################################

  open(1,file='_preprocDict')

 !! $ Procura de 'nz'

  p=1
  do while (p.eq.1)
    read(1,121) leitura
    if(leitura(1:2).eq.'nz') then
        read(leitura(3:100),*) equal,nz
        p=2
        rewind 1
    endif
  enddo

  !! $ Procura de 'ustar'

  p=1
  do while (p.eq.1)
      read(1,121) leitura
      if(leitura(1:5).eq.'ustar') then
          read(leitura(6:100),*) equal,ustar
          p=2
          rewind 1
      endif
  enddo
```

```
72
73    !! $ Procura de 'z0i'
74
75    p=1
76    do while (p.eq.1)
77        read(1,121) leitura
78        if(leitura(1:3).eq.'z0i') then
79            read(leitura(4:100),*) equal,z0
80            p=2
81            rewind 1
82        endif
83    enddo
84
85    !! $ Procura de 'delta'
86
87    p=1
88    do while (p.eq.1)
89        read(1,121) leitura
90        if(leitura(1:5).eq.'delta') then
91            read(leitura(6:100),*) equal,delta
92            p=2
93            rewind 1
94        endif
95    enddo
96
97   !! $ Procura de 'kv'
98
99    p=1
100   do while (p.eq.1)
101       read(1,121) leitura
102       if(leitura(1:2).eq.'kv') then
103           read(leitura(3:100),*) equal,kv
104           p=2
105           rewind 1
106       endif
107   enddo
108
109   !! $ Procura de 'cmu'
110
111   p=1
112   do while (p.eq.1)
113       read(1,121) leitura
114       if(leitura(1:3).eq.'cmu') then
115           read(leitura(4:100),*) equal,cmu
116           p=2
117           rewind 1
118       endif
119   enddo
120
121   !! $ Procura de 'zrs'
122
123   p=1
124   do while (p.eq.1)
125       read(1,121) leitura
126       if(leitura(1:3).eq.'zrs') then
127           read(leitura(4:100),*) equal,zrs
128           p=2
129           rewind 1
130       endif
131   enddo
132
133   !! $ Fim do motor de busca
134
135   close(1)
136
137   121  format(a100)
138
139  !!$ ************************************************
140  !!$ Leitura do ficheiro de configuração windie.cfg
141  !!$ ************************************************
142
143  !!$ ############ read WINDIE.CFG - configuration file ############
144    open(1,file='_gsurf_WINDIE/windie.cfg',form='formatted')
145    nlines = 0
146    do
147       read(1,'(a70)',iostat=istat) line(nlines + 1)
148       if(istat < 0 ) EXIT
149       nlines = nlines + 1
150    enddo
151    close(1)
152
153    nig = int(stringparser('nig',line,ierror,'WINDIE.CFG'))
154    if (ierror.ne.0) then
155       write(*,*) 'Error parsing ''nig''! Aborting'
156       STOP
157    endif
```

```
  njg = int(stringparser('njg',line,ierror,'WINDIE.CFG'))
  if (ierror.ne.0) then
     write(*,*) 'Error parsing ''njg''! Aborting'
     STOP
  endif

  niv = nig - 1
  njv = njg - 1
  nkv = nz + 1

  nic = niv - 1
  njc = njv - 1

!!$ ########################################
!!$ Leitura do ficheiro VTK criado pelo GSURF
!!$ ########################################

  print*,'- '
  print*,'-----------------------------------------------'
  print*,'- '
  print*,'- A ler ficheiro ".vtk"...'

  open(1,file='vtk/topo_LOCAL.vtk')

!! $ Procura de 'DIMENSIONS'

  p=1
  j=1

  do while (p.eq.1)
    read(1,121) leitura
    if(leitura(j:j+10).eq.'DIMENSIONS') then
        read(leitura(12:50),*) xdim,ydim,zdim
        ni = xdim - 1
        nj = ydim -
        nk = zdim - 1
        p=2
    endif
  enddo

  !!$ Leitura das coordenadas
  read(1,121) leitura    !Salta linha
  allocate(xvec(xdim*ydim),yvec(xdim*ydim),zvec(xdim*ydim))
  do i=1,xdim*ydim
    read(1,*) xvec(i),yvec(i),zvec(i)
  enddo
  close(1)

  !!$ Resumo de dados
  xmin = minval(xvec)
  ymin = minval(yvec)
  zmin = minval(zvec)

  !!$ Transformação de vetores em matrizes
  allocate(xmat(xdim,ydim),ymat(xdim,ydim),zmat(xdim,ydim))
  allocate(yinlet(ydim),zinlet(ydim))

  xmat = reshape(xvec,(/xdim,ydim/))
  ymat = reshape(yvec,(/xdim,ydim/))
  zmat = reshape(zvec,(/xdim,ydim/))

!!$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
  print*,''
  print*,'-----------------------------------------------'
  print*,'- GERAÇÃO DE PERFIS DE ENTRADA PARA OpenFOAM -'
  print*,'-----------------------------------------------'
  print*,'-'
  write(*,1231) ustar
  write(*,1232) z0
  write(*,1233) delta
  write(*,1234) kv
  write(*,1235) cmu

1231 format(' - u*      : ',f10.4)
1232 format(' - z0      : ',f10.4)
1233 format(' - delta   : ',f10.2)
1234 format(' - kv      : ',f10.4)
1235 format(' - Cmu     : ',f10.4)

!!$ ########################################
!!$ Leitura do ficheiro boundary
!!$ ########################################
  print*,'- '
  print*,'-----------------------------------------------'
  print*,'- '
```

```
243   print*,'- A ler ficheiro "boundary"...'
244   open(1,file='../constant/polyMesh/boundary')
245
246  !!---------------$ Procura de 'inlet'
247
248   p=1
249   j=1
250
251   do while (p.eq.1)
252       read(1,121) leitura
253       if(leitura(j:j+9).eq.'    inlet') then
254           p=2
255           endif
256   enddo
257
258  !!---------------$ Procura de 'nfaces'
259
260   p=1
261   j=1
262
263   do while (p.eq.1)
264       read(1,121) leitura
265       if(leitura(1:15).eq.'        nFaces') then
266           do while (p.eq.1)
267               if(leitura(j:j).eq.';') then
268                 read(leitura(j-10:j),*) nfaces
269                 p=2
270               else
271                 j=j+1
272               endif
273           enddo
274       endif
275   enddo
276
277  !!---------------$ Procura de 'startFace'
278
279   allocate(faces(nfaces,4))
280
281   p=1
282   j=1
283
284   do while (p.eq.1)
285       read(1,121) leitura
286       if(leitura(1:18).eq.'        startFace') then
287           do while (p.eq.1)
288               if(leitura(j:j).eq.';') then
289                   read(leitura(j-7:j),*) startface
290                   p=2
291               else
292                   j=j+1
293               endif
294           enddo
295       endif
296   enddo
297   close(1)
298
299  !!$ ########################################
300  !!$ Leitura do ficheiro faces
301  !!$ ########################################
302
303   print*,'- '
304   print*,'- A ler ficheiro "faces"...'
305
306   open(2,file='../constant/polyMesh/faces')
307
308    !!---------------$ Procura de face nž 0
309
310   p=1
311   j=1
312
313   do while (p.eq.1)
314       read(2,121) leitura
315       if(leitura(1:2).eq.'4(') then
316           p=2
317       endif
318   enddo
319
320    !!---------------$ Leitura de faces startFace até startFace + nFaces
321
322   if(startface.ne.0) then
323       do i=1,startface-1
324           read(2,121) leitura
325       enddo
326   endif
327
```

```
328   do i=1,nfaces
329       read(2,121) leitura
330       p=1
331       j=1
332       do while (p.eq.1)
333           if(leitura(j:j).eq.')') then
334               p=2
335           else
336               j=j+1
337           endif
338       enddo
339       read(leitura(3:j-1),*) faces(i,1),faces(i,2),faces(i,3),faces(i,4)
340   enddo
341
342   close(2)
343
344   !!$ ########################################
345   !!$ Leitura do ficheiro points
346   !!$ ########################################
347
348   print*,'- '
349   print*,'- A ler ficheiro "points"...'
350
351   open(3,file='../constant/polyMesh/points')
352
353   !!---------------$ Procura de ponto nž 0
354
355   p=1
356   do while (p.eq.1)
357       read(3,121) leitura
358       if(leitura(1:1).eq.'(') then
359           p=2
360       endif
361   enddo
362
363   !!---------------$ Determinação de npoints
364
365   p=1
366   npoints=0
367   do while (p.eq.1)
368       read(3,121) leitura
369       if(leitura(1:1).eq.'(') then
370           npoints = npoints+1
371       else
372           p=2
373       endif
374   enddo
375
376   !!---------------$ Leitura de coordenadas de todos os pontos
377
378   allocate(x(npoints),y(npoints),z(npoints))
379
380   rewind 3
381   p=1
382   do while (p.eq.1)
383       read(3,121) leitura
384       if(leitura(1:1).eq.'(') then
385           p=2
386       endif
387   enddo
388
389   p=1
390   do i=1,npoints
391       p=1
392       j=1
393       read(3,121) leitura
394       do while (p.eq.1)
395           if(leitura(j:j).eq.')') then
396               p=2
397           else
398               j=j+1
399           endif
400       enddo
401       read(leitura(2:j-1),*) x(i),y(i),z(i)
402   enddo
403
404   close(3)
405
406   !!$ Escrita de vetores com os pontos médios das faces
407
408   allocate(xp(nfaces),yp(nfaces),zp(nfaces))
409   allocate(zoff(nfaces),zgroundp(nfaces))
410
411   ii=1
412
```

