INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA MECÂNICA




ESTUDO DO RECURSO EÓLICO USANDO OPENFOAM COM CONDIÇÕES DE FRONTEIRA DE MODELOS DE MESOESCALA

HENRIQUE ALEXANDRE PINHEIRO OLIVEIRA MONTEIRO
outubro de 2023

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

Estudo do recurso eólico usando OpenFOAM com condições de fronteira de modelos de mesoescala

Henrique Alexandre Pinheiro Oliveira Monteiro

2023

Instituto Superior de Engenharia do Porto

Departamento de Engenharia Mecânica

P.PORTO

P.PORTO

Instituto Superior de Engenharia do Porto

Estudo do recurso eólico usando OpenFOAM com condições de fronteira de modelos de mesoescala

Henrique Alexandre Pinheiro Oliveira Monteiro

1170664

Dissertação apresentada ao Instituto Superior de Engenharia do Porto para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Mecânica, realizada sob a orientação do Prof. Doutor Carlos Silva Santos

2023

Instituto Superior de Engenharia do Porto

Departamento de Engenharia Mecânica

P.PORTO

P.PORTO

Agradecimentos

Aos meus pais, que tornaram este trabalho e o percurso até aqui possível, por todo o apoio que nunca deixaram faltar.

Ao Professor Doutor Carlos Silva Santos, pelo desafio que me propôs com o desenvolvimento deste trabalho, por ter partilhado comigo o seu vasto conhecimento e pelo seu contínuo apoio e disponibilidade.

Ao ISEP, por todo o conhecimento que me transmitiu e pelas pessoas que me apresentou.

A todos os que, direta ou indiretamente, contribuíram para o desenvolvimento deste trabalho, em especial ao Professor Doutor Fernando Aristides Castro, por uma valiosa sugestão que contribuiu para a progressão deste trabalho numa fase crucial do seu desenvolvimento.

Resumo

O principal objetivo desta dissertação é o desenvolvimento de um novo procedimento de acoplamento entre o modelo de mesoescala WRF (*Weather Research and Forecast*) e o *software* OpenFOAM (*Open Field Operation and Manipulation*), utilizado neste trabalho para efetuar simulações CFD (*Computational Fluid Dynamics*). O objetivo de acoplar modelos numéricos de mesoescala com modelos CFD de microescala surge da necessidade de obter condições de fronteira para o segundo que sejam o mais representativas da realidade possível.

Neste trabalho foram usadas duas abordagens, uma em que foram utilizadas condições de fronteira idealizadas e outra que faz uso de condições de fronteira provenientes de uma simulação precursora de mesoescala. As duas metodologias foram aplicadas num caso de estudo real onde foram efetuadas simulações de escoamentos atmosféricos neutralmente estratificados sobre terreno complexo em regime estacionário, mantendo sempre a avaliação do recurso eólico no local como um dos principais objetivos.

Para o estudo do recurso eólico, foram desenvolvidos nesta tese novos procedimentos que permitem o estudo detalhado das características do vento em localizações arbitrárias, como por exemplo, onde se pretende a instalação de aerogeradores.

Os resultados de cada uma das duas metodologias são comparados entre si e com dados medidos no local por campanhas de medições em períodos de tempo passados.

O acoplamento trouxe uma redução no erro RMS das validações da velocidade na horizontal em 5.9 % (de 8.1 % para 2.2 %) e em 0.8 % na vertical (de 2.4 % para 1.6 %) quando o transporte da série medida é feito usando os anemómetros mais altos do mastro. Estes resultados, ainda que as conclusões estejam circunscritas a um único caso teste, sugerem que a estratégia de acoplamento implementada pode trazer uma melhoria significativa na qualidade dos resultados à altura de eixo das turbinas eólicas.

Palavras-Chave

Escoamentos Atmosféricos; Mecânica de Fluidos Computacional; OpenFOAM; Acoplamento Microescala-Mesoescala; Avaliação do Recurso Eólico.

Abstract

The main purpose of this thesis is to develop a new coupling procedure between the WRF (Weather Research and Forecast) mesoscale model and OpenFOAM (Open Field Operation and Manipulation), which is used in this work to perform CFD (Computational Fluid Dynamics) simulations. The aim of coupling mesoscale numerical models with microscale CFD models arises from the need to obtain boundary conditions for the latter that are as representative of reality as possible.

Two approaches were employed in this work, one using idealised boundary conditions and the other using boundary conditions from a precursor mesoscale simulation. The two methodologies were applied to a real case study in which steady state simulations of neutrally stratified atmospheric flows over complex terrain were performed, with a primary focus on wind resource assessment at the site.

In order to study the wind resource, new procedures have been developed in this thesis that allow the detailed study of wind characteristics in arbitrary locations, such as those where wind turbines are intended to be installed.

The results obtained from each of the two methodologies are compared with each other and with data collected on-site during previous measurement campaigns.

The coupling strategy led to a 5.9 % reduction in horizontal velocity validation RMS errors (from 8.1 % to 2.2 %) and a 0.8 % reduction in vertical RMS errors (from 2.4 % to 1.6 %) when transporting the measured data using higher mast-mounted anemometers. These results, although the conclusions are limited to a single test case, suggest that the implemented coupling strategy has the potential to significantly enhance the quality of results at the wind turbine's hub height.

Keywords

Atmospheric Flows; Computational Fluid Dynamics; OpenFOAM;
Microscale-Mesoscale Coupling, Wind Resource Assessment

Índice

Índice de Figuras

Índice de Tabelas

Nomenclatura

Lista de siglas e abreviaturas

ABL	*Atmospheric Boundary Layer*
ANS	Acima do nível do solo
ASCII	*American Standard Code for Information Interchange*
CBL	*Convective Boundary Layer*
CFD	*Computational Fluid Dynamics*
DNS	*Direct Numerical Simulation*
LES	*Large Eddy Simulation*
LT-ERA5	Série temporal de longo termo provinda da base de dados ERA5
LT-MESO	Dados de longo termo provindos de uma simulação de meso-escala
MCP	*Measure-Correlate-Predict*
NBL	*Nocturnal Boundary Layer*
netCDF	*Network Common Data Form*
NWP	*Numerical Weather Prediction*
OpenFOAM	*Open Field Operation and Manipulation*
pbs	*Portable Batch Script*
RMS	*Root Mean Square*
RaNS	*Reynolds Averaged Navier-Stokes*
SC	Superfície de Controlo
SIST	Sistema
TTR	Teorema de Transporte de Reynolds
TUI	*Text User Interface*
UTM	*Universal Transverse Mercator*
VC	Volume de Controlo
VLES	*Very Large Eddy Simulation*
VTK	*Visualization Toolkit*
WAsP	*Wind Atlas and Application Program*
WRF	*Weather Research and Forecast*

Lista de símbolos

$\mathcal{A}$	Área
B	Quantidade extensiva
c	Fator de escala
c_p	Calor específico a pressão constante
C_μ, σ_k, σ_ε, C_1, C_2	Constantes adimensionais empíricas do modelo
F	Fator de correção
f	Distribuição de Weibull
F_{ext}	Forças externas
g	Aceleração da gravidade
$\vec{g}$	Vetor da aceleração da gravidade
k	Fator de forma adimensional (em 2.1.4 e 3.1.2), energia cinética da turbulência (restante documento)
m	Massa
$\vec{n}$	Vetor unitário
p	Pressão
p_0	Pressão de referência
P_k	Produção mecânica de turbulência
p_s	Pressão na superfície do domínio
p_t	Pressão no topo do domínio
$\dot{Q}_i$	Fluxo na fronteira i
$\dot{Q}_{\max}$	Fluxo máximo
R	Constante específica do gás
Re	Número de Reynolds
t	Tempo
T	Temperatura
T	Período
u	Velocidade
u, v, w	Componentes x, y e z de velocidade
u', v', w'	Componentes x, y e z de velocidade instantânea
$\bar{u}$	Velocidade média
u_*	Velocidade de fricção
u_i, v_i, w_i	Componentes do vetor velocidade na fronteira i
$u_{\mathrm{i,corr}}$, $v_{\mathrm{i,corr}}$, $w_{\mathrm{i,corr}}$	Componentes do vetor velocidade corrigidos na fronteira i
$\mathcal{V}$	Volume
$\vec{V}$	Vetor velocidade
v_m	Velocidade média
x, y, z	Coordenadas cartesianas
y^+	Distância adimensional à parede
z_0	Rugosidade superficial ou rugosidade aerodinâmica
z_p	Distância entre a parede e o centro geométrico do volume de controlo adjacente

Lista de símbolos (Continuação)

β	Quantidade intensiva
Γ	Função *gamma* (em 3.1.2), gradiente térmico adiabático (restante documento)
Γ_d	Gradiente térmico adiabático seco
$\Delta\phi$	Diferença entre a direção do vento na turbina e a direção de vento no mastro
δ	Altura da camada limite
δ_{ij}	Delta de Kronecker
ε	Taxa de dissipação da energia cinética da turbulência
η	Níveis de pressão constante
θ	Temperatura potencial
θ_v	Temperatura potencial virtual
κ	Constante de *von Kármán*
μ	Viscosidade dinâmica
μ_t	Viscosidade dinâmica turbulenta
ν_t	Viscosidade cinemática turbulenta
ν_{t_w}	Viscosidade cinemática turbulenta em região próxima à parede
ν_w	Viscosidade cinemática do fluido em região próxima à parede
ρ	Massa volúmica
σ	Desvio padrão
τ	Tensão de corte
$\vec{\tau}_{ij}$	Tensões viscosas
ω	Frequência de dissipação da energia cinética da turbulência
ω_v	Humidade específica

1 Introdução

1.1 Contexto

O crescimento do setor da energia eólica e o elevado investimento inerente à instalação de um parque eólico têm motivado o desenvolvimento de procedimentos que permitam a correta avaliação do recurso eólico, a médio e longo prazo, dos locais onde se pretende a instalação de aerogeradores, assim como selecionar a melhor localização para as turbinas eólicas no domínio em estudo. Para estes efeitos, existem modelos tradicionalmente implementados na indústria eólica, e amplamente usados, como por exemplo o modelo WAsP (*Wind Altlas and Application Program*) [1, 2].

Os modelos lineares, como é o caso do WAsP, são de fácil utilização mas têm limitações, nomeadamente para terreno complexo. Por outro lado, os modelos não lineares de CFD (*Computational Fluid Dynamics*) têm, nos últimos anos, vindo a complementar ou mesmo substituir os tradicionais modelos lineares, mesmo para orografias de complexidade moderada ou baixa [2].

O desenvolvimento de modelos CFD nesta área de estudo está relacionado com a necessidade de instalar parques eólicos em locais onde a topografia é complexa e o escoamento é significativamente afetado pelo terreno, o que resulta num aumento significativo de complexidade do escoamento, provocado por declives acentuados da orografia e cobertura florestal significativa. Nestas condições, os modelos lineares têm limitada aplicabilidade e produzem resultados menos precisos relativamente aos modelos CFD que, para além de melhores resultados, também indicam aspetos importantes sobre o escoamento nestas condições que os modelos lineares não conseguem prever, como zonas de recirculação e a componente vertical do vetor velocidade [3].

1.2 Breve Descrição do Problema

Uma das fases importantes na construção do modelo CFD é a adequada especificação das condições de fronteira nos limites do domínio computacional em estudo. Em simulações de escoamentos atmosféricos sobre topografia complexa, especificar condições de fronteira que representem a realidade pode ser complexo. Uma estratégia comum é a utilização de perfis idealizados de velocidade e quantidades de turbulência na fronteira de entrada e a configuração das simulações como se de um túnel de vento numérico se tratasse.

Outra estratégia consiste na utilização de dados de modelos de mesoscala, como é o caso do modelo WRF (*Weather Research and Forecast*) [4]. Incorporar esses resultados como condições de fronteira no modelo CFD de microescala é um processo ao qual se dá o nome de acoplamento. É esta estratégia que este trabalho se propõe implementar e testar, usando o modelo CFD OpenFOAM [5].

1.3 Estrutura da Tese

No capítulo 2 é efetuada uma revisão bibliográfica sobre os temas considerados relevantes como base teórica para o trabalho desenvolvido nesta dissertação.

No capítulo 3 foram desenvolvidas simulações CFD de escoamentos atmosféricos neutralmente estratificados com condições de fronteira idealizadas, onde se adotaram as técnicas desenvolvidas em [6] para o pré-processamento e pós-processamento do modelo numérico. Para a realização das simulações, foi implementado *batch processing* e *cloud computing*. Para o estudo do recurso eólico e resultados do modelo numérico, foram desenvolvidas técnicas de assimilação de medições em pós-processamento.

No capítulo 4 foram desenvolvidas simulações CFD de escoamentos atmosféricos neutralmente estratificados com acoplamento, onde se apresentam as técnicas desenvolvidas neste trabalho para implementar condições de fronteira provenientes de modelos de mesoescala. Neste capítulo serão também usadas as técnicas do capítulo 3 para computação de alta *performance* e estudo do recurso eólico.

No capítulo 5 são apresentadas as conclusões finais e o trabalho futuro que se propõe como continuidade ao trabalho desenvolvido nesta dissertação.

Na presente dissertação, optou-se por utilizar o ponto (.) como separador decimal, por questões de coerência com os gráficos produzidos com recurso a linguagens de programação ou imagens obtidas com *software* de visualização de resultados.

2 Revisão bibliográfica

2.1 Camada limite atmosférica

2.1.1 Estrutura

A camada limite atmosférica representa a parte da troposfera que é diretamente influenciada pela superfície terrestre e a sua altura está tipicamente definida num intervalo de 100 a 3000 metros acima do solo [7]. Os escoamentos de vento que nela existem têm origem em fenómenos atmosféricos resultantes de gradientes de pressão do ar em larga escala e na força aparente de Coriolis, associada à rotação da Terra. Ao balanço destes dois fenómenos é atribuído o termo de condições geostróficas e o vento nestas condições é definido como vento geostrófico, condição válida para alturas acima do solo elevadas [8].

Na camada limite atmosférica, o vento não pode ser classificado como geostrófico porque o escoamento é influenciado por fatores como a orografia do terreno, transferência de calor, evaporação e emissão de poluentes, o que também provoca uma variação espacial e temporal da altura da ABL (*Atmospheric Boundary Layer*). Durante o ciclo diurno, a ABL pode ser classificada como CBL (*Convective Boundary Layer*) e NBL (*Nocturnal Boundary Layer*) [7, 9].

Durante o dia, a radiação solar aquece o ar junto ao solo, o que leva à formação de correntes convectivas resultantes da diminuição da massa volúmica do ar perto do terreno e que provocam movimentos verticais do ar (figura 2.1). Por contraste, à noite, a massa volúmica do ar junto ao solo vai aumentado com a descida da temperatura do ar e desta forma já não se verifica a ocorrência significativa de correntes de convecção (figura 2.2) [10].

A velocidade do vento é tipicamente aproximada por um perfil logarítmico que decresce junto ao solo devido à tensão de corte induzida pela superfície terrestre. Na CBL, a velocidade do vento apresenta pouca variação quando em alturas acima da

camada superficial e toma sempre valores de velocidade abaixo do valor correspondente ao vento geostrófico (ver figura 2.1). Na NBL, o vento pode tomar valores acima do correspondente ao vento geostrófico, num fenómeno denominado de jato noturno e que potencialmente irá gerar turbulência (ver figura 2.2) [6, 11].

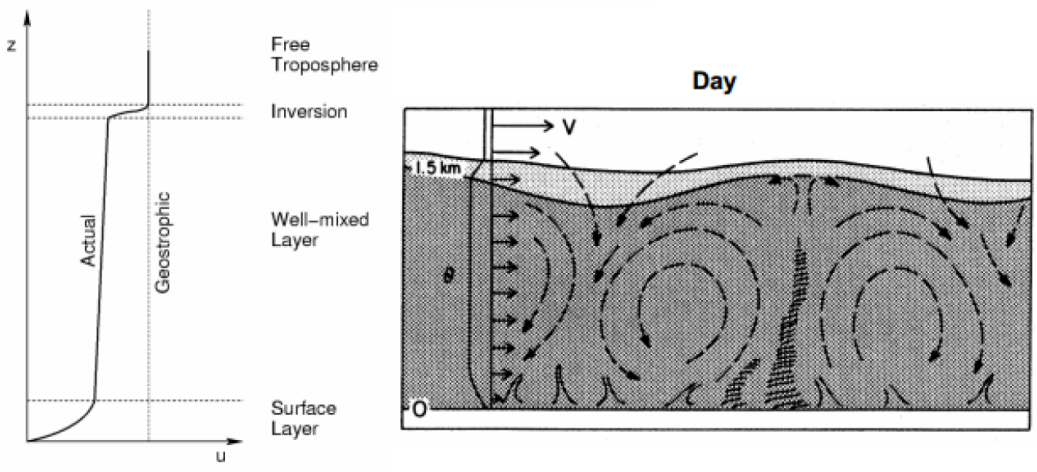


Figura 2.1 – CBL e perfis típicos de escoamento (retirado de [10])

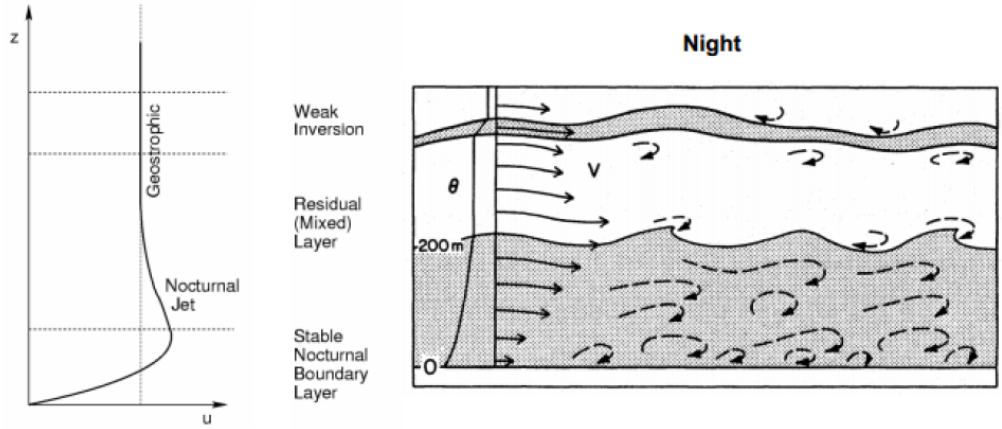


Figura 2.2 – NBL e perfis típicos de escoamento (retirado de [10])

2.1.2 Estabilidade atmosférica

O estabelecimento de gradientes de temperatura que provocam o desenvolvimento de correntes de convecção é uma das principais causas da turbulência na camada limite atmosférica.

Num regime atmosférico neutro, a temperatura T diminui gradualmente em altura a uma taxa vulgarmente designada por gradiente térmico adiabático Γ, o qual, considerando ar seco, pode ser definido por ([12]):

$$\Gamma_d = -\frac{\partial T}{\partial z} = \frac{g}{c_p} \approx 10\,^{\circ}\mathrm{C/km} \ . \tag{2.1}$$

A atmosfera deixa de ser neutra quando existe uma diferença entre a temperatura do solo e a temperatura do ar na envolvente, o que provoca um desvio do perfil de temperatura em relação ao seu gradiente térmico adiabático [3]. Na situação em que uma parcela de ar está mais quente que o ar da sua envolvente, esta sofre uma força de impulsão ascendente, como consequência da diminuição da sua massa volúmica [6]. Este processo gera as correntes de convecção representadas na figura 2.1.