```
413   do i=1,nfaces
414
415       xp(ii)=0.
416       yp(ii)=0.
417       zp(ii)=0.
418
419       do j=1,4
420           xp(ii)=xp(ii)+x(faces(i,j)+1)
421           yp(ii)=yp(ii)+y(faces(i,j)+1)
422           zp(ii)=zp(ii)+z(faces(i,j)+1)
423       enddo
424
425       xp(ii)=xp(ii)/4.
426       yp(ii)=yp(ii)/4.
427       zp(ii)=zp(ii)/4.
428
429       yinlet = ymat(1,:)
430       zinlet = zmat(1,:)
431       zgroundp(ii) = interp1D(size(yinlet),yinlet,zinlet,yp(ii))
432       zoff(ii) = zp(ii) - zgroundp(ii)
433
434       !!$ write(*,*) ii,xp(ii),yp(ii),zp(ii),zoff(ii)
435       ii=ii+1
436   enddo
437
438   !!$ Cálculo das condições de entrada como em 'Castro, 2010'
439
440   print*,'- '
441   print*,'- A gerar condições de entrada...'
442
443   !!$ ------------ Perfil u(z)
444
445   allocate(u(nfaces))
446
447   do i=1,nfaces
448       if (zoff(i).lt.delta) then
449           u(i) = (ustar/kv)*log((z0+zoff(i))/z0)
450       else
451           u(i) = (ustar/kv)*log((z0+delta)/z0)
452       endif
453   enddo
454
455   !!$ ------------ Perfil k(z)
456
457   allocate(kappa(nfaces))
458
459   do i=1,nfaces
460       if (zoff(i).le.0.99*delta) then
461           kappa(i) = ((ustar**2.)/(cmu**.5))*(1.-zoff(i)/delta)
462       else
463           kappa(i) = (ustar**2.)/(100.*cmu**.5)
464       endif
465   enddo
466
467   !!$ ------------ Perfil epsilon(z)
468
469   allocate(ep(nfaces))
470
471   do i=1,nfaces
472       if (zoff(i).le.0.95*delta) then
473           ep(i) = (cmu**.75)*(kappa(i))**1.5/(kv*zoff(i))
474       else
475           ep(i) = (cmu**.75)*(kappa(i))**1.5/(0.95*delta*kv)
476       endif
477   enddo
478
479   !!$ ------------ Perfil omega(z)
480
481   allocate(omega(nfaces))
482
483   do i=1,nfaces
484       omega(i) = ep(i)/(cmu*kappa(i))
485   enddo
486
487   !!$ ####################################
488   !!$ Escrita de ficheiros
489   !!$ ####################################
490
491   !!$ Escrita de ficheiros ".dat"
492
493   print*,'- '
494   print*,'- A escrever ficheiros ".dat"...'
495
496   !!$ ------------ Escrita de u.dat
497
```

```
498   open(1,file='u.dat')
499
500   !!open(2,file='u_gplot_abs.dat')
501
502   open(3,file='u_gplot_rel.dat')
503
504   do i=1,nfaces
505       write(1,10) u(i),0.,0.
506       !!write(2,11) zp(i),u(i)
507       write(3,11) zoff(i),u(i)
508   end do
509
510   close(1)
511
512   !!close(2)
513
514   close(3)
515
516   call system('mv -f u.dat ../0/dat')
517
518   !!call system('mv -f u_gplot_abs.dat plot')
519
520   call system('mv -f u_gplot_rel.dat plot')
521
522   !!$ ------------ Escrita de k.dat
523
524   open(1,file='k.dat')
525
526   !!open(2,file='k_gplot_abs.dat')
527
528   open(3,file='k_gplot_rel.dat')
529
530   do i=1,nfaces
531       write(1,12) kappa(i)
532       !!write(2,11) zp(i),kappa(i)
533       write(3,11) zoff(i),kappa(i)
534   end do
535
536   close(1)
537
538   !!close(2)
539
540   close(3)
541
542   call system('mv -f k.dat ../0/dat')
543
544   !!call system('mv -f k_gplot_abs.dat plot')
545
546   call system('mv -f k_gplot_rel.dat plot')
547
548   !!$ ------------ Escrita de epsilon.dat
549
550   open(1,file='epsilon.dat')
551
552   !!open(2,file='epsilon_gplot_abs.dat')
553
554   open(3,file='epsilon_gplot_rel.dat')
555
556   do i=1,nfaces
557       write(1,12) ep(i)
558       !!write(2,11) zp(i),ep(i)
559       write(3,11) zoff(i),ep(i)
560   end do
561
562   close(1)
563
564   !!close(2)
565
566   close(3)
567
568   call system('mv -f epsilon.dat ../0/dat')
569
570   !!call system('mv -f epsilon_gplot_abs.dat plot')
571
572   call system('mv -f epsilon_gplot_rel.dat plot')
573
574   !!$ ------------ Escrita de omega.dat
575
576   open(1,file='omega.dat')
577
578   !!open(2,file='omega_gplot_abs.dat')
579
580   open(3,file='omega_gplot_rel.dat')
581
582   do i=1,nfaces
```

```
583         write(1,12) omega(i)
584         !!write(2,11) zp(i),omega(i)
585         write(3,11) zoff(i),omega(i)
586     end do
587
588     close(1)
589
590     !!close(2)
591
592     close(3)
593
594     call system('mv -f omega.dat ../0/dat')
595
596     !!call system('mv -f omega_gplot_abs.dat plot')
597
598     call system('mv -f omega_gplot_rel.dat plot')
599
600     !!$ Escrita de ficheiro "k"
601
602     kint = ustar**2./cmu**0.5
603     kw = ustar**2./cmu**0.5
604
605     open(1,file='k')
606
607     2001  format('*---------------------------------*- C++ -*----------------------------------*')
608     2011  format('| =========                 |                                                 |')
609     2021  format('| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |')
610     2031  format('|  \\    /   O peration     | Version:  2.0.0                                 |')
611     2041  format('|   \\  /    A nd           | Web:      www.OpenFOAM.com                      |')
612     2051  format('|    \\/     M anipulation  |                                                 |')
613     2061  format('\*---------------------------------------------------------------------------*/')
614     2071  format('FoamFile')
615     2081  format('{')
616     2091  format('    version     2.0;')
617     2101  format('    format      ascii;')
618     2111  format('    class       volScalarField;')
619     2121  format('    location    "0";')
620     2131  format('    object      k;')
621     2141  format('}')
622     2151  format('// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //')
623
624     write(1,2001)
625     write(1,2011)
626     write(1,2021)
627     write(1,2031)
628     write(1,2041)
629     write(1,2051)
630     write(1,2061)
631     write(1,2071)
632     write(1,2081)
633     write(1,2091)
634     write(1,2101)
635     write(1,2111)
636     write(1,2121)
637     write(1,2131)
638     write(1,2141)
639     write(1,2151)
640     write(1,*) ''
641
642     2161  format('dimensions      [0 2 -2 0 0 0 0];')
643     write(1,2161)
644     write(1,*) ''
645     2171  format('internalField   uniform ',f6.3,';')
646     write(1,2171) kint
647     write(1,*) ''
648     2181  format('boundaryField')
649     write(1,2181)
650     2191  format('{')
651     write(1,2191)
652     2201  format('    inlet')
653     write(1,2201)
654     2211  format('    {')
655     write(1,2211)
656     2221  format('        type            fixedValue;')
657     write(1,2221)
658     2231  format('        value           nonuniform')
659     write(1,2231)
660     2232  format('        (#include "dat/k.dat");')
661     write(1,2232)
662     2241  format('    }')
663     write(1,2241)
664     write(1,*) ''
665     2251  format('    outlet')
666     write(1,2251)
667     2212  format('    {')
```

```
668   write(1,2212)
669   2261  format('        type            zeroGradient;')
670   write(1,2261)
671   2281  format('    }')
672   write(1,2281)
673   write(1,*) ''
674   2291  format('    front')
675   write(1,2291)
676   2301  format('    {')
677   write(1,2301)
678   2311  format('        type            symmetryPlane;')
679   write(1,2311)
680   2321  format('    }')
681   write(1,2321)
682   write(1,*) ''
683   2332  format('    back')
684   write(1,2332)
685   2341  format('    {')
686   write(1,2341)
687   2351  format('        type            symmetryPlane;')
688   write(1,2351)
689   2361  format('    }')
690   write(1,2361)
691   write(1,*) ''
692   2371  format('    ground')
693   write(1,2371)
694   2381  format('    {')
695   write(1,2381)
696   2391  format('        type            kqRWallFunction;')
697   write(1,2391)
698   2401  format('        value           uniform',f6.3,';')
699   write(1,2401) kw
700   2411  format('    }')
701   write(1,2411)
702   write(1,*) ''
703   2452  format('    sky')
704   write(1,2452)
705   2462  format('    {')
706   write(1,2462)
707   2472  format('        type            symmetryPlane;')
708   write(1,2472)
709   2482  format('    }')
710   write(1,2482)
711   write(1,*) ''
712   2412  format('}')
713   write(1,2412)
714   write(1,*) ''
715   write(1,2151)
716
717   close(1)
718
719   call system('mv -f k ../0')
720
721   !!$ Escrita de ficheiro "epsilon"
722
723   epsint = cmu**0.75*kint**1.5/(kv*zrs)
724   epsw = ustar**3/(kv*(z0))
725
726   open(1,file='epsilon')
727
728   2003  format('*---------------------------------*- C++ -*----------------------------------*')
729   2013  format('| =========                 |                                                 |')
730   2023  format('| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |')
731   2033  format('|  \\    /   O peration     | Version:  2.0.0                                 |')
732   2043  format('|   \\  /    A nd           | Web:      www.OpenFOAM.com                      |')
733   2053  format('|    \\/     M anipulation  |                                                 |')
734   2063  format('\*-------------------------------------------------------------------------*/')
735   2073  format('FoamFile')
736   2083  format('{')
737   2093  format('    version     2.0;')
738   2103  format('    format      ascii;')
739   2113  format('    class       volScalarField;')
740   2123  format('    location    "0";')
741   2133  format('    object      epsilon;')
742   2143  format('}')
743   2153  format('// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //')
744
745   write(1,2003)
746   write(1,2013)
747   write(1,2023)
748   write(1,2033)
749   write(1,2043)
750   write(1,2053)
751   write(1,2063)
752   write(1,2073)
```