Com o objetivo de comparar várias massas de ar a diferentes níveis de pressão, é comum a definição de um parâmetro θ, à qual se dá o nome de temperatura potencial [11, 9],

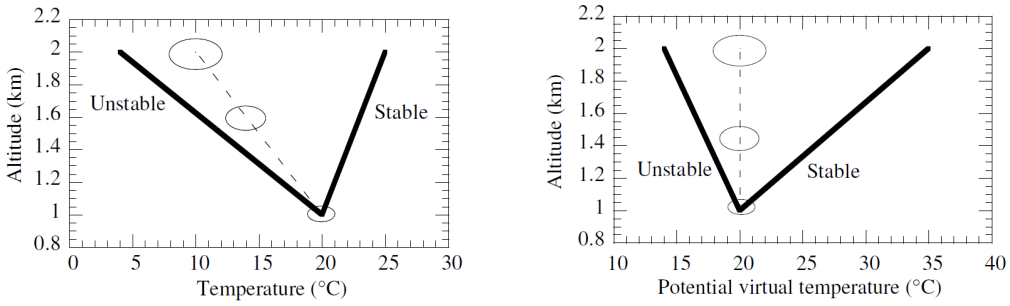
$$\theta = T\left(\frac{p_0}{p}\right)^{R/c_p} \quad , \tag{2.2}$$

onde p representa a pressão, p_0 uma pressão de referência, R a constante específica do gás e c_p o calor específico a pressão constante.

Em alguns casos, também é comum introduzir o parâmetro θ_v, que representa a temperatura potencial virtual, onde também é considerada a quantidade de vapor de água presente no ar através da humidade específica ω_v. A temperatura potencial de uma parcela de ar representa a temperatura que ela teria se fosse transportada adiabaticamente para uma pressão de 100 kPa [11, 9].

$$\theta_v = T\left(1 + 0.608\omega_\mathrm{v}\right)\left(\frac{p_0}{p}\right)^{R/c_p} \quad . \tag{2.3}$$

A estabilidade atmosférica pode ser avaliada com base no seu gradiente térmico Γ (figura 2.3a) ou na variação da temperatura potencial virtual θ_v (figura 2.3b).



(a) (b)

Figura 2.3 – Variação do gradiente térmico adiabático (a) e da temperatura potencial virtual (b) em função da altura para diferentes regimes de estabilidade atmosférica (adaptado de [9])

Com base no gradiente térmico de uma atmosfera Γ e na temperatura potencial virtual θ_v pode-se definir três regimes de estabilidade atmosférica [3, 11]:

- **Regime Neutro**, quando $\Gamma = \Gamma_d$ e $\partial\theta_v/\partial z = 0$, nestas condições não existem trocas de calor da superfície terrestre com o ar da envolvente, o que resulta

numa taxa de variação da temperatura em altura igual ao gradiente térmico adiabático. Esta situação ocorre quando o solo está à mesma temperatura que o ar da envolvente.

- **Regime Estável**, quando $\Gamma < \Gamma_d$ e $\partial\theta_v/\partial z > 0$, nesta situação existe uma taxa de variação de temperatura em altura inferior ao gradiente térmico adiabático, condição em que o solo está mais frio que o ar da envolvente e a tendência é haver correntes convectivas verticais que suprimem turbulência.

- **Regime Instável**, quando $\Gamma > \Gamma_d$ e $\partial\theta_v/\partial z < 0$, nesta situação existe uma taxa de variação de temperatura em altura superior ao gradiente térmico adiabático. Desta forma, a tendência é a massa de ar junto ao solo subir e afastar-se da posição inicial, resultando da temperatura do solo ser superior à temperatura do ar da envolvente e gerar correntes convectivas e turbulência de origem térmica.

2.1.3 Turbulência na camada limite atmosférica

Para além das correntes de convecção resultantes da impulsão provocada por gradientes térmicos, a outra grande causa da turbulência na camada limite atmosférica é a tensão de corte induzida pela superfície terrestre e presença de obstáculos, como por exemplo floresta.

Como já referido anteriormente na secção 2.1.1, na camada limite atmosférica, o perfil de velocidade do vento pode ser aproximado por uma função logarítmica empírica, a qual, para escoamentos neutralmente estratificados (onde não há presença significativa da estratificação térmica referida na secção 2.1.2), pode ser aproximada por ([13]);

$$u(z) = \frac{u_*}{\kappa} \ln\left(\frac{z}{z_0}\right) \ , \tag{2.4}$$

em que κ é a constante de *von Kármán* e u_* é a velocidade de fricção que relaciona a tensão de corte τ com a massa volúmica ρ através de $u_*^2 = \tau/\rho$.

O parâmetro z_0, designado por rugosidade superficial ou rugosidade aerodinâmica, assume diferentes valores caracterizados pela forma e densidade dos elementos da superfície [11]. A rugosidade superficial representa a altura acima da superfície à qual a função logarítmica que caracteriza a velocidade do vento é zero (ver figura 2.4) [9].

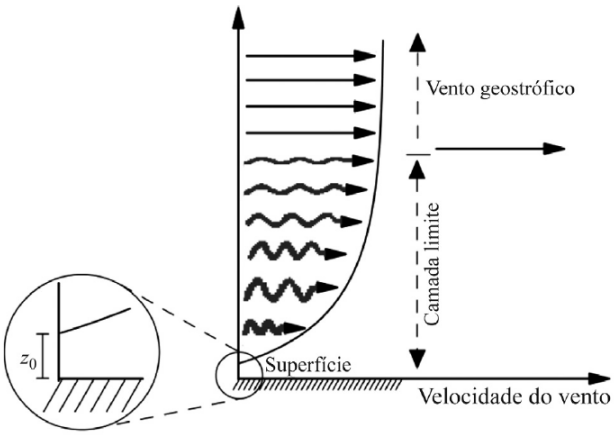


Figura 2.4 – Perfil típico de velocidade do vento considerando o parâmetro z_0 (retirado de [14])

A definição de valores *standard* de rugosidade superficial para diferentes situações é ainda um tema que pode ter diferentes interpretações, e por isso, os valores de z_0 podem variar com o autor em questão. Na tabela 2.1 estão indicados alguns valores típicos.

Tabela 2.1 – Valores típicos de rugosidade z_0 para diferentes tipos de superfície (Adaptado de [8] e [11])

Superfície	z_0 [m]
Água	0.1 a 10.0 $\times 10^{-5}$
Gelo e neve	0.1 a 10.0 $\times 10^{-4}$
Areia e deserto	0.0003
Relvados	0.003 a 0.1
Culturas agrícolas	0.04 a 0.2
Zonas Urbanas	0.5 a 1
Floresta	1 a 6

Usualmente, um mapa de rugosidade com valores de z_0 definidos em todos os pontos do domínio é um dos *inputs* em modelos lineares e não-lineares.

2.1.4 Medições de vento

As medições de vento locais consistem geralmente em séries temporais de velocidade e direção. Para descrever a variação de velocidade do vento é comum a utilização de uma função de densidade de probabilidade, a distribuição de Weibull, por ser das que melhor se adequa aos padrões da velocidade do vento. A distribuição de Weibull f, é dada por ([15, 3]):

$$f(u) = \left(\frac{k}{c}\right)\left(\frac{u}{c}\right)^{k-1} e^{-(u/c)^k} \quad , \tag{2.5}$$

em que k é o fator de forma adimensional e c é o fator de escala. A velocidade média

$\bar{u}$, é calculada com:

$$\bar{u} = \int_0^\infty u f(u) du \ . \tag{2.6}$$

Existem vários métodos para calcular os parâmetros k e c e assim definir uma curva que se ajuste aos dados de vento locais. Alguns desses métodos são aqui listados [16]:

- **Método gráfico** - Neste método, os dados são representados graficamente e transformados de forma a obter uma distribuição linear ou aproximadamente linear, sendo o declive da reta formada o fator de forma k;
- **Método de regressão dos mínimos quadrados** - Nesta abordagem os dados são ajustados a uma curva teórica com o objetivo da minimização da soma dos quadrados das diferenças entre os valores observados e os valores previstos por essa curva;
- **Método da máxima verosimilhança** - Consiste em determinar os valores dos parâmetros que maximizam a verosimilhança dos dados observados, *i.e.*, os valores que tornam mais provável a ocorrência dos dados observados de acordo com a distribuição de Weibull;
- **Método dos momentos** - O método dos momentos é uma técnica relativamente simples baseada no conceito da igualdade entre os momentos teóricos da distribuição e os momentos da amostra de dados, como a média e o desvio padrão.

2.2 Modelação numérica na energia eólica

2.2.1 Modelos lineares

Os modelos lineares usados na avaliação do recurso eólico foram desenvolvidos com base na teoria desenvolvida por Jackson e Hunt [17], onde foi apresentada uma solução analítica para um escoamento bidimensional turbulento e adiabático em torno de uma montanha com baixa altitude.

O modelo linear mais usado na indústria eólica é o WAsP [1], que calcula o atlas do vento local com base em *inputs* de medições locais do vento e mapas de orografia e rugosidade, fazendo uso da teoria de Jackson e Hunt para retirar os efeitos do terreno e rugosidade dos valores medidos no mastro. A teoria é depois aplicada de forma inversa para estimar as condições de vento em qualquer ponto.

O WAsP determina a distribuição de vento nos vários pontos de interesse usando métodos estatísticos, particularmente a distribuição de Weibull (ver capítulo 2.1.4).

É também possível modelar o efeito da perda de velocidade do vento provocada por aerogeradores a montante, vulgarmente designada por efeito de esteira, usando o modelo PARK [18].

Os métodos usados pelo WAsP e por outros modelos lineares têm sido largamente utilizados na indústria eólica e revelado bons resultados em várias situações. No entanto, a fiabilidade dos resultados produzidos por modelos lineares está limitada a situações de baixa complexidade orográfica, por não possibilitarem a simulação de zonas de recirculação e turbulência.

2.2.2 Modelos não-lineares de mecânica de fluidos computacional (CFD)

Os modelos não-lineares de CFD (*Computational Fluid Dynamics*) são essencialmente códigos e algoritmos implementados em computador que resolvem numericamente as equações fundamentais da mecânica de fluidos. Estas equações surgem da aplicação das leis da conservação, nomeadamente a lei da conservação da massa (equação 2.7) e a lei da conservação da quantidade de movimento ou segunda lei de Newton (equação 2.8) [19].

$$\frac{dm}{dt} = 0 \ , \tag{2.7}$$

$$\frac{d(m\vec{V})}{dt} = \sum \vec{F}_{\text{ext}} \ . \tag{2.8}$$

em que m representa a massa, $\vec{V}$ o vetor velocidade, t o tempo e $\sum \vec{F}_{\text{ext}}$ o somatório das forças externas.

Teorema de transporte de Reynolds

Em mecânica de fluidos, é conveniente tratar os escoamentos definindo um espaço virtual, designado por volume de controlo (VC) delimitado por superfícies de controlo (SC), uma estratégia designada por descrição Euleriana. O teorema de transporte de Reynolds (TTR), é usado para converter a formulação Lagrangeana das leis fundamentais numa formulação Euleriana, *i.e.* diferencial em integral. No TTR, uma quantidade extensiva B é definida no interior do VC [20, 21, 19]:

$$B = \int_{\text{SIST}} \rho\beta d\mathcal{V} \ , \tag{2.9}$$

em que SIST representa todo um sistema de massa em estudo e β é uma quantidade intensiva, $\beta = dB/dm$. Assim,

$$\frac{dB}{dt} = \frac{d}{dt}\left(\int_{\mathcal{VC}} \rho\beta d\mathcal{V}\right) + \int_{\mathcal{SC}} \rho\beta\left(\vec{V}\cdot\vec{n}\right) d\mathcal{A} \ . \tag{2.10}$$

onde $\mathcal{A}$ representa a área, $\mathcal{V}$ o volume e $\vec{n}$ um vetor unitário normal à SC.

Conservação da massa

A partir do TTR (equação 2.10), pode ser formulada a equação da conservação da massa, com $B = m$ e $\beta = 1$ [19];

$$\frac{d}{dt}\left(\int_{\mathcal{VC}} \rho d\mathcal{V}\right) + \int_{\mathcal{SC}} \rho\left(\vec{V}\cdot\vec{n}\right) d\mathcal{A} = 0 \ . \tag{2.11}$$

Se o volume de controlo for infinitesimal, é obtido para a conservação da massa:

$$\frac{\partial\rho}{\partial t} + \nabla\cdot(\rho\vec{V}) = 0 \ . \tag{2.12}$$

Conservação da quantidade de movimento

A conservação da quantidade de movimento também pode ser definida a partir do TTR, definindo $\beta = \vec{V}$ [19];

$$\frac{d}{dt}\left(\int_{\mathcal{VC}} \rho\vec{V} d\mathcal{V}\right) + \int_{\mathcal{SC}} \rho\left(\vec{V}\vec{V}\vec{n}\right) d\mathcal{A} = \sum \vec{F}_{\text{ext}} \ , \tag{2.13}$$

em que $\sum \vec{F}_{\text{ext}}$ representa o somatório das forças externas que atuam no VC;

$$\sum \vec{F}_{\text{ext}} = \underbrace{-\vec{\nabla}p}_{\text{Gradiente de pressão}} + \underbrace{\nabla\vec{\tau}_{ij}}_{\text{Tensões viscosas}} + \rho\vec{g} \ , \tag{2.14}$$

em que $\vec{g}$ é o vetor da aceleração da gravidade. Considerando volumes de controlo infinitesimais, a equação diferencial geral que representa a conservação da quantidade de movimento;

$$\frac{\partial}{\partial t}(\rho\vec{V}) + \nabla(\rho\vec{V}\vec{V}) = -\vec{\nabla}p + \nabla\vec{\tau}_{ij} - \rho\vec{g} \ . \tag{2.15}$$

Para fluidos newtonianos e para escoamentos incompressíveis e isotérmicos, as tensões viscosas $\vec{\tau}_{ij}$, utilizando agora a notação de Einstein, são dadas por [20];

$$\vec{\tau}_{ij} = \mu\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) \ , \tag{2.16}$$

em que μ é a viscosidade dinâmica do fluido.

Equações de Navier-Stokes

Ao conjunto das equações da conservação da massa e de quantidade de movimento dá-se o nome de equações de Navier-Stokes. Para escoamentos incompressíveis (ρ constante no espaço e no tempo), a conservação da massa, tomando como ponto de partida a equação 2.12, toma a seguinte forma;

$$\nabla \cdot \vec{V} = 0 \;\Leftrightarrow\; \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} = 0 \;. \tag{2.17}$$

Para a conservação da quantidade de movimento, no seguimento das equações 2.15 e 2.16, obtém-se nas três componentes x, y e z, respetivamente;

$$\rho\frac{\partial u}{\partial t} + \rho u\frac{\partial u}{\partial x} + \rho v\frac{\partial u}{\partial y} + \rho w\frac{\partial u}{\partial z} = -\frac{\partial p}{\partial x} + \mu\left[\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2}\right] \;, \tag{2.18}$$

$$\rho\frac{\partial v}{\partial t} + \rho u\frac{\partial v}{\partial x} + \rho v\frac{\partial v}{\partial y} + \rho w\frac{\partial v}{\partial z} = -\frac{\partial p}{\partial y} + \mu\left[\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} + \frac{\partial^2 v}{\partial z^2}\right] \;, \tag{2.19}$$

$$\rho\frac{\partial w}{\partial t} + \rho u\frac{\partial w}{\partial x} + \rho v\frac{\partial w}{\partial y} + \rho w\frac{\partial w}{\partial z} = -\frac{\partial p}{\partial z} + \mu\left[\frac{\partial^2 w}{\partial x^2} + \frac{\partial^2 w}{\partial y^2} + \frac{\partial^2 w}{\partial z^2}\right] - \rho g \;. \tag{2.20}$$

Para iniciar a resolução numérica destas equações, é necessária a especificação de condições de fronteira nos limites do domínio em estudo para todas as variáveis.

Embora os modelos CFD exijam recursos computacionais mais elevados comparativamente aos modelos lineares, algumas das vantagens da sua utilização na avaliação do recurso eólico são aqui listadas [3]:

- Produzem bons resultados para orografia complexa e zonas florestadas;
- Modelam zonas de recirculação e turbulência;
- Modelam o efeito da estratificação térmica (ver secção 2.1.2), através da resolução da equação da energia juntamente com as equações 2.17, 2.18, 2.19 e 2.20;
- Possibilitam estudos em regime transiente.

2.2.3 Modelação da turbulência

Todos os escoamentos atmosféricos, principalmente na camada limite atmosférica, são de natureza turbulenta. Por esse motivo, é importante o estudo de formas de modelar a turbulência de modo a equilibrar os custos computacionais envolvidos com a precisão dos resultados obtidos. As técnicas usadas para a simulação de escoamentos turbulentos podem ser divididas em três grandes grupos [22, 20]:

Direct Numerical Solution (DNS)

A técnica DNS consiste em determinar todas as variáveis diretamente da solução numérica das equações de Navier-Stokes, em quase todas as escalas espaciais e temporais. Isso implica um custo computacional muito elevado e que aumenta bastante com o aumento da turbulência do escoamento, porque os volumes de controlo devem ser muito pequenos para permitir a modelação de todos ou quase todos os vórtices turbulentos [23]. Em escoamentos atmosféricos esta técnica é ainda impraticável, uma vez que o estudo destes escoamentos está associado a largas escalas e números de Reynolds, Re, muito elevados.

Large Eddy Simulation (LES)

Na técnica LES, o objetivo é determinar diretamente os vórtices de maiores dimensões e usar um modelo de turbulência para os vórtices de pequenas dimensões. Um dos objetivos desta estratégia é diminuir os recursos computacionais em relação à DNS, possibilitando a utilização de malhas de cálculo menos refinadas. Esta técnica já foi usada em escoamentos atmosféricos, por exemplo [24] entre vários outros, onde foram efetuadas simulações LES de escoamentos atmosféricos neutralmente estratificados sobre terreno moderadamente complexo.

Reynolds Averaged Navier-Stokes (RaNS)

No caso da técnica RaNS, toda a turbulência do escoamento é descrita por um modelo de turbulência. Esta estratégia é baseada no estudo da evolução estatística como alternativa à solução numérica direta e continua a ser a técnica mais utilizada na modelação de escoamentos turbulentos, devido aos recursos computacionais serem significativamente inferiores comparativamente às outras técnicas.

Essencialmente, a análise RaNS combina uma média temporal das equações de Navier-Stokes com um modelo para as flutuações de uma qualquer variável do escoamento no tempo, a este processo de decomposição de variável é chamado decomposição de Reynolds. Tomando como exemplo a velocidade u,

$$u(\vec{x},t) = \bar{u}(\vec{x}) + u'(\vec{x},t) \quad , \tag{2.21}$$

em que $\vec{x}$ é o vetor posição, t é um instante de tempo e $\bar{u}(\vec{x})$ é dado, para o período T em análise por,

$$\bar{u}(\vec{x}) = \frac{1}{\mathrm{T}} \int_0^{\mathrm{T}} u(\vec{x},t)dt \quad . \tag{2.22}$$

Da decomposição de Reynolds aplicada às variáveis do escoamento nas equações de Navier-Stokes, vão surgir termos adicionais de tensões, chamadas tensões de

Reynolds $\vec{\tau}_{ij} = -\rho\overline{u_i' u_j'}$, que resultam de flutuações turbulentas e são nos modelos RaNS aproximadas com recurso ao princípio de Boussinesq [25, 26], onde é definido o conceito de viscosidade turbulenta μ_t,

$$\vec{\tau}_{ij} = -\rho\overline{u_i' u_j'} \equiv \mu_t \left(\frac{\partial \bar{u}_i}{\partial x_j} + \frac{\partial \bar{u}_j}{\partial x_i} \right) - \rho \frac{2}{3} k \delta_{ij} \quad , \tag{2.23}$$

em que δ_{ij} é o delta de Kronecker e toma o valor 1 se $i = j$ ou 0 em caso contrário [19]. A energia cinética da turbulência, k, é definida por,

$$k \equiv \frac{1}{2} \left(\overline{u'^2} + \overline{v'^2} + \overline{w'^2} \right) \quad . \tag{2.24}$$

As tensões de Reynolds são determinadas usando um modelo de turbulência e usando o conceito da viscosidade turbulenta μ_t.