```
753    write(1,2083)
754    write(1,2093)
755    write(1,2103)
756    write(1,2113)
757    write(1,2123)
758    write(1,2133)
759    write(1,2143)
760    write(1,2153)
761    write(1,*) ''
762
763    2163  format('dimensions      [0 2 -3 0 0 0 0];')
764    write(1,2163)
765    write(1,*) ''
766    2173  format('internalField   uniform ',f6.3,';')
767    write(1,2173) epsint
768    write(1,*) ''
769    2183  format('boundaryField')
770    write(1,2183)
771    2193  format('{')
772    write(1,2193)
773    2203  format('    inlet')
774    write(1,2203)
775    2213  format('    {')
776    write(1,2213)
777    2223  format('        type            fixedValue;')
778    write(1,2223)
779    2233  format('        value           nonuniform')
780    write(1,2233)
781    2234  format('        (#include "dat/epsilon.dat");')
782    write(1,2234)
783    2243  format('    }')
784    write(1,2243)
785    write(1,*) ''
786    2253  format('    outlet')
787    write(1,2253)
788    2214  format('    {')
789    write(1,2214)
790    2263  format('        type            zeroGradient;')
791    write(1,2263)
792    2283  format('    }')
793    write(1,2283)
794    write(1,*) ''
795    2293  format('    front')
796    write(1,2293)
797    2303  format('    {')
798    write(1,2303)
799    2313  format('        type            symmetryPlane;')
800    write(1,2313)
801    2323  format('    }')
802    write(1,2323)
803    write(1,*) ''
804    2334  format('    back')
805    write(1,2334)
806    2343  format('    {')
807    write(1,2343)
808    2353  format('        type            symmetryPlane;')
809    write(1,2353)
810    2363  format('    }')
811    write(1,2363)
812    write(1,*) ''
813    2373  format('    ground')
814    write(1,2373)
815    2383  format('    {')
816    write(1,2383)
817    2393  format('        type            epsilonWallFunction;')
818    write(1,2393)
819    2403  format('        Cmu            ',f6.3,';')
820    write(1,2403) cmu
821    2413  format('        kappa           ',f4.2,';')
822    write(1,2413) kv
823    2423  format('        E               9.8;')
824    write(1,2423)
825    2433  format('        value           uniform ',f8.3,';')
826    write(1,2433) epsw
827    2443  format('    }')
828    write(1,2443)
829    write(1,*) ''
830    2453  format('    sky')
831    write(1,2453)
832    2463  format('    {')
833    write(1,2463)
834    2473  format('        type            symmetryPlane;')
835    write(1,2473)
836    2483  format('    }')
837    write(1,2483)
```

```
838   2493  format('}')
839   write(1,2493)
840   write(1,*) ''
841   write(1,2153)
842
843   close(1)
844
845   call system('mv -f epsilon ../0')
846
847   !!$ Escrita de ficheiro "omega"
848
849   omegaint = epsint/(cmu*kint)
850   omegaw = epsw/(cmu*kw)
851
852   open(1,file='omega')
853
854   3003  format('*---------------------------------*- C++ -*----------------------------------*')
855   3013  format('| =========                 |                                                 |')
856   3023  format('| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |')
857   3033  format('|  \\    /   O peration     | Version:  2.0.0                                 |')
858   3043  format('|   \\  /    A nd           | Web:      www.OpenFOAM.com                      |')
859   3053  format('|    \\/     M anipulation  |                                                 |')
860   3063  format('\*-------------------------------------------------------------------------*/')
861   3073  format('FoamFile')
862   3083  format('{')
863   3093  format('    version     2.0;')
864   3103  format('    format      ascii;')
865   3113  format('    class       volScalarField;')
866   3123  format('    location    "0";')
867   3133  format('    object      omega;')
868   3143  format('}')
869   3153  format('// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //')
870
871   write(1,3003)
872   write(1,3013)
873   write(1,3023)
874   write(1,3033)
875   write(1,3043)
876   write(1,3053)
877   write(1,3063)
878   write(1,3073)
879   write(1,3083)
880   write(1,3093)
881   write(1,3103)
882   write(1,3113)
883   write(1,3123)
884   write(1,3133)
885   write(1,3143)
886   write(1,3153)
887   write(1,*) ''
888
889   3163  format('dimensions      [0 0 -1 0 0 0 0];')
890   write(1,3163)
891   write(1,*) ''
892   3173  format('internalField   uniform ',f6.3,';')
893   write(1,3173) omegaint
894   write(1,*) ''
895   3183  format('boundaryField')
896   write(1,3183)
897   3193  format('{')
898   write(1,3193)
899   3203  format('    inlet')
900   write(1,3203)
901   3213  format('    {')
902   write(1,3213)
903   3223  format('        type            fixedValue;')
904   write(1,3223)
905   3233  format('        value           nonuniform')
906   write(1,3233)
907   3234  format('        (#include "dat/omega.dat");')
908   write(1,3234)
909   3243  format('    }')
910   write(1,3243)
911   write(1,*) ''
912   3253  format('    outlet')
913   write(1,3253)
914   3214  format('    {')
915   write(1,3214)
916   3263  format('        type            zeroGradient;')
917   write(1,3263)
918   3283  format('    }')
919   write(1,3283)
920   write(1,*) ''
921   3293  format('    front')
922   write(1,3293)
```

```
923   3303  format('    {')
924   write(1,3303)
925   3313  format('        type            symmetryPlane;')
926   write(1,3313)
927   3323  format('    }')
928   write(1,3323)
929   write(1,*) ''
930   3334  format('    back')
931   write(1,3334)
932   3343  format('    {')
933   write(1,3343)
934   3353  format('        type            symmetryPlane;')
935   write(1,3353)
936   3363  format('    }')
937   write(1,3363)
938   write(1,*) ''
939   3373  format('    ground')
940   write(1,3373)
941   3383  format('    {')
942   write(1,3383)
943   3393  format('        type            omegaWallFunction;')
944   write(1,3393)
945   3403  format('        Cmu            ',f6.3,';')
946   write(1,3403) cmu
947   3413  format('        kappa           ',f4.2,';')
948   write(1,2413) kv
949   3423  format('        E               9.8;')
950   write(1,3423)
951   3424  format('        beta1           0.075;')
952   write(1,3424)
953   3433  format('        value           uniform ',f8.3,';')
954   write(1,3433) omegaw
955   3443  format('    }')
956   write(1,2443)
957   write(1,*) ''
958   3453  format('    sky')
959   write(1,3453)
960   3463  format('    {')
961   write(1,3463)
962   3473  format('        type            symmetryPlane;')
963   write(1,3473)
964   3483  format('    }')
965   write(1,3483)
966   3493  format('}')
967   write(1,3493)
968   write(1,*) ''
969   write(1,2153)
970
971   close(1)
972
973   call system('mv -f omega ../0')
974
975   10 format('(' 1pe11.4,1x,1pe11.4,1x,1pe11.4,')')
976
977   11 format(f10.4,1x,1pe11.4)
978
979   12 format(1pe11.4)
980
981   !!$ stop
982
983   13 format(' - Terminado em ',f6.2,' s')
984
985   call cpu_time(t2)
986
987   print*,'-'
988   write(*,13) t2-t1
989   print*,'-'
990   print*,'-----------------------------------------------'
991   print*,''
992
993   end program write_bCs_Cano
994   !!$################################
995   !!$################################
996   real function stringparser(strvar,wholefile,ierr,strfile)
997
998   implicit none
999
1000  !!$ Input Variables
1001  character (len=*) :: strvar,strfile
1002  character (len=70), dimension(100) :: wholefile
1003  integer :: ierr
1004
1005  !!$ Local Variables
1006  integer :: ilen,ibeg,iend
1007  integer :: i,j,k,l
```

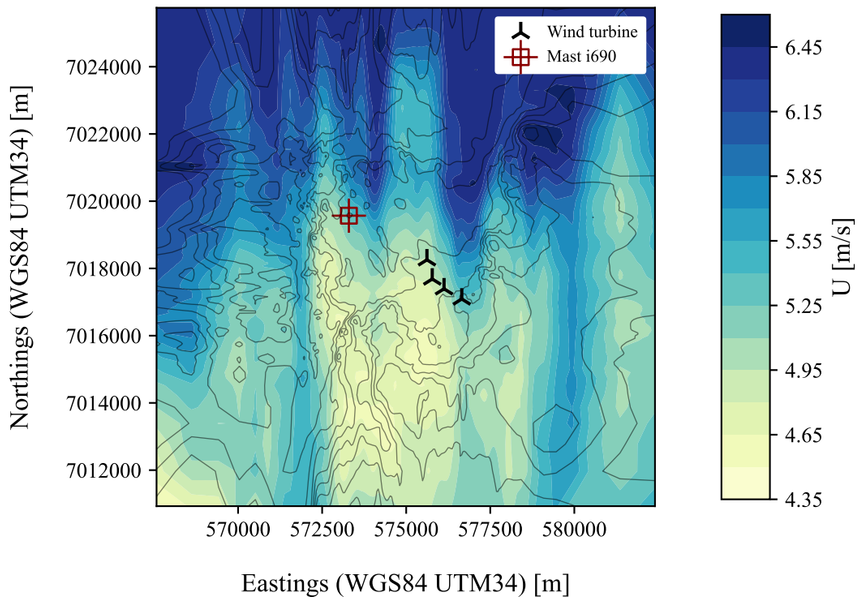
```
1008    integer :: iline,found,isdigit
1009    integer :: ierror
1010    character (len=15) :: extdigitset
1011    character (len=26) :: loweralphaset,upperalphaset
1012    character (len=70) :: line,trimline,snumber
1013    real :: r
1014
1015    !!$ .......... Init ..........
1016    ierr = 0
1017    ierror = 0
1018    extdigitset = '0123456789.+-eE'
1019    loweralphaset = 'abcdefghijklmnopqrstuvwxyz'
1020    upperalphaset = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
1021    found = 0
1022    ibeg = 0
1023    iend = 0
1024    ilen = len(strvar)
1025
1026    !!$ Initial search for string for error checking
1027    do i = 1, 100
1028
1029      line = wholefile(i)
1030      ibeg = index(line,strvar,.FALSE.)
1031      iend = index(line,strvar,.TRUE.)
1032    !     write(*,*) i,'-',strvar,'-','.',line,'.'
1033    !     write(*,*) ibeg,iend
1034
1035      if (ibeg.ne.0.and.iend.ne.0) then
1036          iend = iend + len(strvar) - 1
1037          if (ibeg.ne.1) then
1038            if (.not.(scan(line(ibeg-1:ibeg-1),extdigitset).gt.0.or. &
1039                scan(line(ibeg-1:ibeg-1),loweralphaset).gt.0.or. &
1040                scan(line(ibeg-1:ibeg-1),upperalphaset).gt.0.or. &
1041                scan(line(iend+1:iend+1),extdigitset).gt.0.or. &
1042                scan(line(iend+1:iend+1),loweralphaset).gt.0.or. &
1043                scan(line(iend+1:iend+1),upperalphaset).gt.0)) then
1044                found = found + 1
1045    !               write(*,*) '!!FOUND ',strvar,'at line',i,'chars',ibeg,'--',iend
1046            end if
1047          else
1048            if (.not.(scan(line(iend+1:iend+1),extdigitset).gt.0.or. &
1049                scan(line(iend+1:iend+1),loweralphaset).gt.0.or. &
1050                scan(line(iend+1:iend+1),upperalphaset).gt.0)) then
1051                found = found + 1
1052    !               write(*,*) '!!FOUND ',strvar,'at line',i,'chars',ibeg,'--',iend
1053            end if
1054          end if
1055      elseif (ibeg.ne.iend) then
1056          found = found + 2
1057    !         write(*,*) '!!FOUND ',strvar,'at line',i,'chars',ibeg, &
1058    !         ' -and',iend,'-'
1059      end if
1060    end do
1061
1062    if (found.eq.0) then
1063      write(*,*) 'ERROR! : ',strvar,' not found in ',strfile
1064      ierr = 1
1065    end if
1066
1067    !!$ Find line 'i' of WINDIE.CFG with parameter string
1068    ibeg = 0
1069    iend = 0
1070    i = 1
1071    do
1072      line = wholefile(i)
1073      ibeg = index(trim(line),strvar,.FALSE.)
1074      iend = index(trim(line),strvar,.TRUE.)
1075      if (ibeg.ne.0.or.iend.ne.0) then
1076          iend = iend + len(strvar) - 1
1077          if (ibeg.ne.1) then
1078            if (.not.(scan(line(ibeg-1:ibeg-1),extdigitset).gt.0.or. &
1079                scan(line(ibeg-1:ibeg-1),loweralphaset).gt.0.or. &
1080                scan(line(ibeg-1:ibeg-1),upperalphaset).gt.0.or. &
1081                scan(line(iend+1:iend+1),extdigitset).gt.0.or. &
1082                scan(line(iend+1:iend+1),loweralphaset).gt.0.or. &
1083                scan(line(iend+1:iend+1),upperalphaset).gt.0)) EXIT
1084          else
1085            if (.not.scan(line(iend+1:iend+1),extdigitset).gt.0.and. &
1086                .not.scan(line(iend+1:iend+1),loweralphaset).gt.0.and. &
1087                .not.scan(line(iend+1:iend+1),upperalphaset).gt.0) EXIT
1088          end if
1089      end if
1090      i = i + 1
1091    end do
1092
```