2.2.3.1 Modelo $k - \varepsilon$

O modelo $k - \varepsilon$ resolve as equações de transporte para duas quantidades de turbulência, k e ε [22]. A versão *standard* foi desenvolvida por Launder e Spalding [27] e define a viscosidade turbulenta μ_t como,

$$\mu_t = \rho C_\mu \frac{k^2}{\varepsilon} \quad , \tag{2.25}$$

onde C_μ é uma constante adimensional e ε é a taxa de dissipação da energia cinética da turbulência. O modelo requer duas equações de transporte adicionais para k e ε [20],

$$\rho \frac{\partial \left(\overline{u_j} k \right)}{\partial x_j} = \frac{\partial}{\partial x_j} \left[\left(\mu + \frac{\mu_t}{\sigma_k} \right) \frac{\partial k}{\partial x_j} \right] + P_k - \rho\varepsilon \quad , \tag{2.26}$$

$$\rho \frac{\partial \left(\overline{u_j} \varepsilon \right)}{\partial x_j} = \frac{\partial}{\partial x_j} \left[\left(\mu + \frac{\mu_t}{\sigma_\varepsilon} \right) \frac{\partial \varepsilon}{\partial x_j} \right] + \frac{C_1 \varepsilon}{k} P_k - \frac{C_2 \rho \varepsilon^2}{k} \quad , \tag{2.27}$$

sendo P_k a produção mecânica de turbulência, definida por,

$$P_k = \vec{\tau}_{ij} \frac{\partial \bar{u}_i}{\partial x_j} \quad , \tag{2.28}$$

em que $\vec{\tau}_{ij}$ representa as tensões de Reynolds, dadas pelo princípio de Boussinesq (equação 2.23). As constantes C_μ, σ_k, σ_ε, C_1 e C_2 são adimensionais e empíricas, obtidas com a finalidade de ajustar o modelo. Na tabela 2.2 estão indicados os valores para as constantes do modelo *standard*, obtidas por Launder e Sharma [28], e valores que melhor se adequam a escoamentos atmosféricos, propostas por Beljaars *et al.* [29];

Tabela 2.2 – Constantes adimensionais empíricas do modelo $k-\varepsilon$ *standard* e atmosférico

	Standard	Atmosférico
C_μ	0.090	0.033
σ_k	1.000	1.000
σ_ε	1.300	1.850
C_1	1.440	1.440
C_2	1.920	1.920

2.2.3.2 Outros modelos RaNS

Embora o modelo $k-\varepsilon$ seja dos modelos RaNS mais usados na modelação da turbulência em simulações CFD, outros modelos revelaram produzir melhores resultados em algumas situações, dependendo das características do escoamento. O modelo *Realizable* $k-\varepsilon$ [30] revelou melhores resultados para situações em que existem gradientes de pressão elevados, já o modelo *RNG* $k-\varepsilon$ [31] foi desenvolvido com objetivo de descrever melhor o escoamento a pequenas escalas.

Também é comum a utilização de modelos RaNS que descrevem a turbulência a partir de uma quantidade designada por frequência de dissipação da energia cinética da turbulência ω em vez da sua taxa de dissipação ε , como é o caso dos modelos $k-\omega$ [32] e *SST* $k-\omega$ [33], que revelam bons resultados em escoamentos com elevada vorticidade.

2.3 Códigos CFD

2.3.1 OpenFOAM

O OpenFOAM (*Open Field Operation and Manipulation*) [5] é um *software open source* desenvolvido no *Imperial College London* e disponibilizado pela primeira vez em 2004. Atualmente é distribuído sob a *GNU General Public License* pela *OpenFOAM Foundation*, o que significa que o utilizador pode modificar e manipular o *software*.

Desenvolvido com base em código C++, o OpenFOAM é, na sua essência, uma biblioteca de operadores de campos escalares e vetoriais que inclui vários *solvers* para a solução de problemas de CFD como, por exemplo, escoamentos incompressíveis ou compressíveis, sujeitos a processos de transferência de calor, mudança de fase, combustão e outros. Para além disso, inclui várias utilidades para o pré-processamento, como geração ou manipulação da malha de cálculo, e pós-processamento, como extração de dados.

O OpenFOAM tem sido cada vez mais usado no estudo de escoamentos atmosféricos, o que tem motivado o desenvolvimento de ferramentas com o objetivo de melhorar a sua aplicabilidade a esta área (ver [6, 10]).

2.3.2 Outros modelos CFD usados na indústria eólica

Para além do OpenFOAM, existem outros modelos CFD que são usados na indústria eólica, dos quais se destacam:

- **WINDIE™** - O WINDIE é um modelo CFD desenvolvido por um grupo de investigadores do Instituto Superior de Engenharia do Porto (ISEP) especificamente para a modelação de escoamentos atmosféricos e avaliação do recurso eólico que nasceu do trabalho desenvolvido em [13]. Este modelo já contribuiu para a avaliação de mais de 8 GW de potência em recurso eólico, em mais de 25 países [3]. O WINDIE possibilita a modelação avançada de floresta, acoplamento com modelos de mesoescala e modelação de efeitos térmicos e da força aparente Coriolis.
- ***Meteodyn*** - O *Meteodyn* é um modelo CFD comercial usado internacionalmente em estudos de consultoria para a indústria eólica que possibilita a previsão do recurso eólico através da análise das características do vento e simulação de escoamentos atmosféricos sobre terreno real. Neste modelo também é possível a modelação de floresta e efeitos térmicos, aplicáveis ao estudo de escoamentos atmosféricos e a análise de microclimas urbanos e estudos da influência do vento em edifícios, pontes e transportes [34].

2.4 Modelação de mesoescala

Modelos de mesoescala são modelos numéricos de CFD que pela sua corrente utilização em previsão meteorológica, são vulgarmente designados por *Numerical Weather Prediction Models* (NWP). A modelação de mesoescala consiste em descrever escoamentos atmosféricos a escalas de centenas a milhares de quilómetros com malhas de cálculo de resolução máxima de 1 km.

Estes modelos tipicamente partem de um estado inicial e avançam no tempo fazendo uso de condições de fronteira provenientes de modelos globais, permitindo descrever certas características regionais do estado meteorológico, como por exemplo formação de nuvens, distribuição de precipitação e padrões de vento, sendo esta última de particular interesse para este trabalho.

2.4.1 Modelo WRF

O modelo WRF (*Weather Research and Forecast*) [4] é um modelo de mesoescala *open source* que integra um *solver* euleriano capaz de modelar a compressibilidade e os efeitos da instabilidade atmosférica. No WRF, a pressão hidrostática é considerada como uma variável independente, de acordo com a teoria proposta por Laprise [35], utilizando um sistema de coordenadas que segue o terreno através de níveis η de pressão constante (ver figura 2.5) [36].

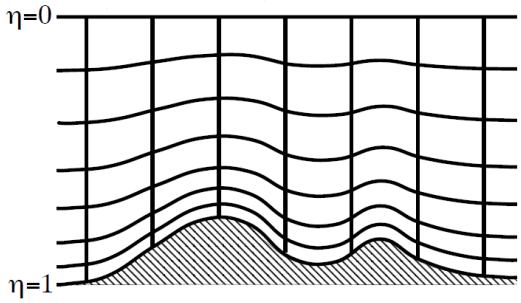


Figura 2.5 – Sistema de coordenadas de níveis η usado no WRF (retirado de [36])

em que;

$$\eta = \frac{p - p_t}{p_s - p_t} \quad , \tag{2.29}$$

onde p_s e p_t representam a pressão à superfície e no topo do domínio, respetivamente.

Na malha de cálculo do WRF, as componentes da velocidade estão desfasadas relativamente às quantidades termodinâmicas, a esta técnica dá-se o nome de *staggering* (ver figura 2.6) [36].

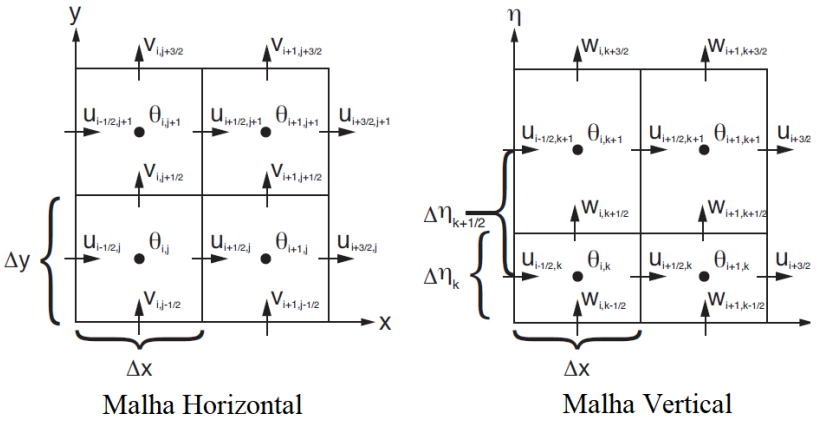


Figura 2.6 – Técnica *staggering* usada no WRF (adaptado de [36])

O WRF tem também capacidade de definir múltiplas resoluções na malha de cálculo horizontal, uma estratégia designada por *nesting* (ver figura 2.7).

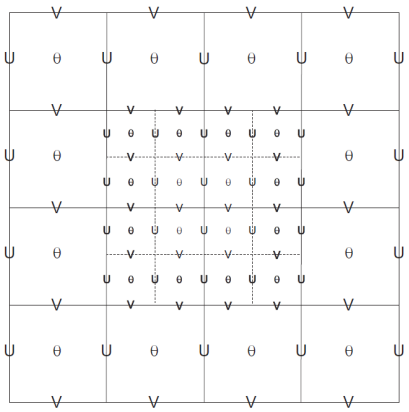


Figura 2.7 – Técnica *nesting* usada no WRF (retirado de [36])

No WRF, a modelação da turbulência segue a técnica VLES (*Very Large Eddy Simulations*), complementada com modelos paramétricos da camada limite planetária.

No pré-processamento, o WRF requer como *inputs*:

- **Dados geográficos**: Informações sobre o terreno.
- **Dados meteorológicos**: Resultados de modelos atmosféricos globais que são usados como condições de fronteira e interpolados no domínio de mesoescala definido no WRF, podendo assimilar ainda medições para correção de simulações no passado (*hindcasts*).

2.5 Acoplamento microescala - mesoescala

Os modelos de mesoescala têm uma resolução significativamente baixa, que não permite aferir com precisão os fenómenos que ocorrem no escoamento a pequenas escalas, mas têm a vantagem de os resultados considerarem fenómenos como a compressibilidade, instabilidade atmosférica e sobretudo de incorporarem os padrões de vento meteorológicos regionais.

Uma possível estratégia é tirar partido dos resultados de modelos de mesoescala em modelos de microescala, como condições de fronteira, permitindo assim uma melhor representação das mesmas.

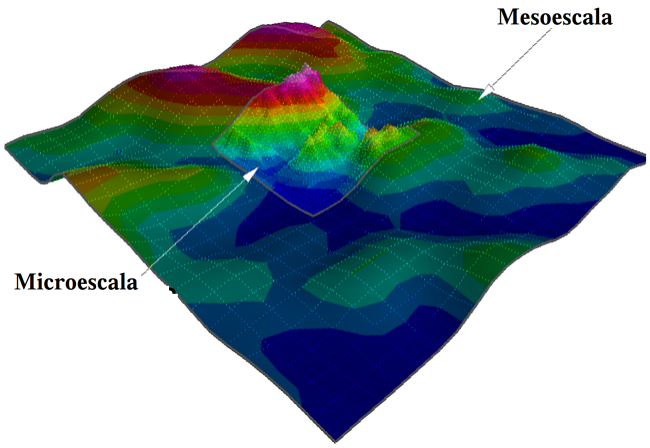


Figura 2.8 – Acoplamento microescala-mesoescala (adaptado de [3])

Uma das técnicas de acoplamento microescala - mesoescala foi apresentada em [2], num trabalho em que é efetuado o acoplamento entre o modelo de microescala WINDIE e o modelo de mesoescala WRF. A estratégia de acoplamento usada consiste na inclusão de resultados do WRF na malha de cálculo de microescala, por recurso a interpolação trilinear e linear no espaço e tempo, respetivamente.

Outros estudos apresentados em [37] e [10] , consistem no acoplamento de resultados de domínios mesoescala do WRF em simulações microescala CFD no OpenFOAM.

3 Simulação de escoamentos atmosféricos com condições de fronteira idealizadas

No presente capítulo será apresentado o caso de estudo e serão descritos os procedimentos adotados para a realização de simulações CFD de escoamentos atmosféricos neutralmente estratificados com condições de fronteira idealizadas, aplicando esses procedimentos especificamente ao caso em estudo. Será apresentada a metodologia adotada para o pré-processamento, pós-processamento e validação dos resultados obtidos.

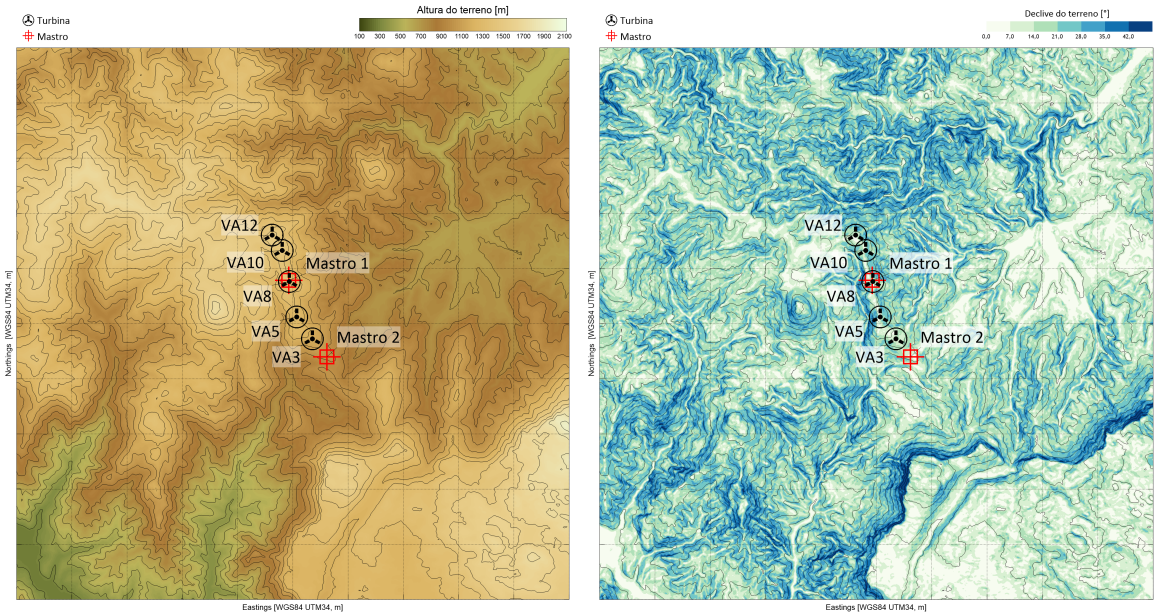
3.1 Descrição do Caso de Estudo

3.1.1 Considerações gerais e orografia

O local em estudo situa-se na Bósnia-Herzegovina, onde o terreno é considerado de complexidade orográfica elevada. Para este caso de estudo, é pretendida a instalação de cinco turbinas eólicas e o local é dotado de dois mastros de medição (ver figura 3.1).

A altura do terreno varia entre 0 a pouco mais de 2000 m, e existem zonas com montanhas de declive acentuado, que pode atingir os 40°. Estas características orográficas levaram ao aparecimento de problemas numéricos em simulações para algumas direções do vento, associadas a grandes bolsas de recirculação e turbulência elevada.

As turbinas eólicas têm 5 MW de potência e o seu eixo dista 102.5 m do solo. O mastro 1 efetuou medições a 35, 55, 76 e 81 metros acima do nível do solo e o mastro 2 efetuou medições a 20, 40 e 60 metros acima do nível do solo.



(a) (b)

Figura 3.1 – Altura (a) e declive (b) do terreno do caso de estudo

3.1.2 Condições de vento locais

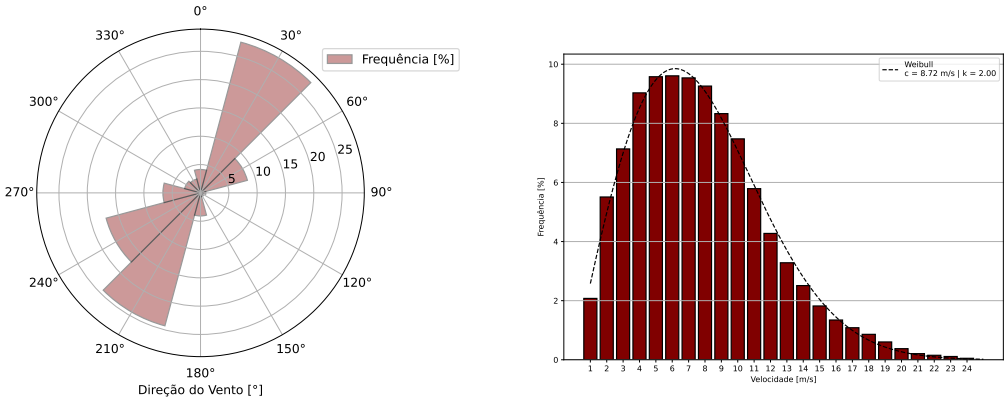
Os dados obtidos pelo mastro 1 foram registados entre 09/12/2018 e 07/07/2021, com taxas de amostragem de 1 Hz transformadas em médias de 10 minutos como é prática corrente na indústria eólica. Para avaliar as características do vento no local, as medições foram filtradas em intervalos de 30° (figura 3.2a) e em intervalos de 1 m/s (figura 3.2b). Para esse efeito foram desenvolvidos *scripts* em *Python* que efetuam a leitura dos ficheiros que contêm os dados das medições e filtram esses dados, fazendo ainda a estimativa dos parâmetros k e c associados à distribuição de Weibull (ver capítulo 2.1.4), pelo método dos momentos. Este método calcula o fator de forma adimensional k e o fator de escala c de acordo com as seguintes equações ([16]),

$$k = \left(\frac{0.9874}{\sigma/v_m}\right)^{1.0983} \quad \text{ou} \quad k = \left(\frac{\sigma}{v_m}\right)^{-1.086} \quad , \tag{3.1}$$

$$c = \frac{v_m}{\Gamma\left(1+\frac{1}{k}\right)} \quad , \tag{3.2}$$

onde σ representa o desvio padrão, v_m a velocidade média e Γ é uma função matemática, designada por função *gamma*.

Os resultados das medições do mastro 1, efetuadas a 81 metros acima do nível do solo, estão sumariados na figura 3.2. O código para processamento das medições e cálculo da distribuição de Weibull pode ser consultado no Apêndice A.



(a) (b)

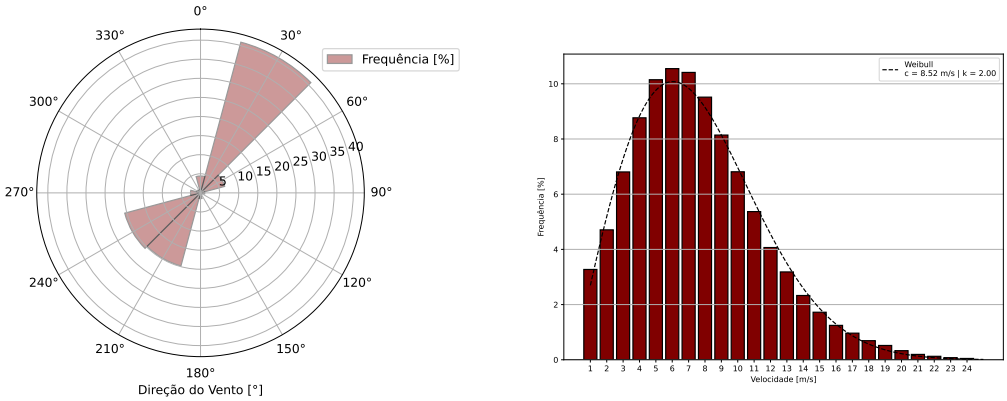
Figura 3.2 – Rosa de frequências (a) e histograma de frequências (b) dos dados medidos pelo mastro 1 a 81 metros acima do nível do solo

No local em estudo, de acordo com os dados registados pelo mastro 1 a 81 metros acima do nível do solo, o vento é predominante de 30° e 210°, verificando-se também alguma frequência de 240°.