```
1093    !!$ Trim line to get a string 'strvar*'
1094    line = trim(adjustl(wholefile(i)))
1095    trimline = line(ibeg+len(strvar)+1:len(line))
1096
1097    !!$ Trim string to get number (in string format, no characters other than those in 'extdigitset')
1098    do ibeg = 1, len(trimline)
1099      isdigit = scan(trimline(ibeg:ibeg),extdigitset)
1100      if (isdigit.ne.0) EXIT
1101    end do
1102    do iend = ibeg, len(trimline)
1103      isdigit = scan(trimline(iend:iend),extdigitset)
1104      if (isdigit.eq.0) EXIT
1105    end do
1106    snumber = trimline(ibeg:iend)
1107
1108    !!$ Convert string to real number
1109    read(snumber,*) r
1110
1111    if (ierror.ne.0) then
1112      ierr = 2
1113      r = 0.0
1114      stringparser = r
1115    else
1116      stringparser = r
1117    endif
1118
1119    if (found.gt.1) then
1120      write(*,*) 'WARNING : more than one instance of ''',strvar, &
1121                 ''' in ',strfile,''
1122      write(*,*) 'Value of ''',strvar,''' taken to be ',r
1123    end if
1124
1125    return
1126
1127    end function stringparser
1128
1129    !!$################################
1130    !!$################################
1131    real function interp1D(n, y, z, y_interp)
1132
1133     implicit none
1134
1135     integer, intent(in) :: n         ! Size of input vectors
1136     real, intent(in) :: y(n), z(n)   ! Input vectors y and z
1137     real, intent(in) :: y_interp     ! Value of y for interpolation
1138
1139     integer :: i
1140     real :: slope, intercept
1141
1142     ! Perform linear interpolation
1143     do i = 1, n-1
1144         if (y_interp >= y(i) .and. y_interp <= y(i+1)) then
1145             slope = (z(i+1) - z(i)) / (y(i+1) - y(i))
1146             intercept = z(i) - slope * y(i)
1147             interp1D = slope * y_interp + intercept
1148             return
1149         endif
1150     end do
1151
1152     ! If y_interp is out of range
1153     write(*,*) 'y out of range, defaulting to z=0.0'
1154     interp1D = 0.0  ! or any other default value
1155
1156     return
1157
1158 end function interp1D
```

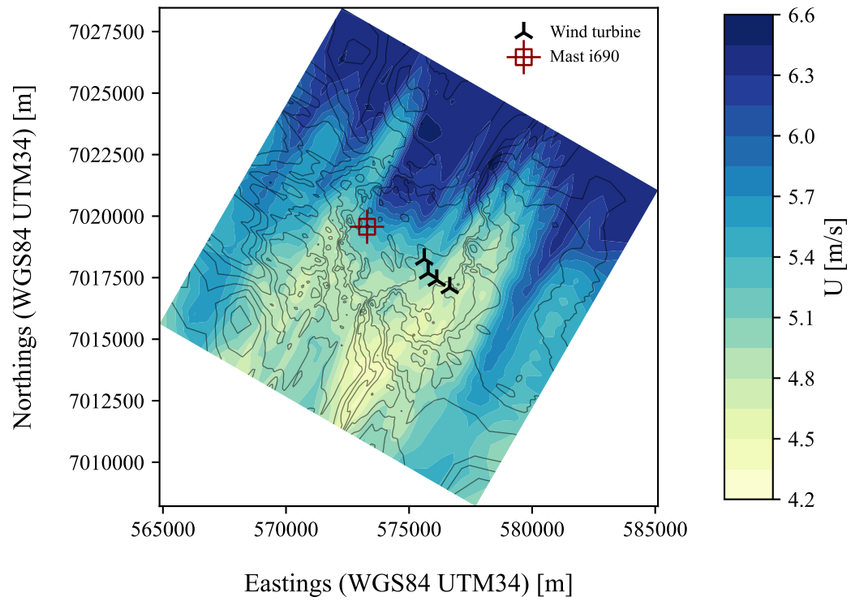
Appendix E

Velocity field obtained for SET_{214} at 166 m AGL (hub height)

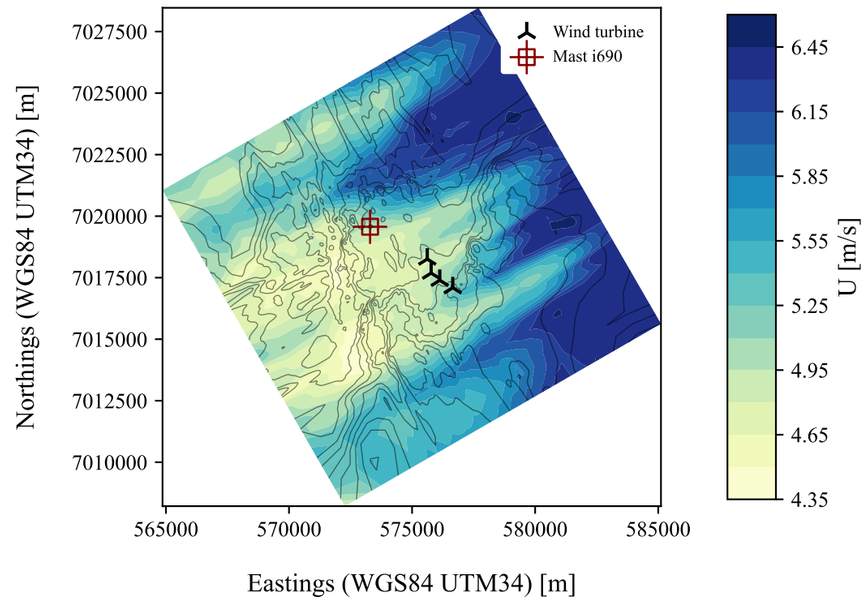
0° Wind



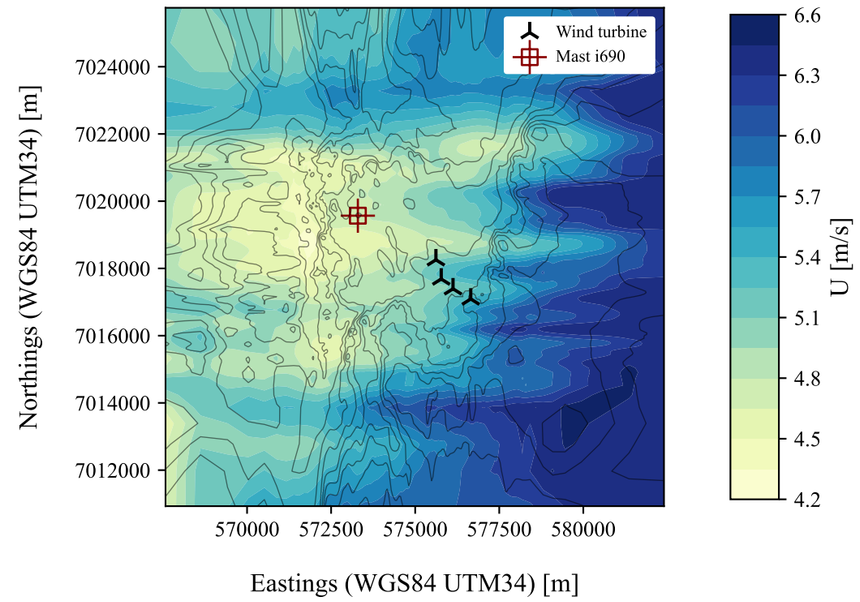
30° Wind



60° Wind



90° Wind



120° Wind

150° Wind

180° Wind

210° Wind

240° Wind

270° Wind

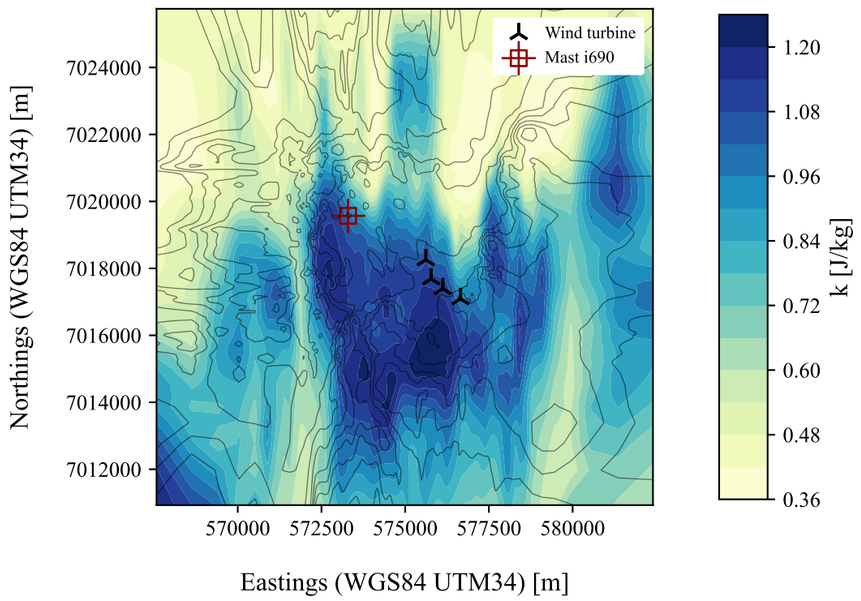
300° Wind

330° Wind

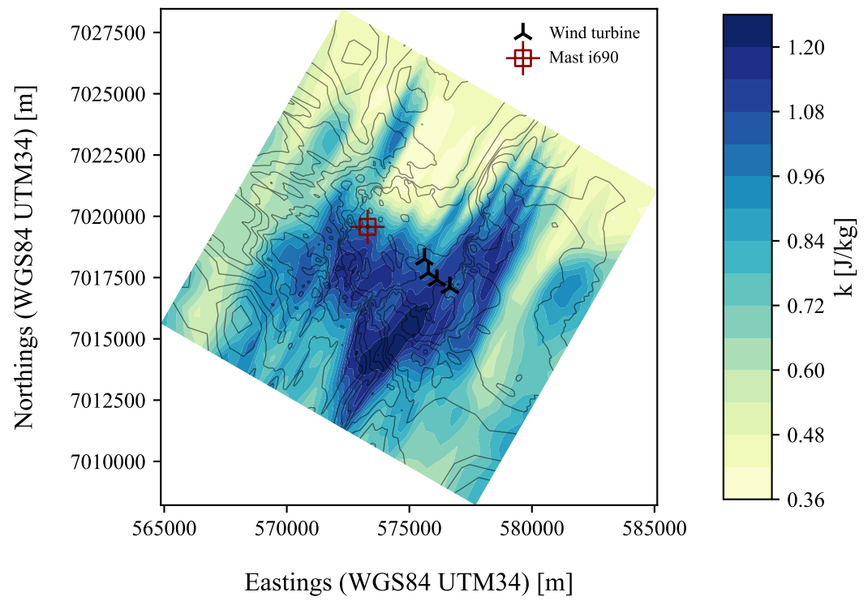
Appendix F

TKE field obtained for SET_{214} at 166 m AGL (hub height)

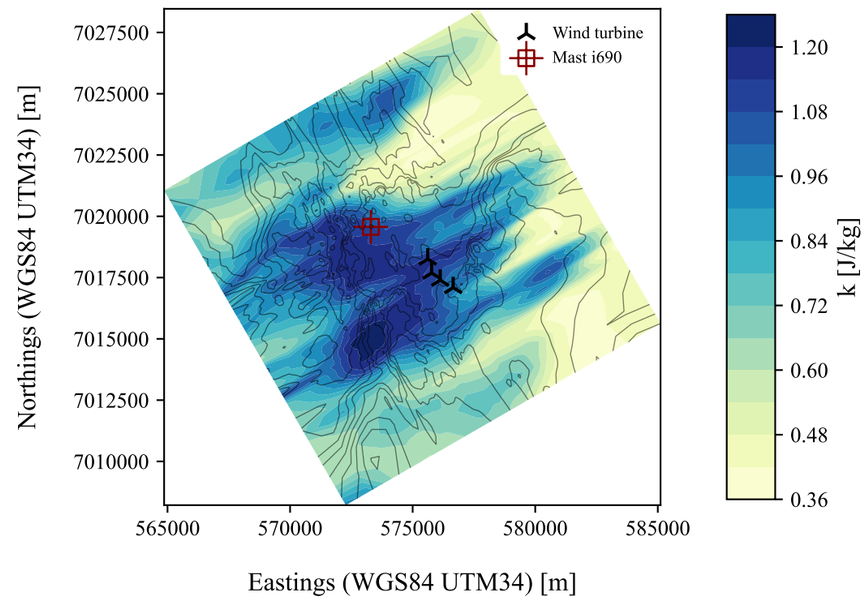
0° Wind



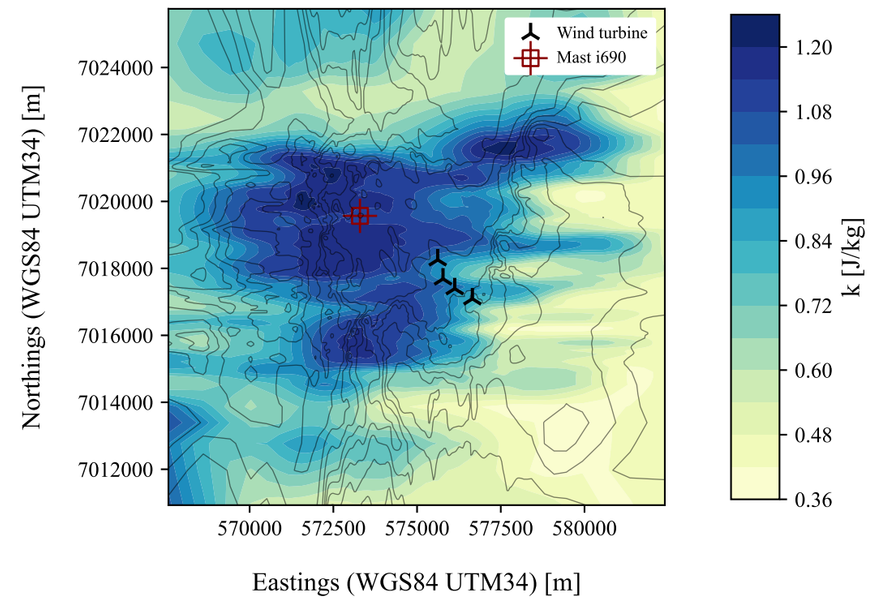
30° Wind



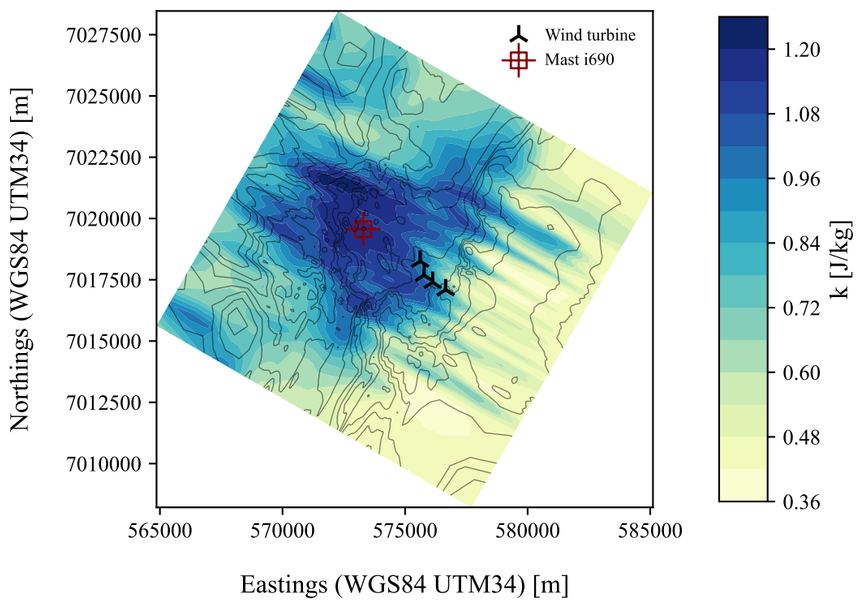
60° Wind



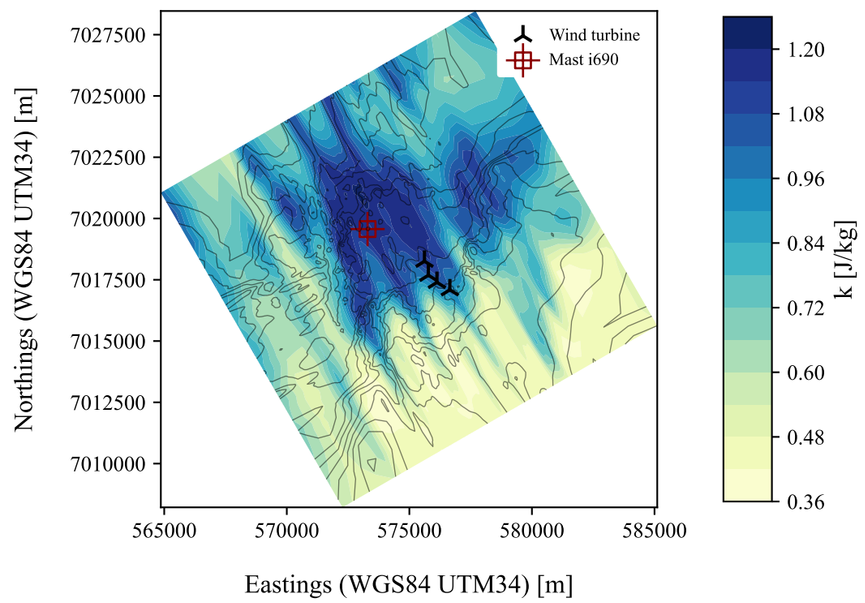
90° Wind



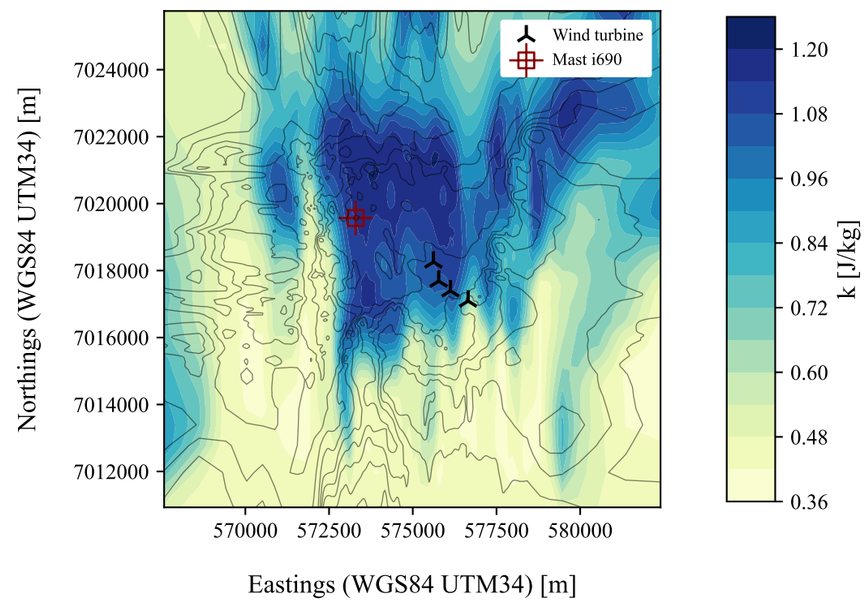
120° Wind