Séries de longo termo

Com o objetivo de se obter uma melhor caracterização do padrão de vento local, foram utilizadas séries temporais de longo termo para comparação com os dados obtidos pelos mastros, que medem durante períodos mais curtos. Neste trabalho foram utilizadas duas abordagens: dados de longo-termo provindos de uma simulação de mesoescala (LT-MESO); e uma série temporal de longo termo obtida com recurso à base de dados ERA5 (LT-ERA5) [38].

Os dados de longo-termo de mesoescala provieram de uma simulação de longo-termo de mesoescala que, neste caso, abarcou um período de 11 anos (2010 a 2021). Os dados de longo-termo são depois obtidos por correlação entre os resultados do modelo de mesoescala e as medições, mas tal processo, denominado de *Measure-Correlate-Predict* (MCP) foi, neste caso, feito no domínio da frequência e não diretamente com a série temporal de mesoescala. Como tal, os dados LT-MESO que foram disponibilizados encontram-se em formato TAB, uma tabela de frequências por velocidade e direção. Para simplificar o processo de tratamento destes dados, este ficheiro TAB foi convertido numa série temporal artificial, com registos de velocidade e direção com as mesmas frequências do ficheiro TAB fornecido. Com recurso à mesma metodologia descrita anteriormente para tratamento dos dados, apresentam-se os resultados na figura 3.3.



(a) (b)

Figura 3.3 – Rosa de frequências (a) e histograma de frequências (b) da série LT-MESO

De acordo com os dados LT-MESO, mantém-se a conclusão de que o vento é predominante nas três direções referidas anteriormente, mas destacando-se a direção de 30° que, com base nestes dados, aparenta ser consideravelmente mais frequente do que as restantes.

Os dados LT-MESO têm a vantagem de incluir resultados de um modelo de mesoescala a uma resolução de cerca de 3 km e de já incorporarem correlação com as medições. Por outro lado, não contêm variação temporal já que a informação foi fornecida no domínio da frequência. Para obviar a isto, fez-se uso dos dados LT-ERA5 (de 2008 a 2022). Através desta série foi estimada a variação da velocidade média anual em relação à velocidade média global correspondente ao período de longo termo em análise, que neste caso foi de 2008 a 2022.

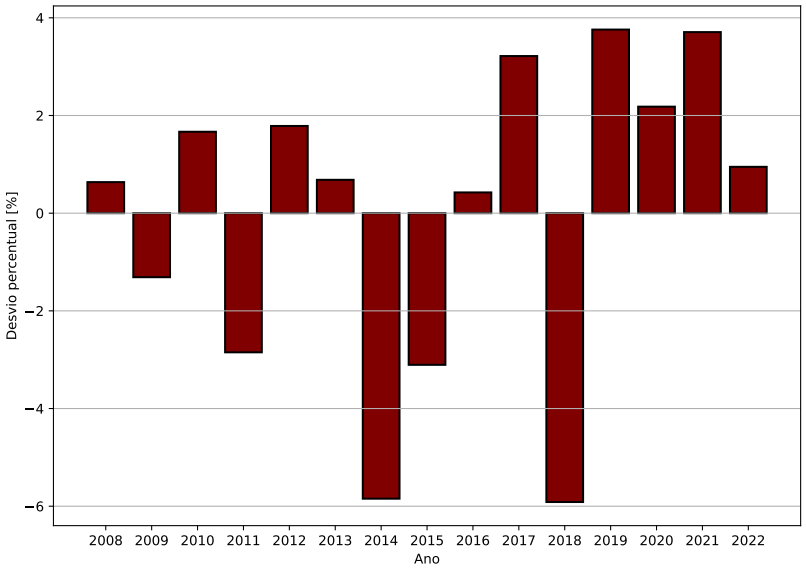


Figura 3.4 – Desvio percentual entre a velocidade média anual e a velocidade média global do período de longo termo

Esta série de longo termo foi também utilizada para determinar o desvio percentual entre a velocidade média do período de medição correspondente a cada um dos mastros e a velocidade média global correspondente a todo o período de longo termo em análise.

Tabela 3.1 – Desvio entre a velocidade média nos períodos de medição dos mastros e a velocidade média global da série de longo termo (LT-ERA5)

	Período	Desvio Percentual
Período global de longo termo	01/01/2008 - 31/12/2022	-
Período de medição do mastro 1	09/12/2018 - 07/07/2021	+4.0115 %
Período de medição do mastro 2	01/05/2009 - 30/09/2010	−3.8967 %

Esta análise às condições de vento locais revela que a produção do parque eólico seria provavelmente sobrestimada usando o mastro 1 (e inversamente com o mastro 2), mostrando a necessidade de correlacionar com as séries de longo de termo.

3.2 Pré-processamento

3.2.1 Geração da malha de cálculo

A malha de cálculo foi gerada com recurso ao código `write_blockMeshDict.f90`, desenvolvido em [6], que recebe ficheiros em formato VTK (*Visualization Toolkit*) gerados por um outro código FORTRAN que faz parte do WINDIE, o `gsrf3.f90`, cuja função é definir os pontos para a malha horizontal tendo como *input* dados reais de topografia (ficheiro `topo.dat`). O código `write_blockMeshDict.f90` define os pontos na vertical tomando como base a superficie gerada pelo `gsrf3.f90` e escreve os blocos que definem toda a malha de cálculo no ficheiro `blockMeshDict`, que é o *input* necessário à execução do `blockMesh`, um dos geradores de malha do OpenFOAM.

Os códigos `gsrf3.f90` e `write_blockMeshDict.f90` geram uma malha de cálculo na qual, dentro de uma área central definida, é aplicada uma melhor resolução, uma vez que se pretende que seja dentro desta área que estejam as turbinas eólicas e onde já existem mastros de medição. Fora dessa área, o tamanho dos volumes de controlo vai aumentando gradualmente em progressão geométrica até atingir as extremidades do domínio computacional. A malha terá também maior resolução junto ao solo e a dimensão vertical dos volumes de controlo irá aumentar em altura de acordo com um fator de expansão definido.

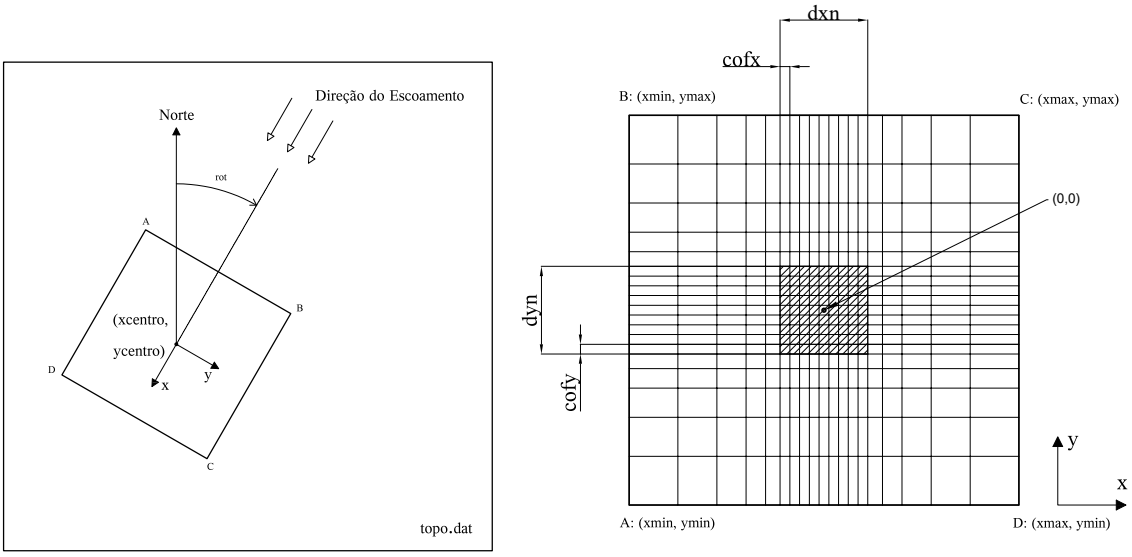
A malha de cálculo é gerada de acordo com os seguintes parâmetros:

- `nig`, `njg` - Número de volumes de controlo em x e em y, respetivamente. Estes parâmetros seguem a estratégia usada no WINDIE, em que é considerado o

número de volumes de controlo acrescido dos dois nós nas arestas dos limites do domínio. Assim, no OpenFOAM o número de volumes de controlo em x e em y será (`nig` - 2) e (`njg` - 2), respetivamente.

- `nz` - Número de volumes de controlo em z;
- `cofx`, `cofy` - Dimensões x e y, respetivamente, dos volumes de controlo mais pequenos, localizados na área central de maior resolução;
- `R` - Fator de expansão vertical;
- `dxn`, `dyn` - Dimensão x e y, respetivamente, da área de maior resolução;
- `xmax`, `xmin`, `ymax`, `ymin`, `sky` - Limites do domínio computacional;
- `npassfilter` - Fator de suavização do terreno nas extremidades;
- `alphalayer` - Espessura nas extremidades onde é aplicada a suavização descrita no ponto anterior.
- `rot` - Direção do vento a simular.

Na figura 3.5 estão ilustrados alguns destes parâmetros.



(a) (b)

Figura 3.5 – Parâmetros do `gsrf3.f90`

Estes *inputs* são especificados pelo utilizador em ficheiros de configuração que são lidos pelos códigos `gsrf3.f90` e `write_blockMeshDict.f90`, ambos escritos em linguagem FORTRAN. Um exemplo de uma malha gerada para este caso de estudo é apresentada na figuras 3.6.

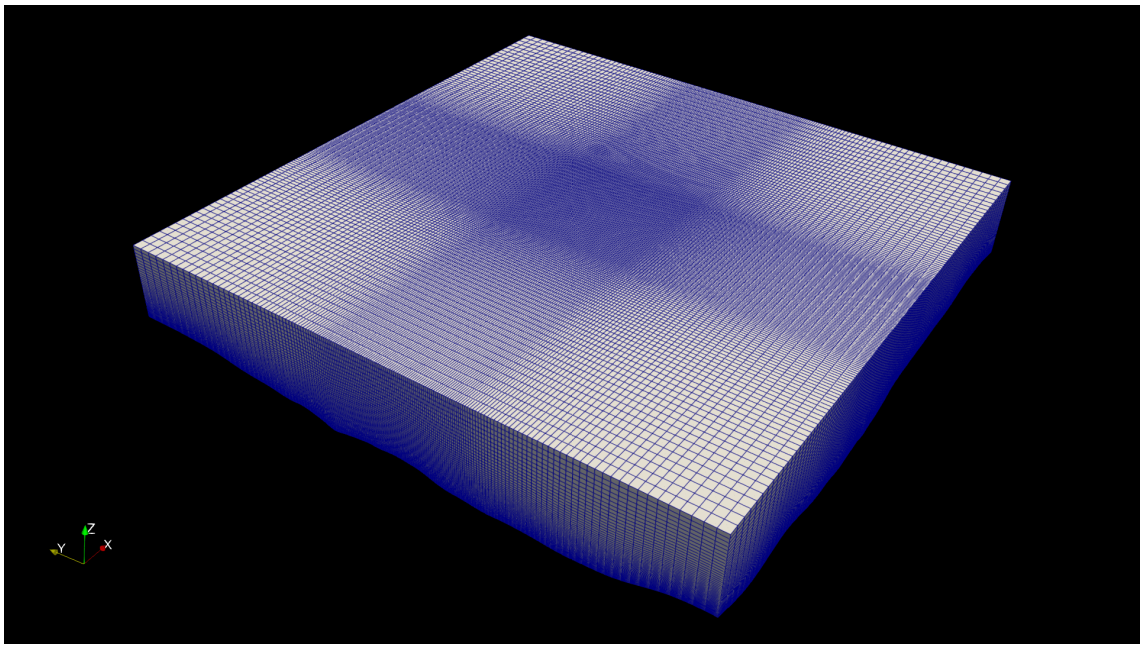


Figura 3.6 – Exemplo de malha de cálculo gerada pelo `gsrf3.f90` e `write_blockMeshDict.f90`

Com a realização de simulações com malhas de cálculo de maior resolução, verificou-se a necessidade de alterar o código `write_blockMeshDict.f90` para escrever mais algarismos na escrita dos blocos no `blockMeshDict`, que estavam até aqui limitados a seis algarismos.

Na geração do ficheiro `blockMeshDict` foram definidas as seis fronteiras do domínio computacional, de acordo com a tabela 3.2.

Tabela 3.2 – Definição das fronteiras no ficheiro `blockMeshDict`

Nome da Fronteira	Atributo
ground	`wall`
inlet	`patch`
outlet	`patch`
front	`symmetryPlane`
back	`symmetryPlane`
sky	`symmetryPlane`

3.2.2 Condições de fronteira

As condições de fronteira idealizadas descritas neste capítulo têm como base os pressupostos que, nestas simulações, os escoamentos atmosféricos são incompressíveis e neutralmente estratificados. Assim, perfis idealizados de velocidade, de energia cinética da turbulência (k) e da sua taxa de dissipação (ε) foram aplicados na fronteira de entrada, também referida neste capítulo como *inlet*.

Os perfis de entrada foram definidos de acordo com as equações propostas em [39], que para a velocidade, k e ε são definidos por,

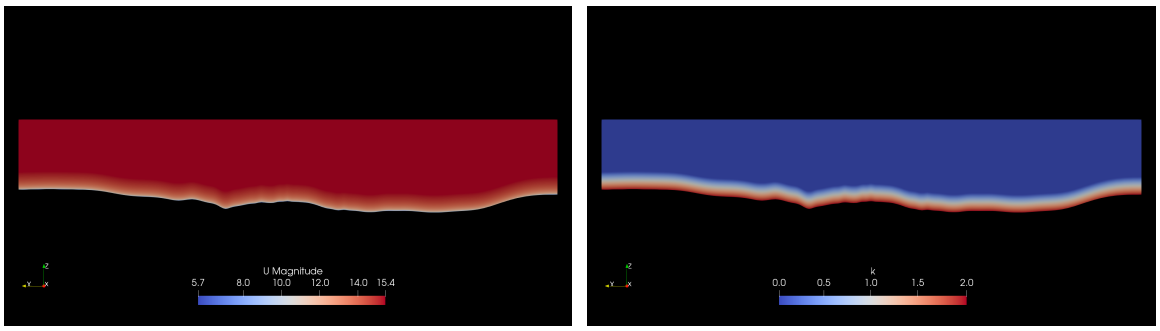
$$u = \begin{cases} u_*/\kappa \cdot \ln(1 + z/z_0), & z < \delta \\ u_*/\kappa \cdot \ln((\delta + z)/z_0), & z \geq \delta \end{cases}, \tag{3.3}$$

$$k = \begin{cases} u_*^2/C_\mu \cdot (1 - z/\delta), & z \leq 0.99\delta \\ u_*^2/(100C_\mu^{1/2}), & z > 0.99\delta \end{cases}, \tag{3.4}$$

$$\varepsilon = \begin{cases} C_\mu^{3/4} k^{3/2}/(kz), & z \leq 0.95\delta \\ C_\mu^{3/4} k^{3/2}/(0.95\delta k), & z > 0.95\delta \end{cases}, \tag{3.5}$$

sendo δ a altura da camada limite.

A especificação de condições de fronteira no OpenFOAM é efetuada via *Text User Interface* (TUI) por ficheiros de texto que estão na pasta `0` do respetivo caso de estudo. Para o cálculo dos perfis idealizados, foi usado o código `write_bCs.f90`, desenvolvido em [6], que através das equações 3.3, 3.4 e 3.5, calcula os perfis das respetivas quantidades e aplica-os na fronteira de entrada, tendo em consideração a sua geometria e malha de cálculo. Estes perfis são escritos em ficheiros *dat* ASCII para posterior leitura pelo OpenFOAM.



(a) (b)

Figura 3.7 – Perfis idealizados da velocidade (a) e energia cinética da turbulência (b) aplicados na fronteira de entrada

Para as fronteiras *front, back e sky* foi aplicada uma condição de simetria. A aplicação desta condição de fronteira implica que a componente normal da velocidade nestas fronteiras será zero, resultando numa condição de fluxo nulo. Na fronteira *ground*, foi aplicada uma condição de velocidade nula (não escorregamento). Para as quantidades k e ε, foram aplicadas leis de parede disponíveis no OpenFOAM, identificadas na tabela 3.3.

Tabela 3.3 – Especificação das condições de fronteira no OpenFOAM para a velocidade, k e ε

Fronteira/*Patch*	Quantidade	Cond. Fronteira	Valor
ground	U	`fixedValue;`	`uniform (0 0 0);`
inlet		`fixedValue;`	Equação 3.3
outlet		`inletOutlet;`	`uniform (0 0 0);`
front, back, sky		`symmetryPlane;`	-
ground	k	`kqRWallFunction;`	`uniform 2.028;`
inlet		`fixedValue;`	Equação 3.4
outlet		`zeroGradient;`	-
front, back, sky		`symmetryPlane;`	-
ground	ε	`epsilonWallFunction;`	`uniform 18.183;`
inlet		`fixedValue;`	Equação 3.5
outlet		`zeroGradient;`	-
front, back, sky		`symmetryPlane;`	-

A função de parede `kqRWallFunction`, usada na fronteira *ground* para a quantidade k, atribui um valor constante na fronteira, impondo desta forma um gradiente nulo [40].
Relativamente à função de parede `epsilonWallFunction`, esta adota uma formulação empírica para modelar o comportamento da taxa da dissipação da energia cinética da turbulência na parede, dada por ([22]),

$$\varepsilon_{wall} = \frac{u_*^3}{k z_p} \ , \tag{3.6}$$

em que z_p representa a distância entre a parede e o centro geométrico do volume de controlo adjacente.

Quanto à pressão, a definição das condições de fronteira estão identificadas na tabela 3.4.

Tabela 3.4 – Especificação das condições de fronteira no OpenFOAM para a pressão

Fronteira/*Patch*	Quantidade	Cond. Fronteira	Valor
ground, inlet	p	`zeroGradient;`	-
sky, front, back		`symmetryPlane;`	-
outlet		`fixedValue;`	`uniform 0;`

3.2.3 Modelação da rugosidade

A rugosidade do terreno foi modelada com recurso ao código `write_z0.f90`, desenvolvido em [6], com algumas alterações que serão referidas nesta secção. Esta ferramenta permite que a rugosidade do terreno seja mapeada de três maneiras:

- Rugosidade uniforme - Neste caso um valor constante de z_0 que será adotado em todos os pontos do terreno é especificado pelo utilizador;
- Rugosidade definida por uma equação que impõe uma variação linear de z_0 [41];
- Rugosidade definida por mapa de rugosidade - Nesta opção, o utilizador deve introduzir como *input* um mapa de rugosidade que contém valores de z_0 em função das coordenadas x e y para o local em estudo.

Neste trabalho foi usada esta última opção, onde um mapa de rugosidade é obtido a partir de dados reais (ficheiro `roug.dat`) que são manipulados pelo `groug.f90`, um código que também faz parte do WINDIE.

De acordo com [6], no OpenFOAM a rugosidade é modelada como condição de fronteira da viscosidade cinemática turbulenta (ν_t), sendo a rugosidade aerodinâmica z_0 um parâmetro especificado no ficheiro `nut` da pasta `0`. Na versão original do código `write_z0.f90` foi usada a condição de fronteira `nutkAtmRoughWallFunction`. Nas versões mais recentes do OpenFOAM esta condição de fronteira foi descontinuada, pelo que foi efetuada uma alteração ao código `write_z0.f90` para fazer uso da `atmNutkWallFunction`, cuja formulação é (ver [42, 43]),

$$\nu_{t_w} = \nu_w \left(\frac{y^+ \kappa}{\ln\left(\max\left(E', 1 + 10^{-4}\right)\right)} - 1 \right) \quad , \tag{3.7}$$

em que ν_w representa a viscosidade cinemática do fluido. Os parâmetros u_*, y^+ e E' são obtidos a partir de,

$$u^* = C_\mu^{1/4} \sqrt{k} \quad , \tag{3.8}$$

$$y^+ = \frac{u_* y}{\nu_w} \quad , \tag{3.9}$$

$$E' = \frac{y + z_0}{z_0} \quad . \tag{3.10}$$

Na tabela 3.5 está identificada a especificação das condições de fronteira no ficheiro `nut`.

Tabela 3.5 – Especificação das condições de fronteira no OpenFOAM para a viscosidade turbulenta (ν_t)

Fronteira/*Patch*	Quantidade	Cond. Fronteira	Valor
ground	ν_t	`atmNutkWallFunction;`	`-`
sky, front, back		`symmetryPlane;`	`-`
inlet, outlet		`fixedValue;`	`uniform 0;`

Os parâmetros C_μ, κ, E' e z_0 para a condição de fronteira `atmNutkWallFunction` foram configurados no ficheiro `nut` de acordo com o seguinte excerto:

```
ground
{
    type            atmNutkWallFunction;
    Cmu             0.033;
    kappa           0.41;
    E               9.8;
    z0              nonuniform List<scalar>
    9604
(
    *
);
    value           uniform 0;
}
```

Acima '`*`' representa uma lista de valores do mapa de rugosidade definidos no ficheiro `roug.dat`, interpolados para os pontos da malha de cálculo da fronteira *ground*.

3.2.4 Modelação da turbulência

Nesta dissertação, todas as simulações foram realizadas usando o modelo RaNS $k-\varepsilon$, descrito no capítulo 2.2.3, com as constantes empíricas do modelo ajustadas para escoamentos atmosféricos, indicadas nesse mesmo capítulo.

3.2.5 Técnicas e recursos computacionais

As simulações finais realizadas no âmbito deste trabalho foram efetuadas com recurso a *cloud computing*, usando o sistema pbs (*portable batch script*) para a submissão das simulações na *cloud*. A utilização de computação de alta *performance* provém da necessidade de efetuar várias simulações CFD com malhas de cálculo de resolução consideravelmente elevada. Também foram elaborados *bash scripts* para a automatização das tarefas como o pré-processamento e lançamento de múltiplas simulações sequencialmente.

As simulações foram efetuadas em paralelo, dividindo o domínio por 16 núcleos (4 em x, 4 em y e 1 em z) usando um particionamento simples. Evitou-se a partição em z devido aos maiores gradientes observados nesta direção.

3.3 Resultados

3.3.1 Configurações do conjunto final de simulações

O conjunto final de simulações foi realizado para 12 direções de vento, com a seguinte configuração dos parâmetros apresentados na secção 3.2.1, para a malha de cálculo:

- `xmax` = `ymax` = 15000 m, para todas as direções;
- `xmin` = `ymin` = −15000 m, para todas as direções;
- `sky` = 5500 m, para todas as direções.

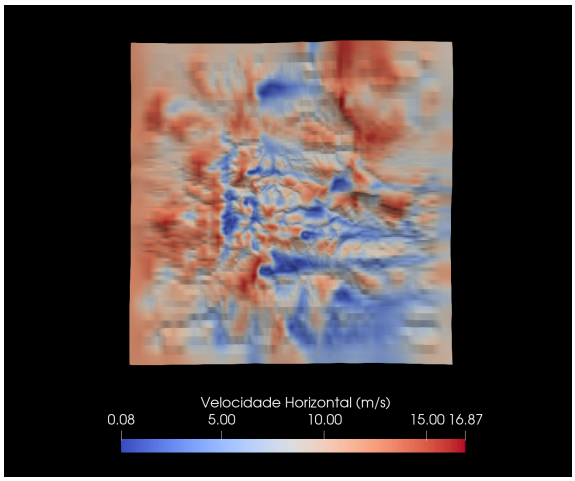
Tabela 3.6 – Configurações do conjunto final de simulações com condições de fronteira idealizadas

Direção (°)	`nig`	`njg`	`nz`	`cofx`	`cofy`	`R`	`npassfilter`	`alphalayer`
0	100	100	80	150	150	100	10	3000
30	100	100	80	150	150	100	10	3000
60	100	100	80	150	150	100	10	3000
90	256	256	100	50	50	100	10	3000
120	256	256	100	50	50	100	10	3000
150	256	256	100	50	50	100	10	3000
180	256	256	100	50	50	100	10	3000
210	256	256	100	50	50	100	10	3000
240	100	100	80	150	150	100	10	3000
270	256	256	100	50	50	100	10	3000
300	256	256	100	50	50	100	10	3000
330	256	256	100	50	50	100	10	3000

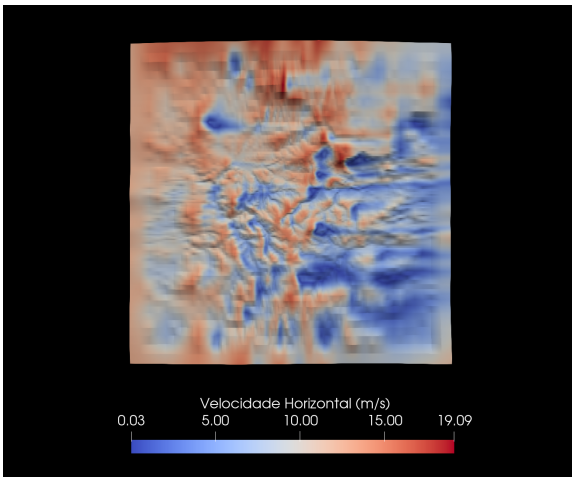
Na realização de simulações CFD com uma malha `nig` = `njg` = 256 e `nz` = 100, as direções 0°, 30°, 60° e 240° revelaram problemas de convergência. Primeiramente, foi aumentada a suavização do terreno nas extremidades aumentando o parâmetro `npassfilter`, bem como a espessura dessa camada onde é aplicada a suavização, aumentando o parâmetro `alphalayer`. Os problemas de convergência persistiram e por essa razão, a resolução da malha de cálculo para essas direções foi diminuída até se atingir convergência.

3.3.2 Modelo numérico

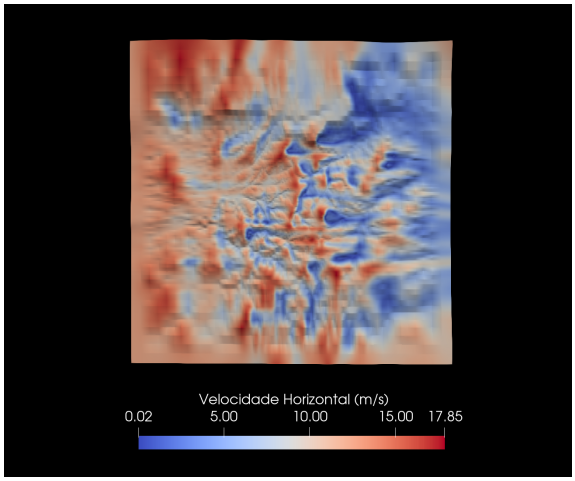
Na figura 3.8, estão representados os resultados do modelo numérico, numa superfície que acompanha o terreno a 102.5 m acima do nível do solo, a altura do eixo das turbinas eólicas.



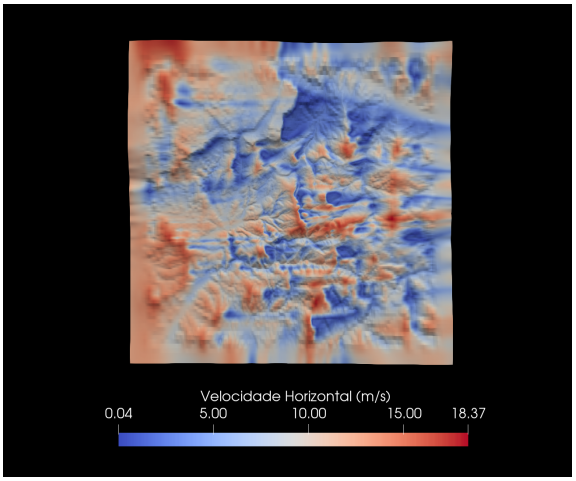
(a) 0°



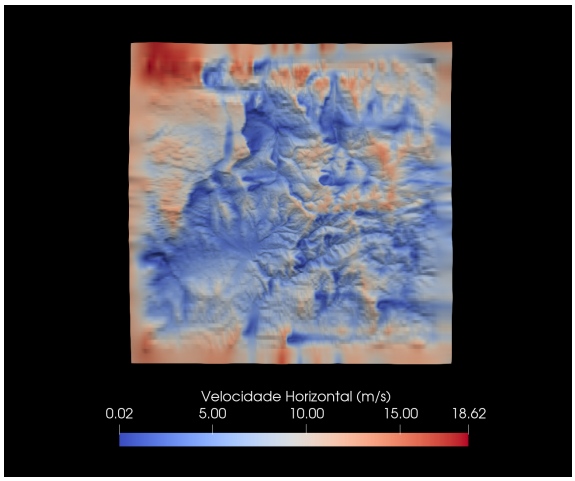
(b) 30°



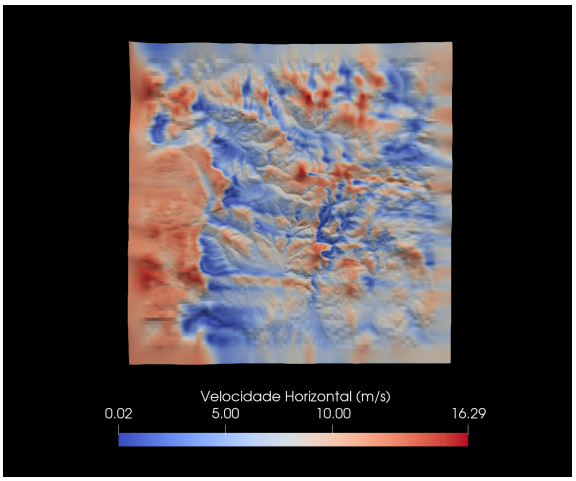
(c) 60°



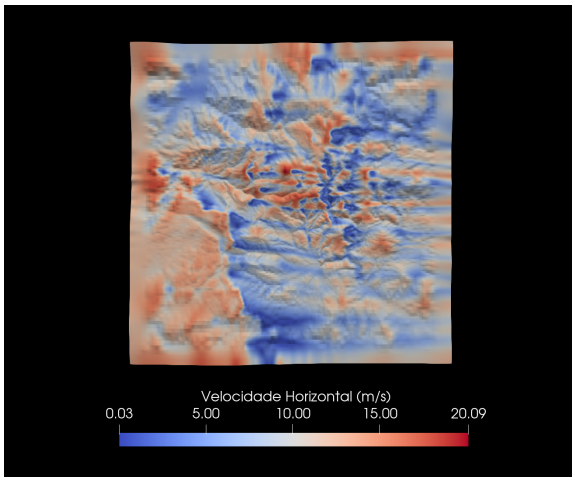
(d) 90°



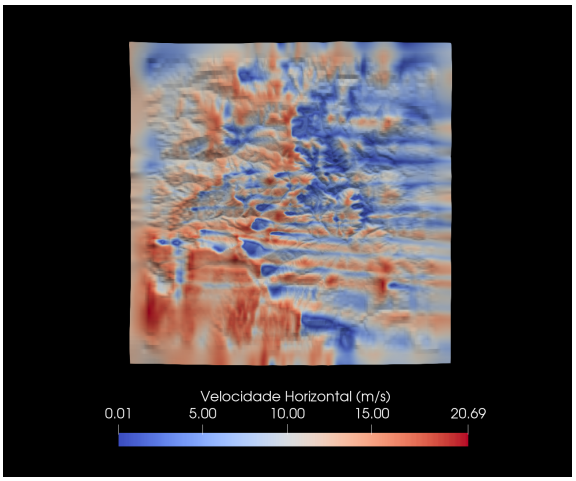
(e) 120°



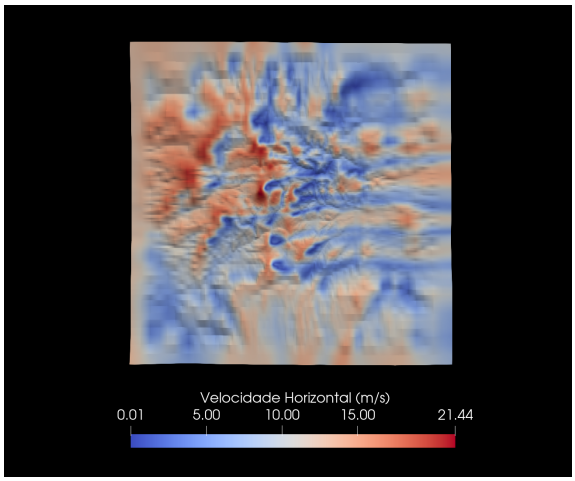
(f) 150°



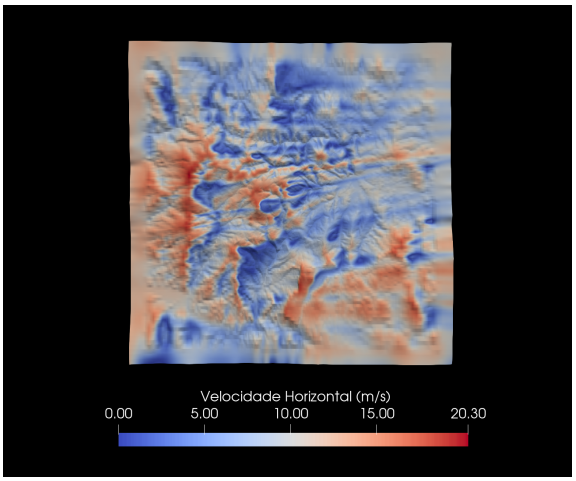
(g) 180°



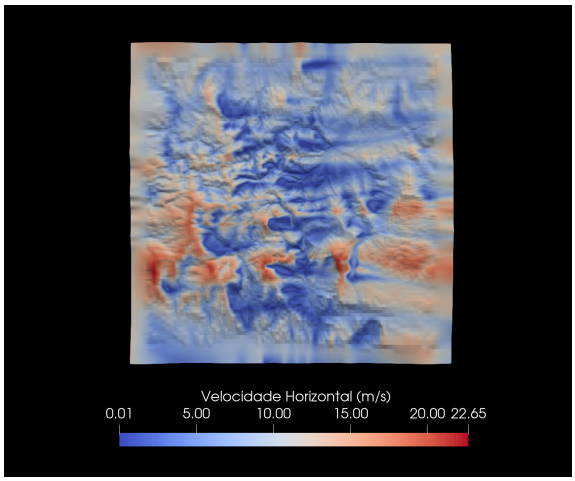
(h) 210°



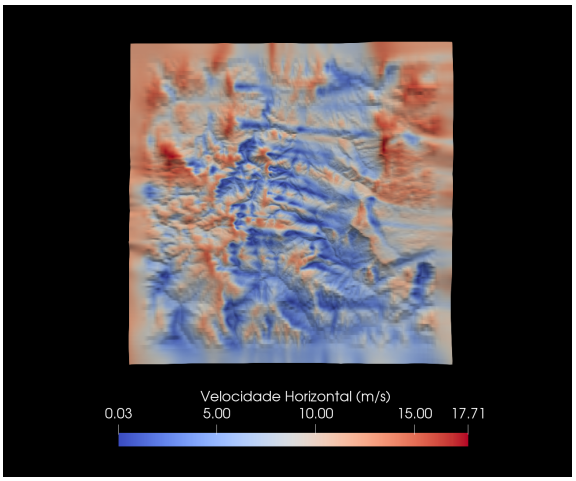
(i) 240°



(j) 270°



(k) 300°



(l) 330°

Figura 3.8 – Magnitude de velocidade horizontal para as 12 direções de vento simuladas

As superfícies ANS (acima do nível do solo) representadas na figura 3.8 foram geradas com o código `write_sample.f90`, desenvolvido em [6], que contém várias utilidades para auxiliar o pós-processamento de resultados de simulações de escoamentos atmosféricos em OpenFOAM. Uma delas é a geração de uma superfície VTK

que acompanha a topografia à altura ANS definida pelo utilizador. Esse ficheiro é depois carregado no *software Paraview* [44], que foi usado neste trabalho para a visualização gráfica dos resultados obtidos pelo modelo numérico.

3.4 Pós-processamento

Foram desenvolvidos, neste trabalho, códigos em *Python* que trabalham com um conjunto de simulações CFD e com uma série temporal de dados medidos para produzir resultados nas localizações das turbinas ou em quaisquer outras dentro do domínio em estudo. A metodologia consiste no cálculo de matrizes de transferência para cada um dos pontos em estudo. Posteriormente, é efetuado o transporte da série temporal medida para esses pontos, mediante os resultados do modelo numérico.

As matrizes de transferência contêm valores da razão entre a velocidade horizontal na turbina e a velocidade horizontal no mastro, obtidos através dos resultados do modelo numérico para 12 direções de vento. Estes dados são depois interpolados para gerar matrizes de transferência, que contêm 360 valores dessa razão correspondentes a cada direção de vento. O mesmo raciocínio foi adotado para obter a direção do vento nas turbinas, mas desta vez através do cálculo de um parâmetro $\Delta\phi$, que representa a diferença entre a direção do vento na turbina e a direção do vento no mastro.

Os códigos desenvolvidos foram estruturados de forma a otimizar o tempo aplicado em processamento, sendo os códigos e as suas funções no algoritmo de pós-processamento aqui descritas:

- **Parte 1** - `OFdata_read.py` - Leitura da malha de cálculo e resultados com recurso ao módulo `fluidfoam` [45]. Este código termina com a escrita desta informação em matrizes no formato binário. A necessidade de separar a leitura da malha de cálculo provém do facto de o `fluidfoam` ter revelado elevadas necessidades computacionais na leitura de malhas de elevada resolução.
- **Parte 2** - `CFDdata_compute.py` - Leitura da informação produzida pelo *script* anterior e transformação das coordenadas locais do OpenFOAM para UTM (*Universal Transverse Mercator*), execução de interpolações trilineares do modelo numérico para as coordenadas dos pontos em estudo e geração das matrizes de transferência.
- **Parte 3** - `resource_assessment.py` - Transporte dos dados das séries medidas para os pontos em estudo, usando as matrizes de transferência calculadas no *script* anterior, filtragem dos resultados transportados e geração dos gráficos polares que se apresentam, por exemplo, nas figuras 3.9 e 3.10.

A fragmentação do código desenvolvido permite que cada uma das partes descritas seja executada tantas vezes quanto necessárias, sem a necessidade de voltar a efetuar

todas as partes, uma vez que cada uma das partes do algoritmo pode demorar algum tempo de processamento. O código pode ser consultado no Apêndice B.

3.4.1 Validação do modelo

Para a validação do modelo, foi efetuado o transporte dos resultados medidos num ponto para outros pontos onde também existem dados medidos, permitindo assim a comparação entre o que foi medido pelos anemómetros e o que foi transportado para as suas localizações.

Como existem resultados de dois mastros de medição que registaram dados a diferentes alturas, é possível avaliar a capacidade do modelo capturar o perfil de velocidades vertical, efetuando o transporte para outras alturas ANS onde se dispõe de dados medidos com o objetivo de comparar as medições com os dados transportados.