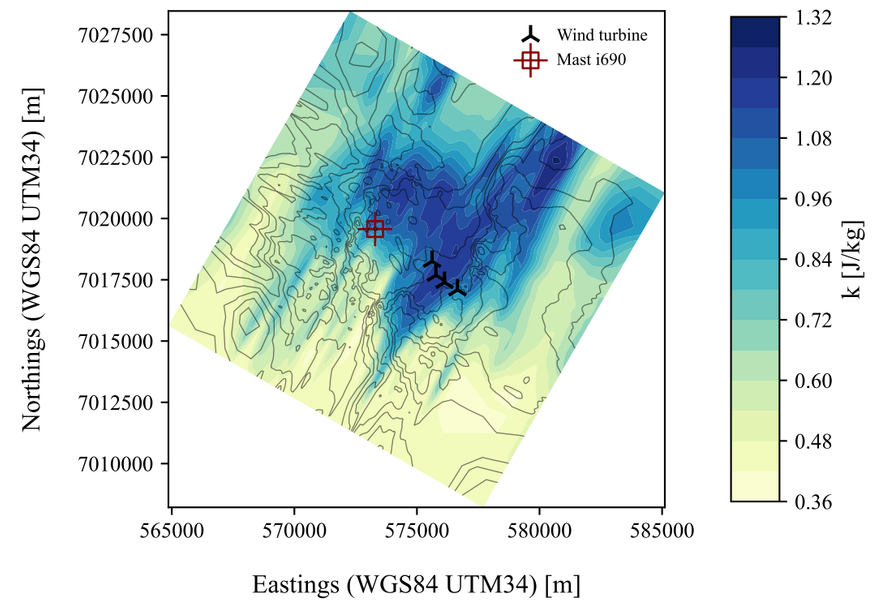
150° Wind



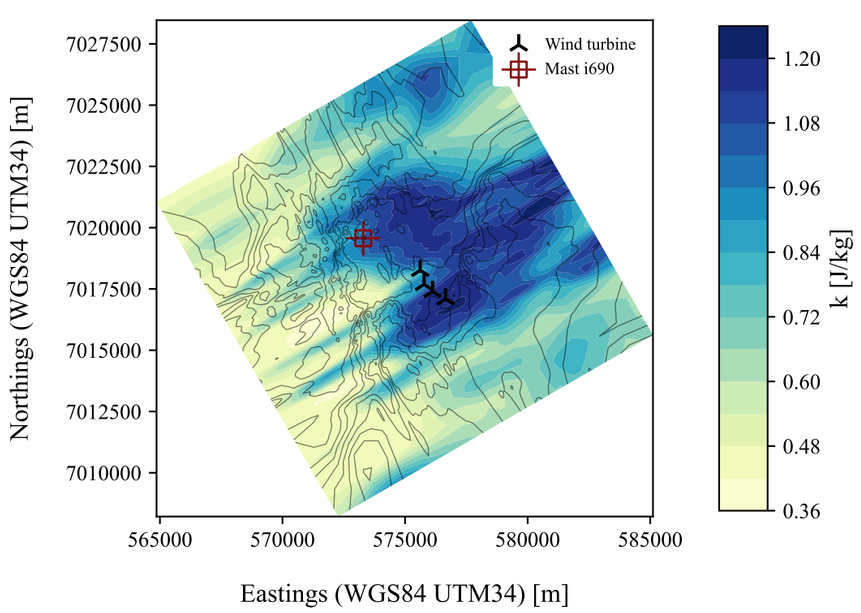
180° Wind



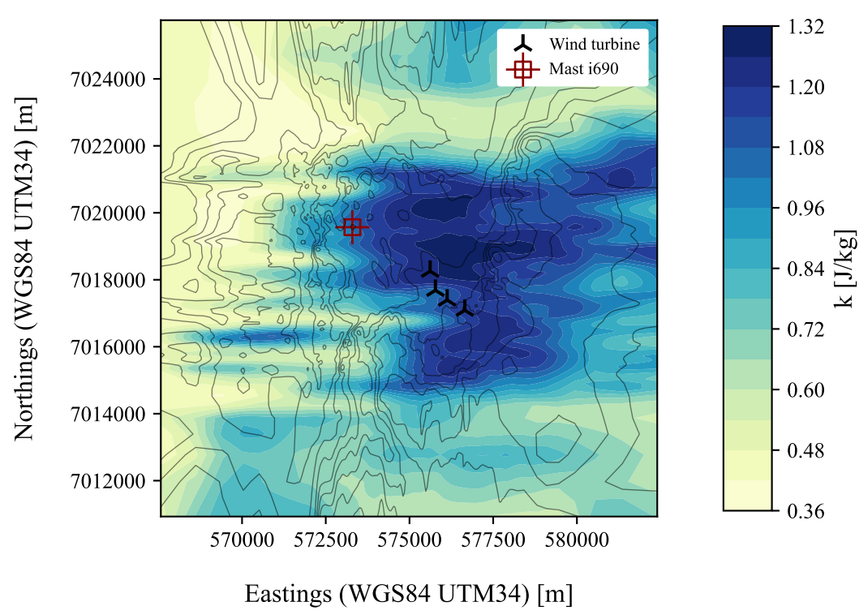
210° Wind



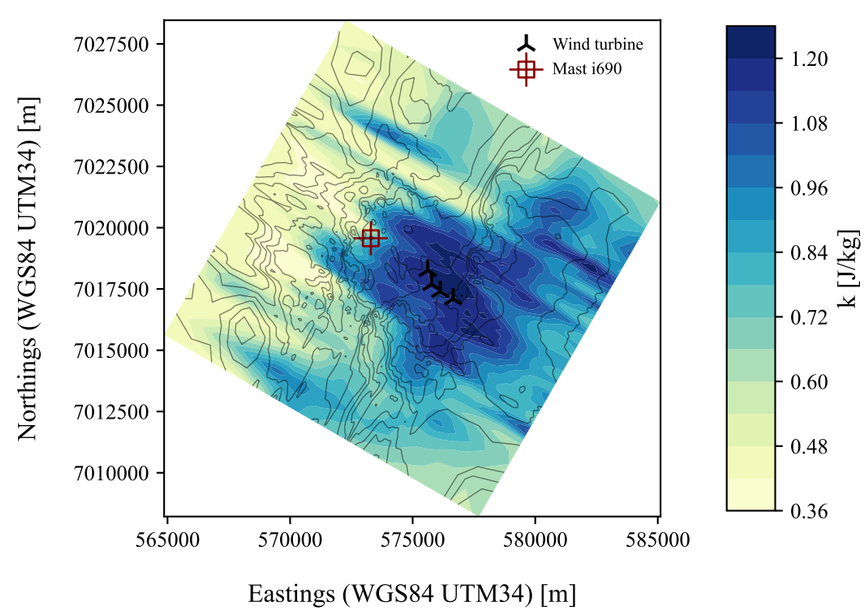
240° Wind



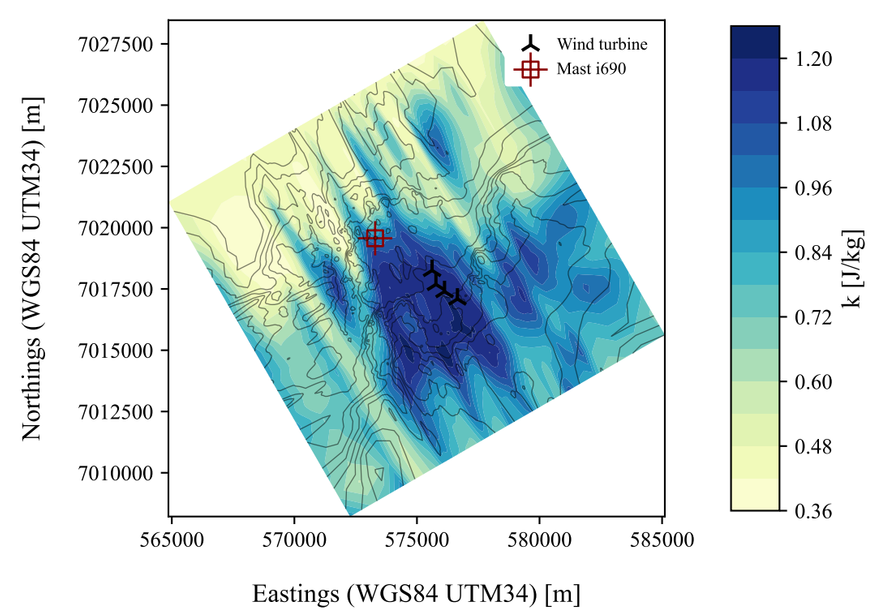
270° Wind



300° Wind



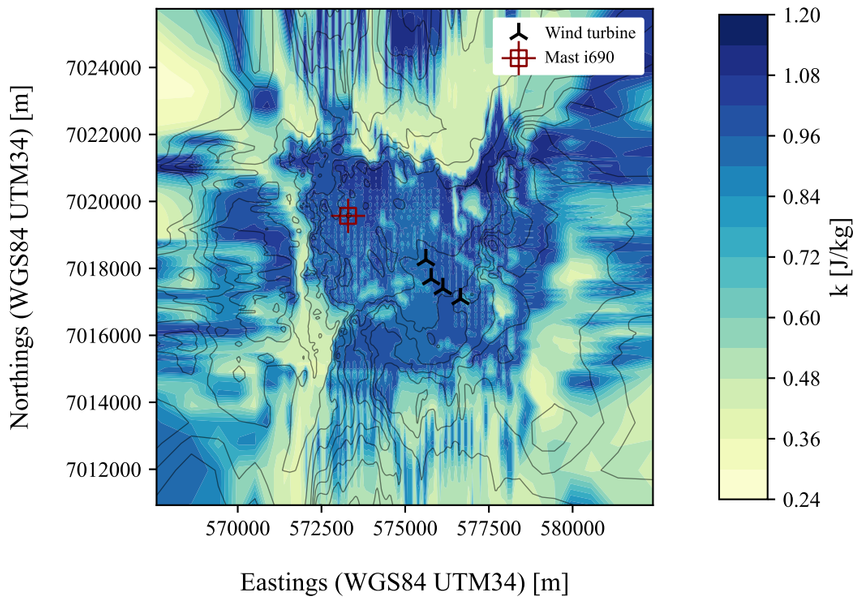
330° Wind



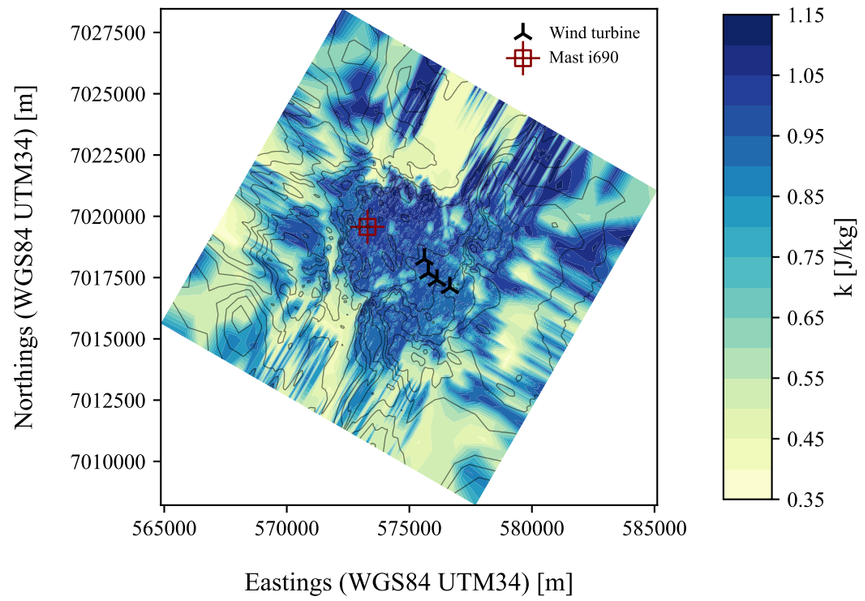
Appendix G

TKE field obtained for SET_{214} at 17.5 m AGL (canopy top height)

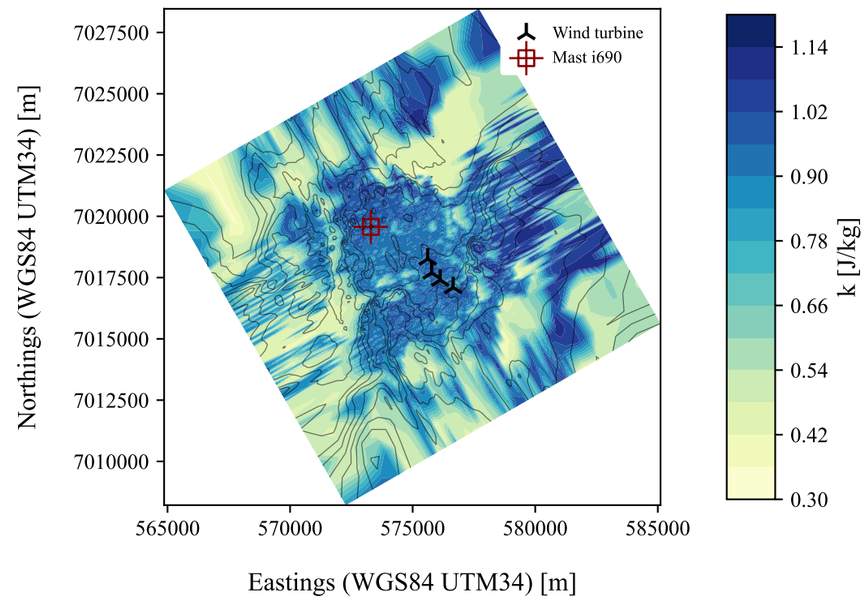
0° Wind



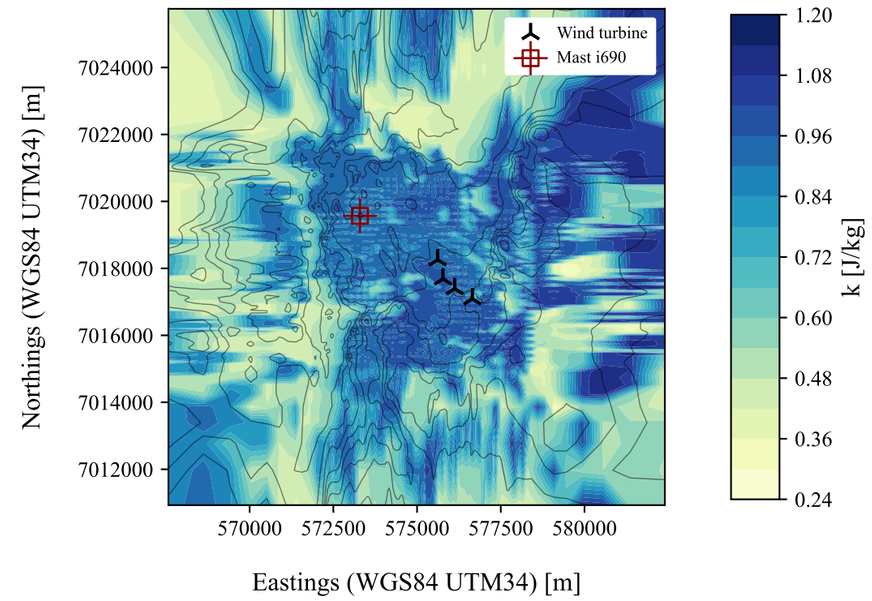
30° Wind



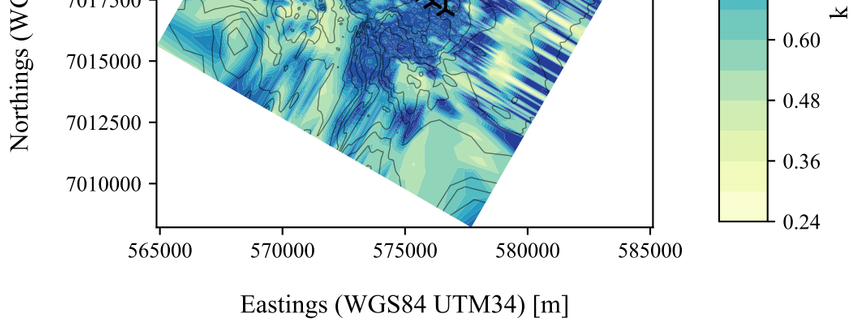
60° Wind



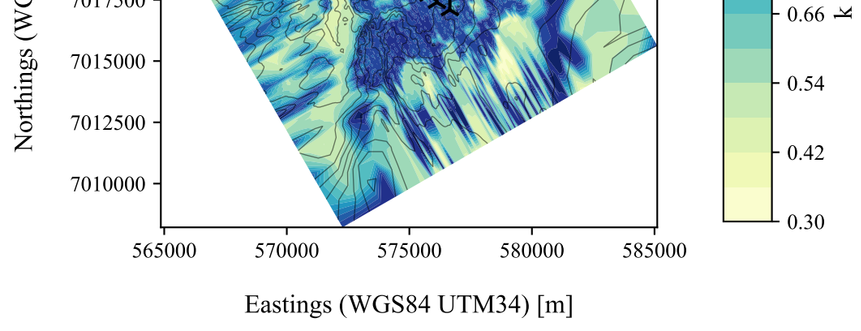
90° Wind



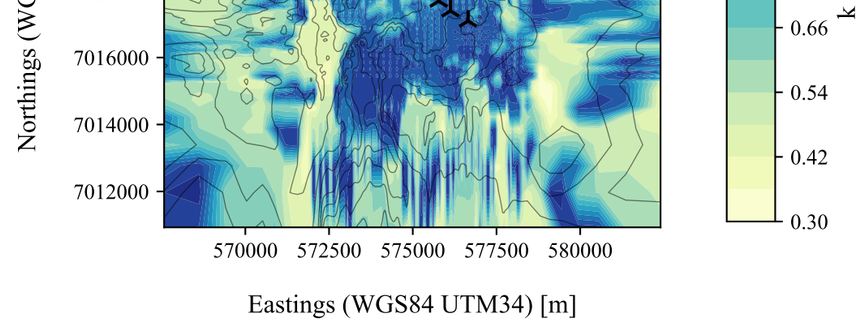
120° Wind



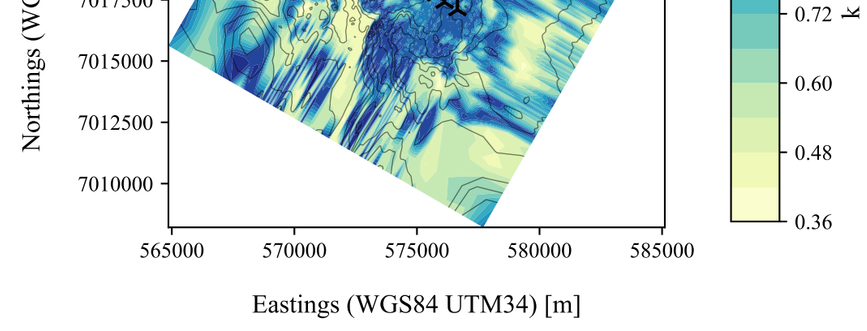
150° Wind



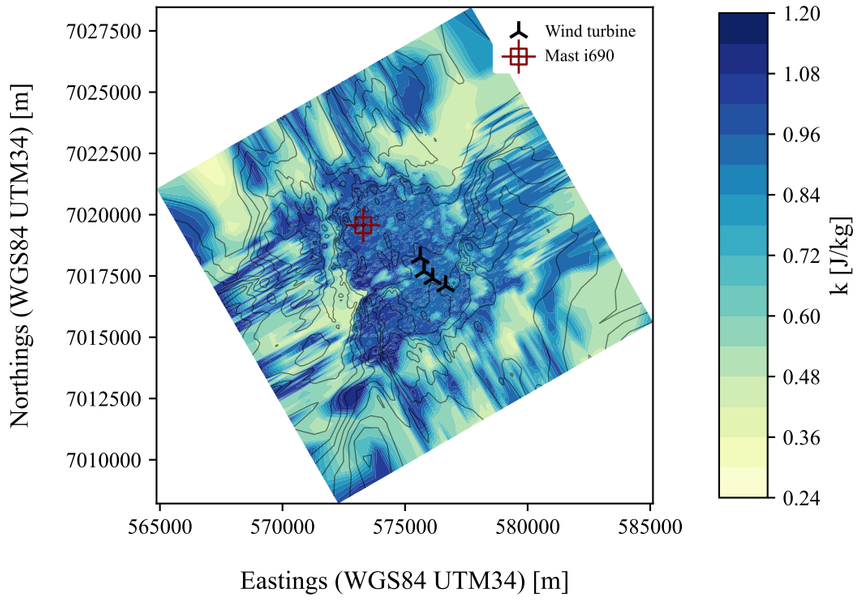
180° Wind



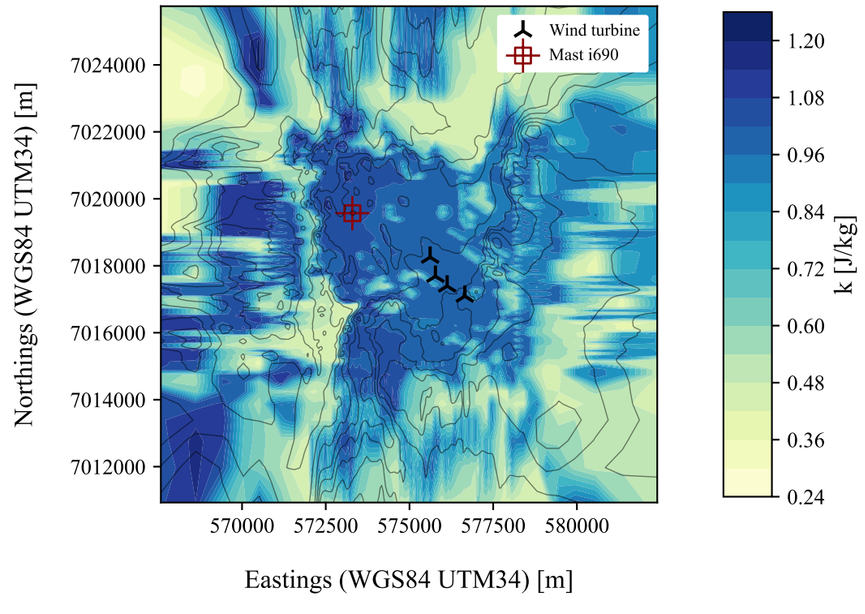
210° Wind



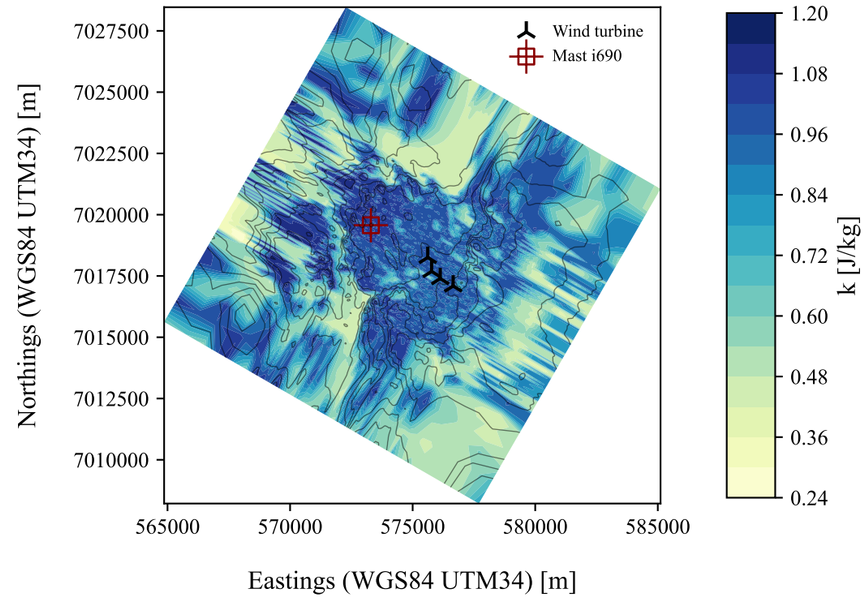
240° Wind



270° Wind



300° Wind



330° Wind