Na tabelas seguintes é comparada a média global das medições às alturas ANS onde se dispõe de dados medidos com a média global dos resultados transportados, usando como referência o mastro 1 a diferentes alturas ANS:

Tabela 3.7 – Validações na vertical usando o mastro 1 a 81 m ANS como referência

Altura ANS [m]		Velocidade [m/s]		Desvio Percentual
Origem	Destino	Medições	Simulações	
81 (Mastro 1)	76 (Mastro 1)	7.0748	7.1815	+1.4858 %
81 (Mastro 1)	55 (Mastro 1)	7.0301	6.8936	−1.9801 %
81 (Mastro 1)	35 (Mastro 1)	6.1450	6.4526	+4.7671 %

Tabela 3.8 – Validações na vertical usando o mastro 1 a 76 m ANS como referência

Altura ANS [m]		Velocidade [m/s]		Desvio Percentual
Origem	Destino	Medições	Simulações	
76 (Mastro 1)	81 (Mastro 1)	7.2494	7.1369	−1.5763 %
76 (Mastro 1)	55 (Mastro 1)	7.0301	6.8169	−3.1275 %
76 (Mastro 1)	35 (Mastro 1)	6.1450	6.4300	+4.4323 %

Tabela 3.9 – Validações na vertical usando o mastro 1 a 55 m ANS como referência

Altura ANS [m]		Velocidade [m/s]		Desvio Percentual
Origem	Destino	Medições	Simulações	
55 (Mastro 1)	81 (Mastro 1)	7.2494	7.4070	+2.1277 %
55 (Mastro 1)	76 (Mastro 1)	7.0748	7.3364	+3.5658 %
55 (Mastro 1)	35 (Mastro 1)	6.1450	6.5637	+6.3790 %

Tabela 3.10 – Validações na vertical usando o mastro 1 a 35 m ANS como referência

Altura ANS [m]		Velocidade [m/s]		Desvio Percentual
Origem	Destino	Medições	Simulações	
35 (Mastro 1)	81 (Mastro 1)	7.2494	6.9283	−4.6346 %
35 (Mastro 1)	76 (Mastro 1)	7.0748	6.8635	−3.0786 %
35 (Mastro 1)	55 (Mastro 1)	7.0301	6.5767	−6.8940 %

Por análise às validações na vertical, conclui-se que o modelo teve alguma dificuldade em prever o comportamento do escoamento junto ao solo, uma vez que se verifica desvios percentuais mais elevados quando o transporte é feito para o mastro 1 a 35 m ANS ou desse anemómetro para cotas mais elevadas. A alturas ao solo mais elevadas, verifica-se que o modelo tem um melhor desempenho no transporte vertical. Como a obtenção de resultados à altura de eixo das turbinas envolve uma extrapolação vertical de 81 para 102.5 m ANS, é expectável que os erros das validações verticais usando o par de anemómetros mais alto seja mais representativo do erro no transporte para os aerogeradores.

Fazendo uso do mesmo raciocínio, pode também ser avaliada a capacidade do modelo de transportar a série medida na horizontal, transportando os resultados a alturas ANS semelhantes de uma localização onde se disponha de dados medidos no mastro 1 para outra do mastro 2 e vice-versa. Os resultados obtidos estão indicados na tabela 3.11.

Tabela 3.11 – Validações na horizontal usando medições de anemómetros das duas estações a alturas ANS semelhantes

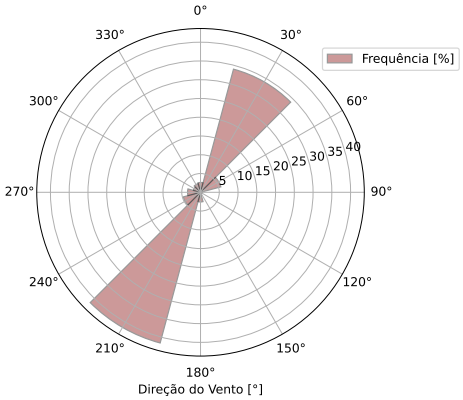
Altura ANS [m]		Velocidade [m/s]		Desvio Percentual
Origem	Destino	Medições	Simulações	
55 (Mastro 1)	60 (Mastro 2)	6.4843	5.8259	−11.3013 %
60 (Mastro 2)	55 (Mastro 1)	7.0301	6.8981	−1.9139 %

As validações horizontais indicam que o modelo tem uma dificuldade considerável em transportar os resultados na horizontal, uma vez que essas validações revelaram assimetria e foram obtidos desvios percentuais elevados do mastro 1 para o mastro 2.

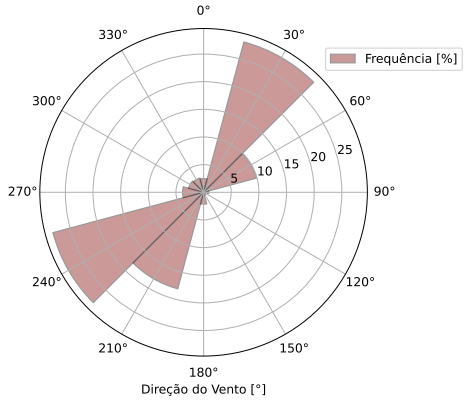
3.4.2 Frequências e velocidades de vento obtidas por direção nas turbinas

Nas figuras seguintes estão apresentados os resultados obtidos para a frequência (figura 3.9) e velocidade do vento por direção (figura 3.10), obtidos nas localizações das 5 máquinas em estudo, filtrados por intervalos de 30°. Os códigos desenvolvidos possibilitam a filtragem por intervalos diferentes, sendo o número de direções a

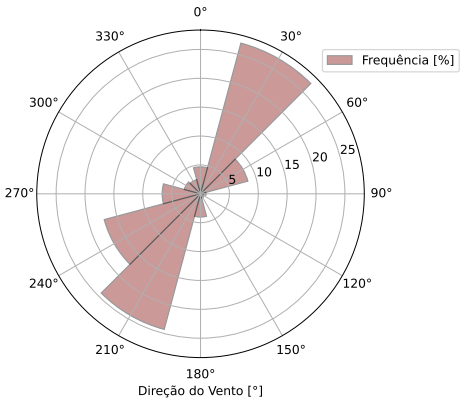
considerar no pós-processamento um dos *inputs* especificados pelo utilizador. A série temporal de dados medidos utilizada foi a do mastro 1 (ver capítulo 3.1.2).



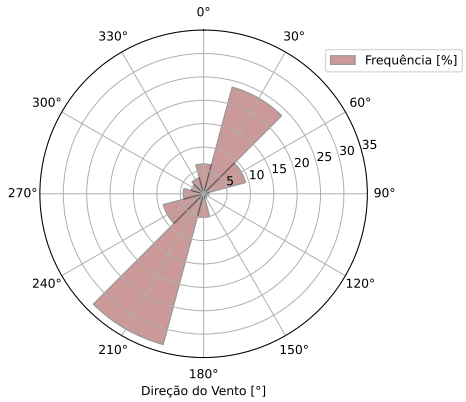
(a) Turbina VA3



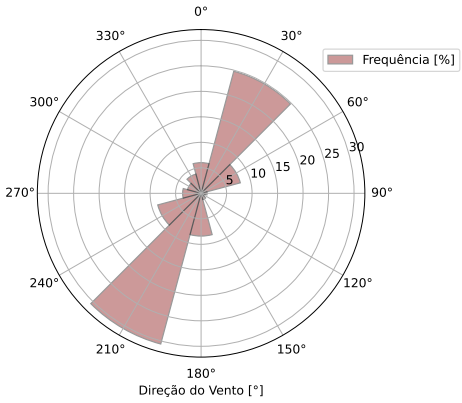
(b) Turbina VA5



(c) Turbina VA8

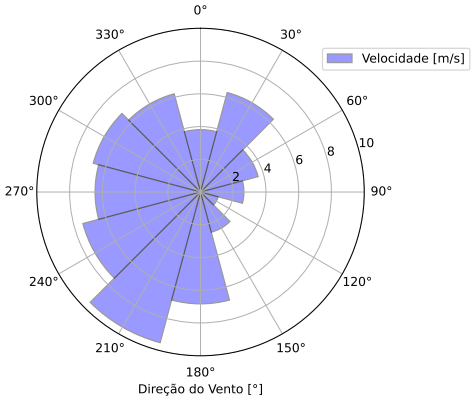


(d) Turbina VA10

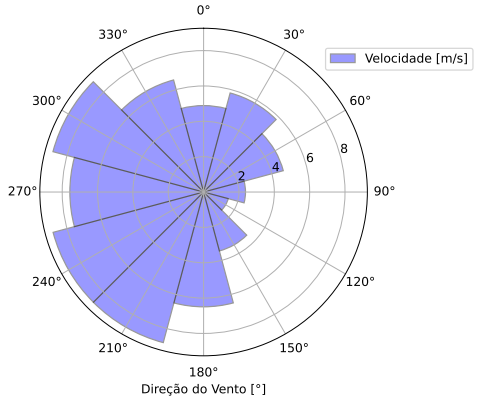


(e) Turbina VA12

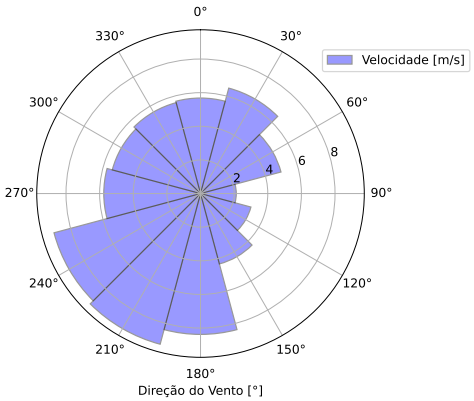
Figura 3.9 – Gráficos polares de frequência por direção obtidos nas localizações das turbinas



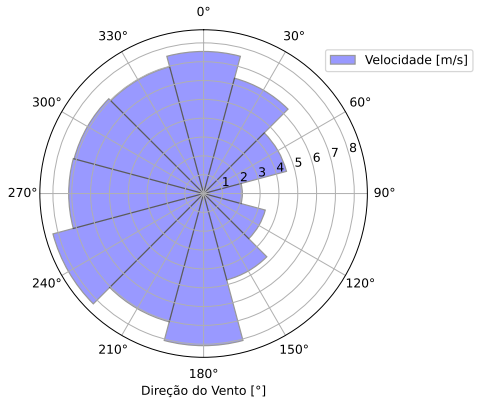
(a) Turbina VA3



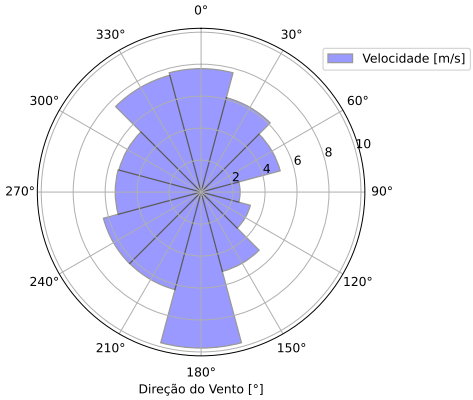
(b) Turbina VA5



(c) Turbina VA8



(d) Turbina VA10



(e) Turbina VA12

Figura 3.10 – Gráficos polares de velocidade por direção obtidos nas localizações das turbinas

3.4.3 Velocidade do vento numa grelha definida de pontos

A metodologia descrita anteriormente pode também ser adotada para efetuar o transporte da série medida para uma grelha de pontos, obtendo assim um mapa de velocidades. Para garantir uma boa resolução no mapeamento, o código deve efetuar o transporte da série medida para um número consideravelmente elevado de pontos. Este processo é computacionalmente exigente e por essa razão, foi implementada computação em paralelo na secção do código onde são efetuadas as interpolações trilineares, de forma a reduzir o tempo de cálculo, dividindo o número de pontos de cálculo pelos vários núcleos dos processadores disponíveis. Na figura 3.11 estão os resultados obtidos numa grelha com resolução de 75 m a partir de medições no mastro 1 a 81 m acima do nível do solo. Na tabela 4.12 está indicada a velocidade média da série transportada para as localizações de cada turbina.

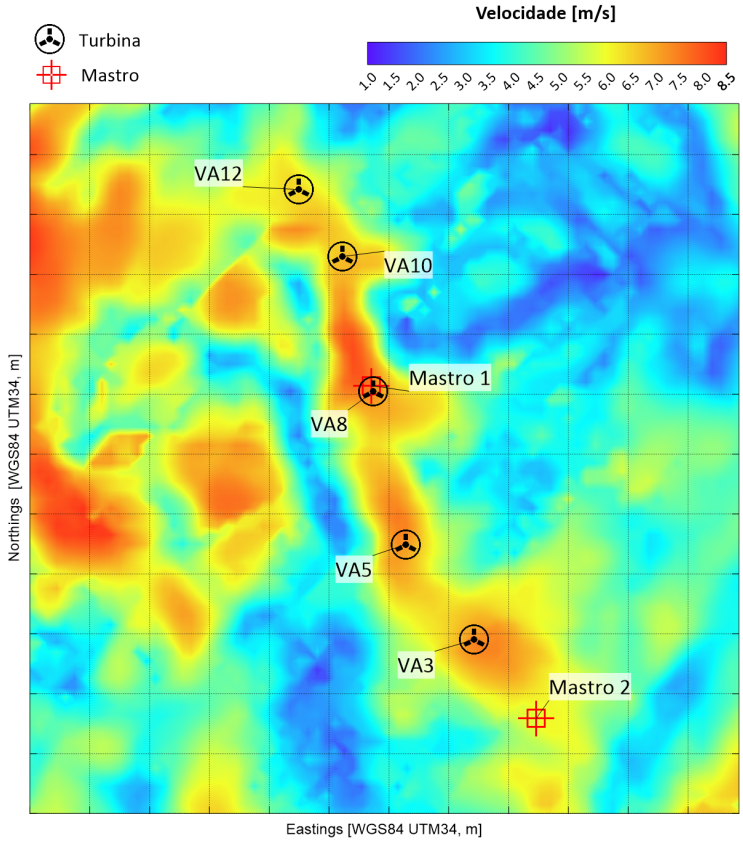


Figura 3.11 – Mapa de velocidades numa grelha de pontos (resolução de 75 m)

Tabela 3.12 – Valores da velocidade média da série transportada para cada turbina

Turbina	Altura do terreno [m]	Velocidade [m/s]
VA3	1170	7.4635
VA5	1298	7.1909
VA8	1400	7.3352
VA10	1470	6.7456
VA12	1520	6.4474

3.5 Discussão de resultados

Por análise aos resultados obtidos conclui-se que a velocidade do escoamento aumenta genericamente com a cota ANS, o que seria expectável. Nas turbinas, as direções mais relevantes são, no geral, 30° e 210°, verificando-se também ocorrência significativa da direção 240° em alguns casos, principalmente na máquina VA5, este efeito é devido à interação do escoamento com um vale que existe a montante dessa turbina (ver figuras 3.12 e 3.13).

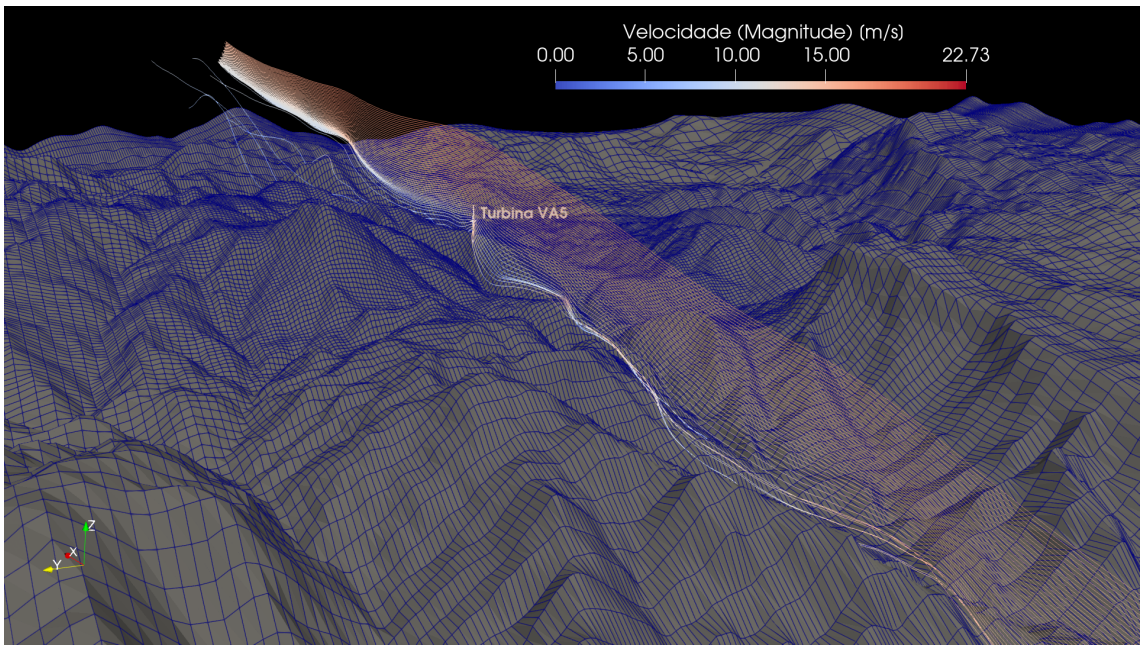


Figura 3.12 – Linhas de corrente na turbina VA5 (1)

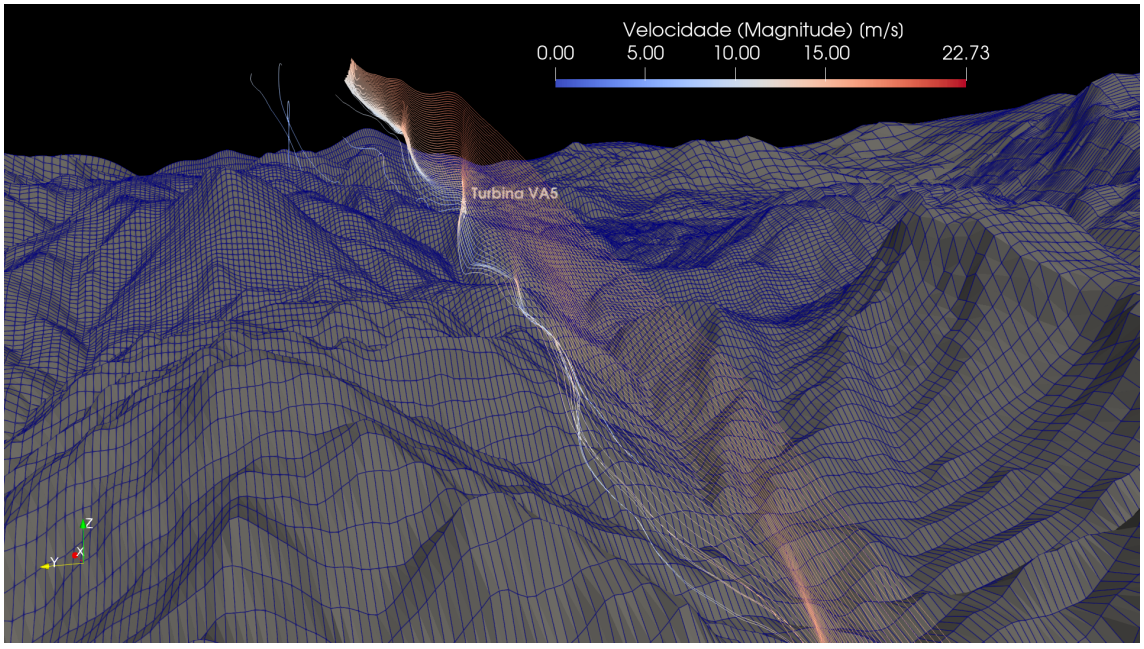


Figura 3.13 – Linhas de corrente na turbina VA5 (2)

O modelo numérico revelou capacidade em captar bolsas de recirculação e turbulência, que, como seria expectável, ocorrem nas zonas onde o terreno tem cota mais baixa a seguir a montanhas com cota elevada, seguindo a direção do escoamento (efeito de esteira a jusante provocado pelas montanhas, ver figura 3.14).

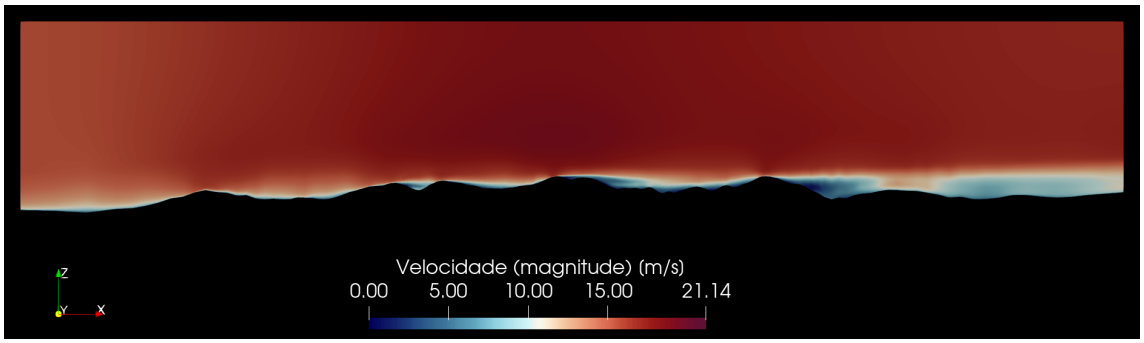


Figura 3.14 – Bolsas de recirculação a jusante das montanhas

O mapeamento da velocidade apresentou alguma correlação com a cota do terreno, mas foram obtidos resultados inesperados para as turbinas a norte. Na cumeada onde estão localizadas as turbinas, seria de esperar que a velocidade fosse superior quando a cota do terreno é mais elevada. Por análise à figura 3.11 e à tabela 3.12, verificou-se que tal não ocorreu, senão a velocidade mais elevada das 5 turbinas verificar-se-ia na turbina VA12. Por contraste, a velocidade mais elevada ocorre na turbina VA3, que é a turbina localizada a uma cota de terreno mais baixa, este resultado pode ser devido a um eventual efeito de canalização do escoamento de vento para a turbina VA3, por efeito do declive do terreno (ver figura 3.15).

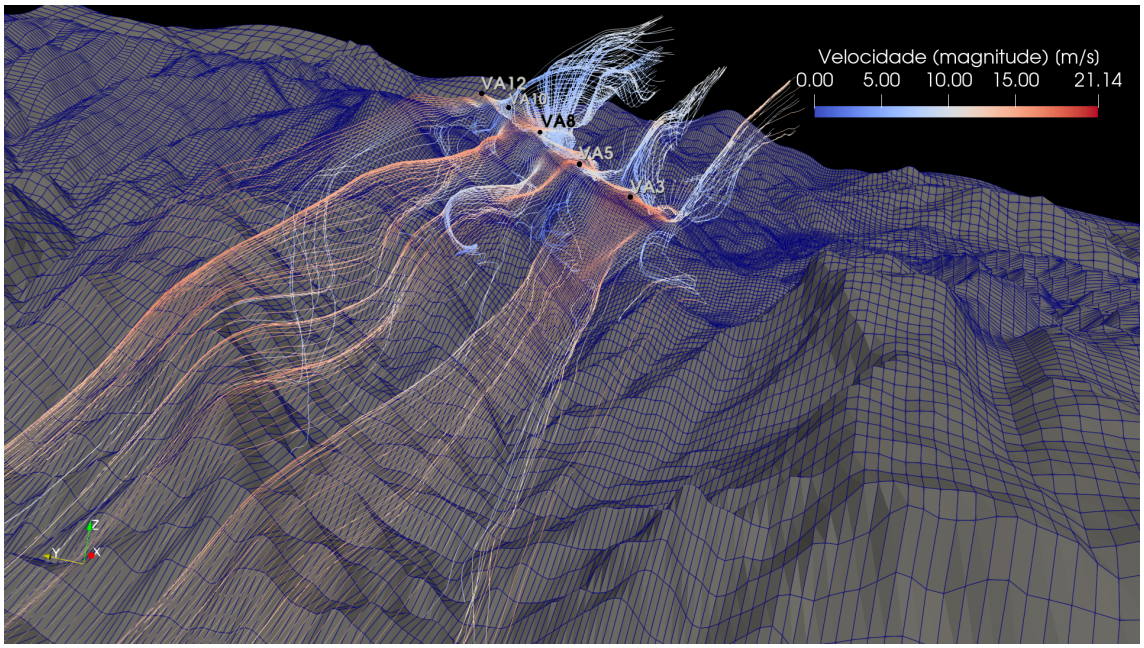


Figura 3.15 – Linhas de corrente nas turbinas

4 Simulação de escoamentos atmosféricos com condições de fronteira de modelos de mesoescala

Neste capítulo será apresentada a metodologia e os procedimentos desenvolvidos para a realização de simulações CFD de escoamentos atmosféricos neutralmente estratificados com condições de fronteira de modelos de mesoescala, aplicando-os ao mesmo caso de estudo do capítulo anterior.

4.1 Metodologia de Acoplamento

4.1.1 Simulação precursora WRF

Foi efetuada uma simulação de mesoescala usando o WRF 4.0.1 com o objetivo de obter os padrões de vento regionais. O período da simulação precursora foi de 2019 a 2021, período que coincide com o dos dados medidos pelo mastro 1. A simulação tem a duração de 3 anos completos (01/01/2019 a 31/12/2021), para evitar eventuais distorções de sazonalidade.

Na simulação, foram usadas condições de fronteira FNL que contêm dados obtidos pelo sistema global de telecomunicações e outras fontes (ver [46]). Esta base de dados contém parâmetros como a pressão à superfície, pressão ao nível do mar, altura geopotencial, temperatura, temperatura da superfície do mar, características do terreno, cobertura de gelo, humidade relativa, componentes u e v da velocidade do vento, movimento vertical, vorticidade e ozono.

As configurações adotadas no WRF são aqui brevemente listadas:

- ***nesting*** - 3 *nests* com resolução de 27 km, 9 km e 3 km.
- **Esquema de microfísica** - WSM6 [47].

- **Esquema de radiação de onda longa** - RRTM [48].
- **Esquema de radiação de onda curta** - Dudhia [49].
- **Esquema de camada limite planetária** - MYJ [50].
- **Esquema de *cumulus*** - Kain-Fritsch [51]

Neste trabalho não foi estudada a configuração ótima para a mesoescala, pelo que os esquemas listados foram selecionados com base em bons resultados que revelaram em estudos passados.

4.1.2 Considerações Gerais

No início do desenvolvimento de um novo procedimento de acoplamento OpenFOAM-WRF, foram efetuadas alterações à nomenclatura utilizada para as fronteiras no capítulo anterior, uma vez que, em simulações com acoplamento, nenhuma das fronteiras terá só entrada ou saída de fluido no domínio computacional. Neste tipo de análise, procedeu-se também à alteração da condição de simetria às fronteiras onde esta tinha sido atribuída. As alterações efetuadas estão sintetizadas na tabela 4.1:

Tabela 4.1 – Nomenclatura das fronteiras para os dois tipos de análise

Sem acoplamento		**Com acoplamento**	
Nome da Fronteira	Atributo	Nome da Fronteira	Atributo
ground	`wall`	*ground*	`wall`
inlet	`patch`	*west*	`patch`
outlet	`patch`	*east*	`patch`
front	`symmetryPlane`	*south*	`patch`
back	`symmetryPlane`	*north*	`patch`
sky	`symmetryPlane`	*sky*	`patch`

Foram implementadas alterações aos códigos FORTRAN referidos no capítulo anterior, `write_blockMeshDict.f90` e `write_z0.f90`, para fazerem uso desta nomenclatura para as simulações acopladas.

4.1.3 Abordagem Inicial

Para a realização de simulações CFD de escoamentos atmosféricos neutralmente estratificados com condições de fronteira de modelos de mesoescala, foram elaborados códigos em linguagem *Python* que recebem ficheiros no formato netCDF (*Network Common Data Form*) do modelo meteorológico WRF ou de um código que faz parte do WINDIE, o `wrfmean.f90`. Este último código compila os resultados de uma simulação WRF num período temporal definido e retorna como *output* resultados médios para cada direção de vento, também no formato netCDF.

O códigos *Python* desenvolvidos neste trabalho lêem os outputs netCDF, fazem tratamento dos dados como a conversão das coordenadas de latitude e longitude para o sistema UTM e a sua transformação para a malha no OpenFOAM, centrada no ponto $x = 0$ e $y = 0$. Seguidamente, os resultados do WRF são aplicados às cinco fronteiras do domínio de microescala.

Numa primeira abordagem pretendeu-se fazer uma simulação OpenFOAM transiente com condições de fronteira a variar no tempo retiradas do WRF. Esta abordagem consistiu na leitura dos resultados do WRF e aplicação direta dos resultados nas cinco fronteiras, pois o OpenFOAM executa interpolações trilineares aquando do uso da condição de fronteira `timeVaryingMappedFixedValue`. Esta condição de fronteira permite especificar valores mapeados de velocidade ou outras quantidades em cada uma das fronteiras, quer estejam estes com a mesma resolução da malha do OpenFOAM ou não.

O código de acoplamento desenvolvido faz a transformação das coordenadas e das componentes u e v da velocidade da mesoescala para o sistema local do OpenFOAM. Após a execução das interpolações para as fronteiras do OpenFOAM, os dados são escritos em ficheiros de texto de acordo com o formato requerido pelo OpenFOAM na pasta `constant`, de acordo com a seguinte estrutura de diretórios e ficheiros:

```
'--- constant/
   '--- boundaryData/
       '--- east/
          ' -- 0/
             -- U
             -- k
          ' -- points
       '--- north/
          ' -- 0/
             -- U
             -- k
          ' -- points
       '--- sky/
          ' -- 0/
             -- U
             -- k
          ' -- points
       '--- south/
          ' -- 0/
             -- U
             -- k
          ' -- points
       '--- west/
          ' -- 0/
             -- U
             -- k
          ' -- points
```

Os ficheiros `U` e `k` contêm valores das três componentes de velocidade e da energia cinética da turbulência, respetivamente. O ficheiro `points` contém as coordenadas x, y e z dos pontos onde estão especificados os valores dessas quantidades, respeitando a mesma ordem de escrita. Estes ficheiros são introduzidos em diretórios que devem ter o nome da fronteira correspondente.

Se a simulação for em regime transiente e existir variação das condições de fronteira ao longo do tempo, há a possibilidade de as especificar para múltiplos instantes de tempo, usando o mesmo método, mas gerando diretórios cujo nome é dado pelos instantes de tempo correspondentes e que contêm os dados associados a cada um desses instantes.

4.1.4 Problema da conservação da massa e regularização dos fluxos

No arranque de uma simulação OpenFOAM, o *software* verifica a conservação da massa através do código `adjustPhi.C` [52], assim, não foi possível o arranque de uma simulação acoplada com a primeira abordagem descrita anteriormente, devido à existência de uma diferença considerável entre fluxo total de entrada e o fluxo total de saída.

Para contornar este problema, foi necessário acrescentar ao algoritmo de acoplamento desenvolvido neste trabalho:

- A leitura da malha do OpenFOAM com recurso ao módulo `fluidfoam`, disponível para *Python*;
- A programação de uma subrotina para efetuar interpolações trilineares da malha do WRF especificamente para o centro dos volumes de controlo nas 5 fronteiras da malha do OpenFOAM, com recurso à função `griddata` do módulo `scipy` [53];
- Alterações ao código da função `readmesh` do módulo `fluidfoam`, para calcular a área de cada volume de controlo nas fronteiras;
- Cálculo dos fluxos em cada fronteira e regularização dos mesmos para respeitar a conservação da massa, como descrito de seguida.

Com os valores das componentes da velocidade especificados nos centros dos volumes de controlo e a área da superfície perpendicular associada, o código calcula o fluxo em cada volume de controlo nas fronteiras, com o propósito de obter um valor de fluxo total por fronteira e posteriormente equilibrar esses fluxos, para respeitar a conservação da massa. Os fluxos em cada uma das fronteiras são obtidos a partir de,

$$\begin{cases} \sum \dot{Q}_{\text{west}} = \sum (u_{\text{west}} \times A_{\text{west}}) \\ \sum \dot{Q}_{\text{east}} = \sum (u_{\text{east}} \times A_{\text{east}}) \\ \sum \dot{Q}_{\text{north}} = \sum (v_{\text{north}} \times A_{\text{north}}) \\ \sum \dot{Q}_{\text{south}} = \sum (v_{\text{south}} \times A_{\text{south}}) \\ \sum \dot{Q}_{\text{sky}} = \sum (w_{\text{sky}} \times A_{\text{sky}}) \ , \end{cases} \tag{4.1}$$

seguidamente, o fluxo máximo é calculado de acordo com:

$$\dot{Q}_{\max} = \max \left(\left| \sum \dot{Q}_{\text{west}} \right| , \left| \sum \dot{Q}_{\text{east}} \right| , \left| \sum \dot{Q}_{\text{north}} \right| , \left| \sum \dot{Q}_{\text{south}} \right| , \left| \sum \dot{Q}_{\text{sky}} \right| \right) . \tag{4.2}$$

Para regularizar os fluxos, foi definido um fator de correção F, que é essencialmente um fator de multiplicação que deve ser aplicado às velocidades na fronteira de maior fluxo e é calculado através da equação 4.3:

$$\begin{cases} F = \frac{-\left(-\sum \dot{Q}_{\text{east}} + \sum \dot{Q}_{\text{south}} - \sum \dot{Q}_{\text{north}} - \sum \dot{Q}_{\text{sky}} \right)}{\sum \dot{Q}_{\text{west}}}, & \text{se} \quad \dot{Q}_{\max} = \left| \sum \dot{Q}_{\text{west}} \right| \\ F = \frac{\sum \dot{Q}_{\text{west}} + \sum \dot{Q}_{\text{south}} - \sum \dot{Q}_{\text{north}} - \sum \dot{Q}_{\text{sky}}}{\sum \dot{Q}_{\text{east}}}, & \text{se} \quad \dot{Q}_{\max} = \left| \sum \dot{Q}_{\text{east}} \right| \\ F = \frac{\sum \dot{Q}_{\text{west}} + \sum \dot{Q}_{\text{south}} - \sum \dot{Q}_{\text{east}} - \sum \dot{Q}_{\text{sky}}}{\sum \dot{Q}_{\text{north}}}, & \text{se} \quad \dot{Q}_{\max} = \left| \sum \dot{Q}_{\text{north}} \right| \\ F = \frac{-\left(\sum \dot{Q}_{\text{west}} - \sum \dot{Q}_{\text{east}} - \sum \dot{Q}_{\text{north}} - \sum \dot{Q}_{\text{sky}} \right)}{\sum \dot{Q}_{\text{south}}}, & \text{se} \quad \dot{Q}_{\max} = \left| \sum \dot{Q}_{\text{south}} \right| \\ F = \frac{\left(\sum \dot{Q}_{\text{west}} - \sum \dot{Q}_{\text{east}} - \sum \dot{Q}_{\text{north}} + \sum \dot{Q}_{\text{south}} \right)}{\sum \dot{Q}_{\text{sky}}}, & \text{se} \quad \dot{Q}_{\max} = \left| \sum \dot{Q}_{\text{sky}} \right| \ . \end{cases} \tag{4.3}$$

O fator de correção é depois aplicado à fronteira de maior fluxo de acordo com a equação 4.4:

$$\begin{cases} u_{\text{west,corr}} = u_{\text{west}} \times F, & \text{se} \quad \dot{Q}_{\max} = \left| \sum \dot{Q}_{\text{west}} \right| \\ u_{\text{east,corr}} = u_{\text{east}} \times F, & \text{se} \quad \dot{Q}_{\max} = \left| \sum \dot{Q}_{\text{east}} \right| \\ v_{\text{north,corr}} = v_{\text{north}} \times F, & \text{se} \quad \dot{Q}_{\max} = \left| \sum \dot{Q}_{\text{north}} \right| \\ v_{\text{south,corr}} = v_{\text{south}} \times F, & \text{se} \quad \dot{Q}_{\max} = \left| \sum \dot{Q}_{\text{south}} \right| \\ w_{\text{sky,corr}} = w_{\text{sky}} \times F, & \text{se} \quad \dot{Q}_{\max} = \left| \sum \dot{Q}_{\text{sky}} \right| \ . \end{cases} \tag{4.4}$$

Esta metodologia permitiu equilibrar os fluxos, e praticamente anular a diferença entre o fluxo de entrada e o fluxo de saída. No entanto, mesmo assim, não foi possível arrancar uma simulação acoplada devido à pequena tolerância definida no código `adjustPhi.C`. A alteração dessa tolerância e recompilação do OpenFOAM permitiu arrancar a simulação mas problemas de convergência apareceram rapidamente.

4.1.5 Condições de fronteira

O passo seguinte para resolver o problema consistiu na alteração às condições de fronteira da pressão, que estavam até aqui definidas como gradiente nulo em todas as fronteiras.

O critério adotado consistiu em dar alguma alguma liberdade ao modelo, definindo pressão zero em toda a fronteira onde se verifica o maior fluxo, já considerando as correções aos fluxos referidas no capítulo anterior. Desta forma, conseguiu-se contornar o problema que até aqui tinha impedido as simulações de arrancar. No entanto, esta abordagem é válida apenas para simulações acopladas em estado estacionário. No caso de simulações acopladas em regime transiente, as condições de fronteira provenientes de modelos de mesoescala também variam no tempo e, consequentemente, a fronteira onde ocorre o maior fluxo também irá variar.

Para contornar esta dificuldade, recorreu-se a outra estratégia em que foram realizadas simulações estacionárias acopladas com resultados provenientes do código `wrfmean.f90`, cuja função foi a leitura dos campos obtidos de uma simulação transiente do WRF para cada instante de tempo. De acordo com a direção de vento obtida na localização do mastro 1 a 81 m ANS, o código atribui o campo de velocidades e de outras quantidades desse instante de tempo a essa direção. Posteriormente, é calculada a média dos campos obtidos mediante intervalos definidos de filtragem, determinando assim campos médios representativos das condições para cada direção.

Foi adicionado ao algoritmo de acoplamento desenvolvido neste trabalho, a escrita do ficheiro `p`, para que as condições de fronteira da pressão sejam definidas em função da fronteira onde há maior fluxo automaticamente para cada caso de estudo e para cada direção. O código de acoplamento pode ser consultado no Apêndice C.

Esta abordagem difere substancialmente da primeira, procurando-se agora simulações estacionárias, cada uma usando condições de fronteira representativas das condições médias dessa direção.

4.2 Resultados

4.2.1 Configurações do conjunto final de simulações

O conjunto final de simulações acopladas também foi realizado para 12 direções de vento e o critério adotado para a sua configuração foi que os parâmetros da malha de cálculo utilizados nas simulações do capítulo anterior fossem mantidos se possível. No entanto, para as direções 180° e 210° surgiram problemas de convergência, o que levou à diminuição da resolução da malha de cálculo para essas direções. A configuração dos parâmetros associados à malha de cálculo para as simulações com acoplamento foi a seguinte:

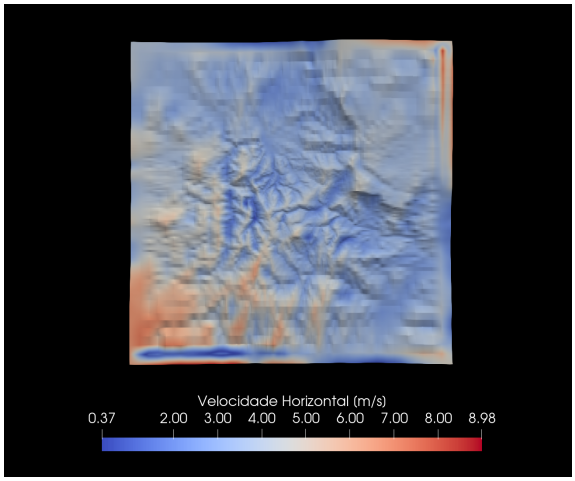
- `xmax` = `ymax` = 15000 m, para todas as direções;
- `xmin` = `ymin` = −15000 m, para todas as direções;
- `sky` = 5500 m, para todas as direções.

Tabela 4.2 – Configurações do conjunto final de simulações com acoplamento

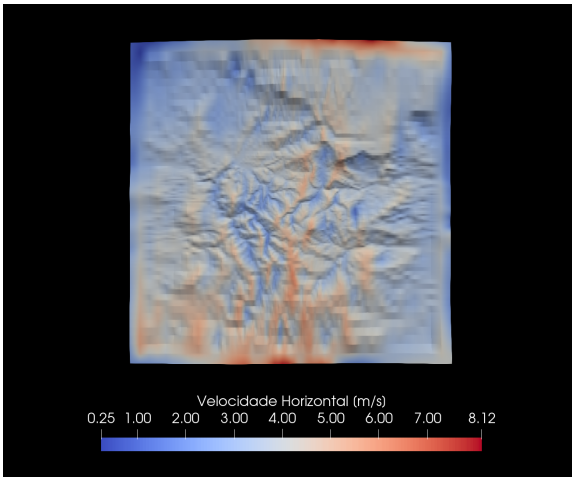
Direção (°)	`nig`	`njg`	`nz`	`cofx`	`cofy`	`R`	`npassfilter`	`alphalayer`
0	100	100	80	150	150	100	10	3000
30	100	100	80	150	150	100	10	3000
60	100	100	80	150	150	100	10	3000
90	256	256	100	50	50	100	10	3000
120	256	256	100	50	50	100	10	3000
150	256	256	100	50	50	100	10	3000
180	140	140	80	100	100	100	10	3000
210	140	140	80	100	100	100	10	3000
240	100	100	80	150	150	100	10	3000
270	256	256	100	50	50	100	10	3000
300	256	256	100	50	50	100	10	3000
330	256	256	100	50	50	100	10	3000

4.2.2 Modelo numérico

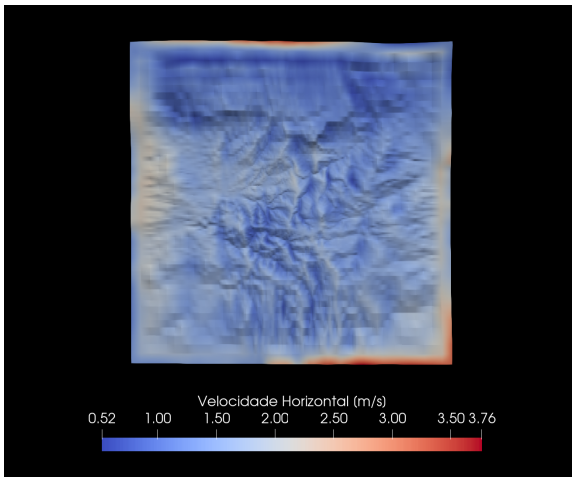
Para a visualização dos resultados do modelo numérico, foi utilizada a mesma metodologia descrita na secção 3.3.2, onde se retirou do modelo 3D uma amostra do campo de velocidades numa superfície que segue o terreno a 102.5 m.



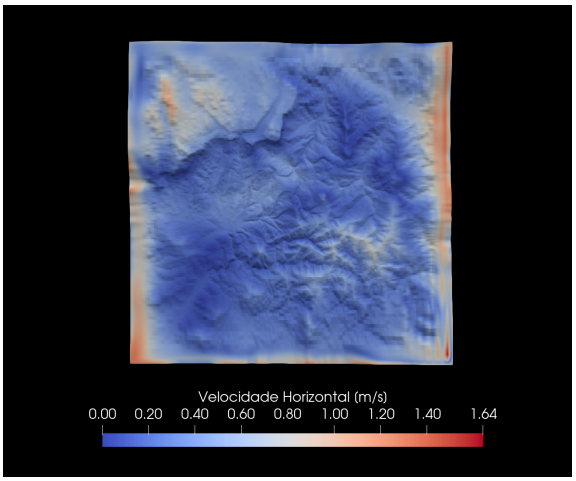
(a) 0°



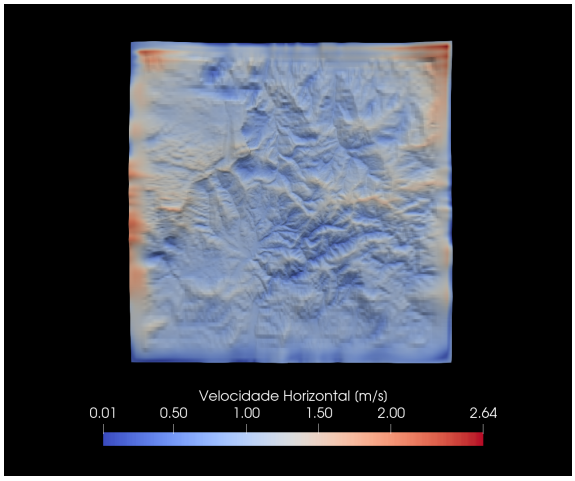
(b) 30°



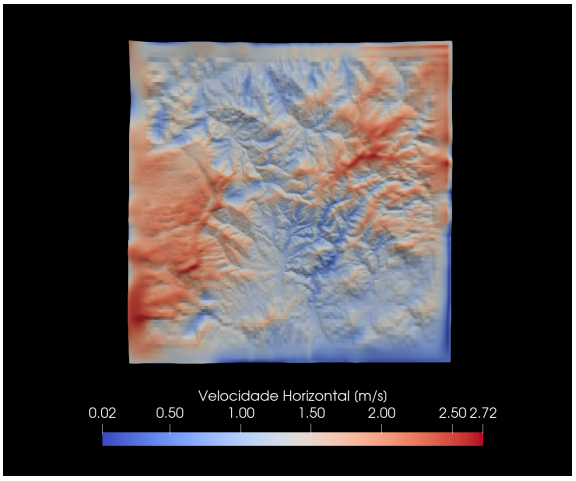
(c) 60°



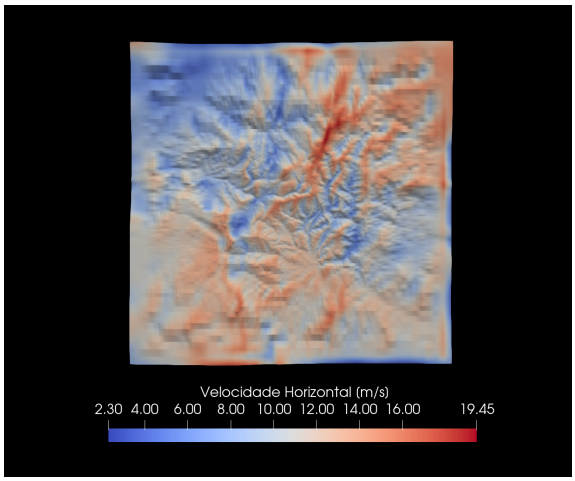
(d) 90°



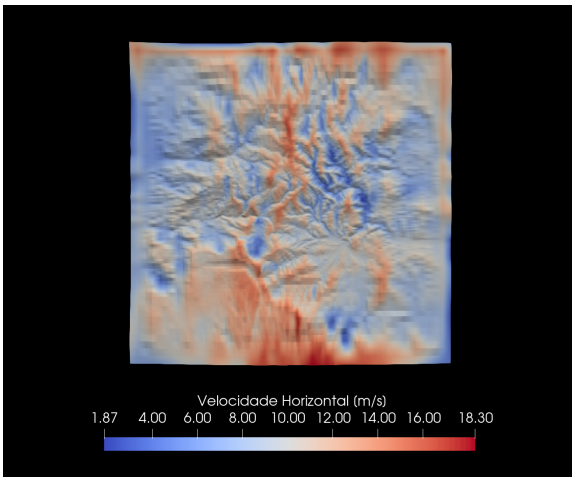
(e) 120°



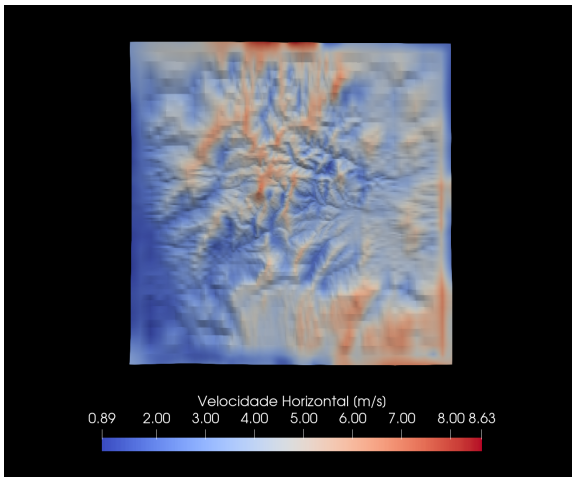
(f) 150°



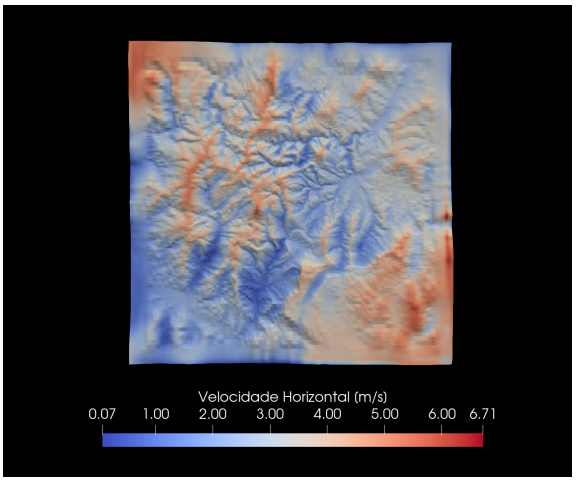
(g) 180°



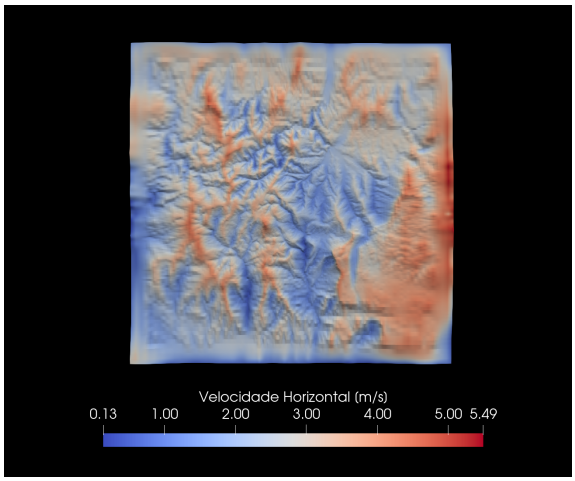
(h) 210°



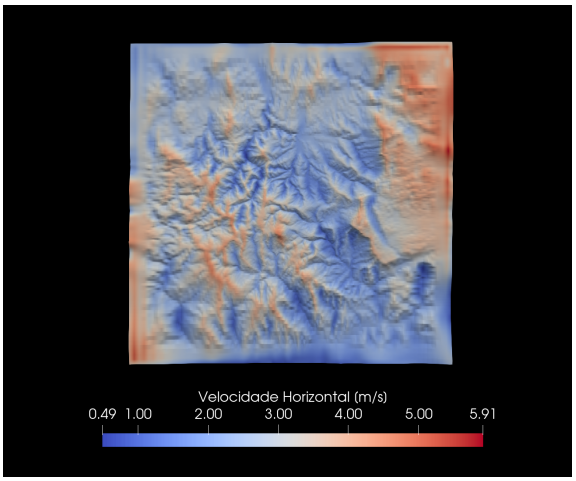
(i) 240°



(j) 270°



(k) 300°



(l) 330°

Figura 4.1 – Magnitude de velocidade horizontal para as 12 direções de vento simuladas com acoplamento

4.3 Pós-processamento

4.3.1 Validação do modelo

O procedimento para validação do modelo segue o mesmo raciocínio que foi adotado para efetuar as validações no capítulo anterior (ver 3.4.1), para avaliar a capacidade do modelo de transportar os resultados na vertical e na horizontal. Nas tabelas seguintes apresentam-se os resultados obtidos, para as simulações com acoplamento.

Tabela 4.3 – Validações na vertical usando o mastro 1 a 81 m ANS como referência

Altura ANS [m]		Velocidade [m/s]		Desvio Percentual
Origem	Destino	Medições	Simulações	
81 (Mastro 1)	76 (Mastro 1)	7.0748	7.2175	+1.9771 %
81 (Mastro 1)	55 (Mastro 1)	7.0301	7.1897	+2.2198 %
81 (Mastro 1)	35 (Mastro 1)	6.1450	7.0433	+12.7540 %

Tabela 4.4 – Validações na vertical usando o mastro 1 a 76 m ANS como referência

Altura ANS [m]		Velocidade [m/s]		Desvio Percentual
Origem	Destino	Medições	Simulações	
76 (Mastro 1)	81 (Mastro 1)	7.2494	7.1062	−2.0151 %
76 (Mastro 1)	55 (Mastro 1)	7.0301	7.0477	+0.2497 %
76 (Mastro 1)	35 (Mastro 1)	6.1450	6.9490	+11.5700 %

Tabela 4.5 – Validações na vertical usando o mastro 1 a 55 m ANS como referência

Altura ANS [m]		Velocidade [m/s]		Desvio Percentual
Origem	Destino	Medições	Simulações	
55 (Mastro 1)	81 (Mastro 1)	7.2494	7.1632	−1.2034 %
55 (Mastro 1)	76 (Mastro 1)	7.0748	7.1336	+0.8243 %
55 (Mastro 1)	35 (Mastro 1)	6.1450	6.9042	+10.9962 %

Tabela 4.6 – Validações na vertical usando o mastro 1 a 35 m ANS como referência

Altura ANS [m]		Velocidade [m/s]		Desvio Percentual
Origem	Destino	Medições	Simulações	
35 (Mastro 1)	81 (Mastro 1)	7.2494	6.3804	−13.6198 %
35 (Mastro 1)	76 (Mastro 1)	7.0748	6.3548	−11.3300 %
35 (Mastro 1)	55 (Mastro 1)	7.0301	6.2398	−12.6655 %

Com base nos resultados obtidos para as validações na vertical nas simulações com acoplamento, conclui-se que a dificuldade de o modelo prever o comportamento do escoamento junto ao solo é ainda maior relativamente ao modelo sem acoplamento. No entanto, a alturas mais elevadas, o modelo com acoplamento revelou uma melhor capacidade de transporte na vertical. Este tópico será explorado em mais detalhe no capítulo 4.4.

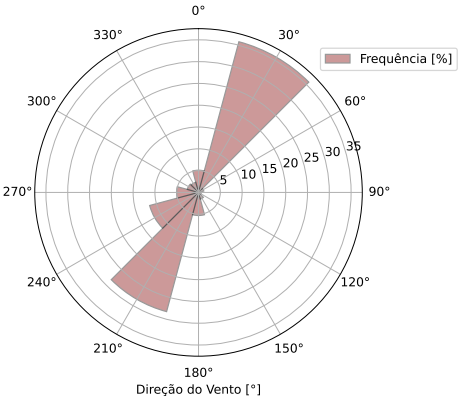
Tabela 4.7 – Validações na horizontal usando medições de anemómetros das duas estações a alturas ANS semelhantes

Altura ANS [m]		Velocidade [m/s]		Desvio Percentual
Origem	Destino	Medições	Simulações	
55 (Mastro 1)	60 (Mastro 2)	6.4843	6.6886	+3.0545 %
60 (Mastro 2)	55 (Mastro 1)	7.0301	6.9846	−0.6514 %

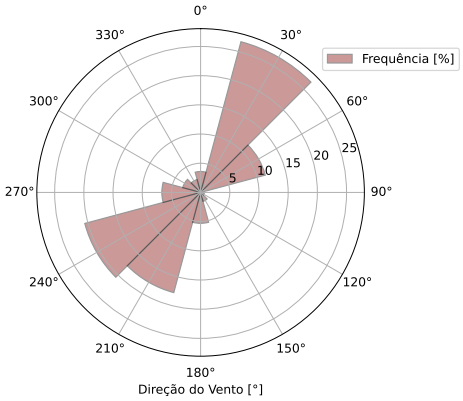
Foi na capacidade do modelo com acoplamento transportar a série medida na horizontal que se verificou uma melhoria bastante significativa em relação ao modelo sem acoplamento, com base na análise às tabelas 3.11 e 4.7. Um resumo estatístico da capacidade de transporte dos dois modelos será apresentada no capítulo 4.4.

4.3.2 Frequências e velocidades de vento obtidas por direção nas turbinas

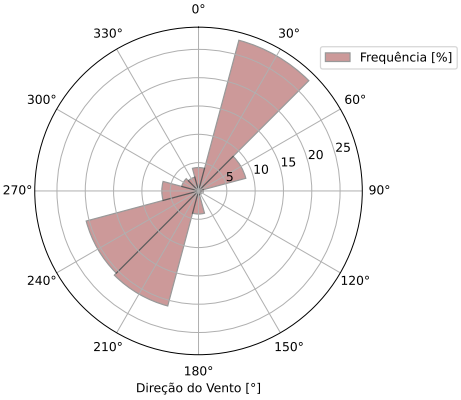
Foi adotado o mesmo processo de pós-processamento já descrito na secção 3.4 do capítulo anterior, para o transporte da série medida pelo mastro 1 para as localizações das turbinas e para uma grelha definida de pontos. Os resultados obtidos para a frequência e velocidade filtrados por direção em intervalos de 30° estão representados nas figuras 4.2 e 4.3, respetivamente.



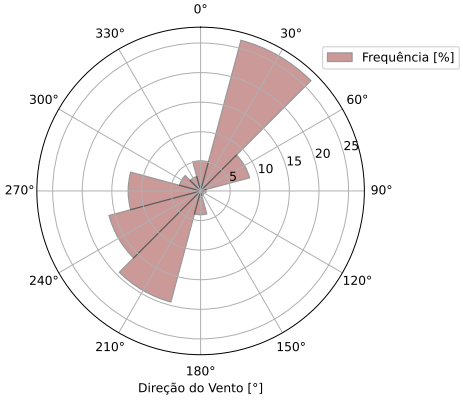
(a) Turbina VA3



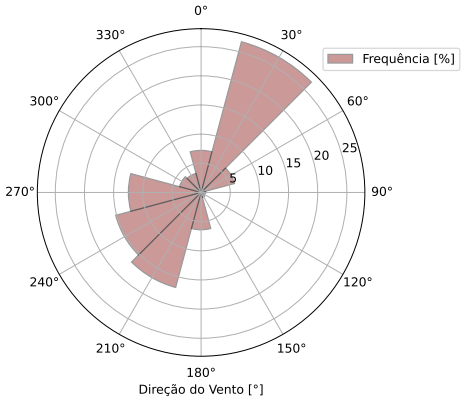
(b) Turbina VA5



(c) Turbina VA8

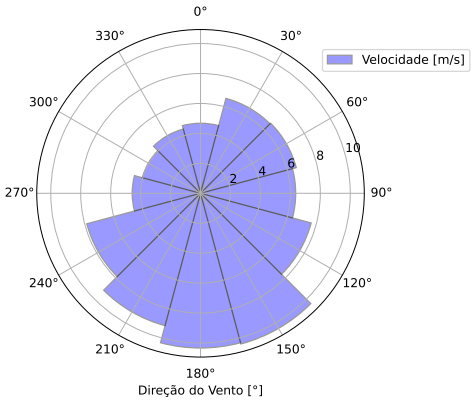


(d) Turbina VA10

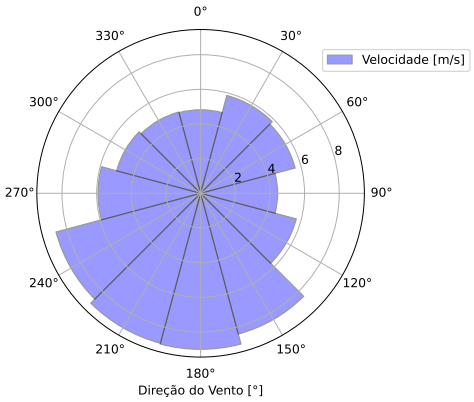


(e) Turbina VA12

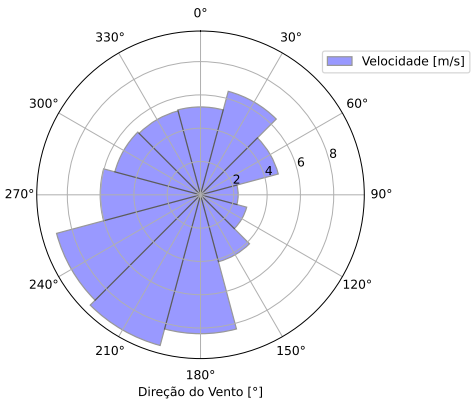
Figura 4.2 – Gráficos polares de frequência por direção obtidos nas localizações das turbinas



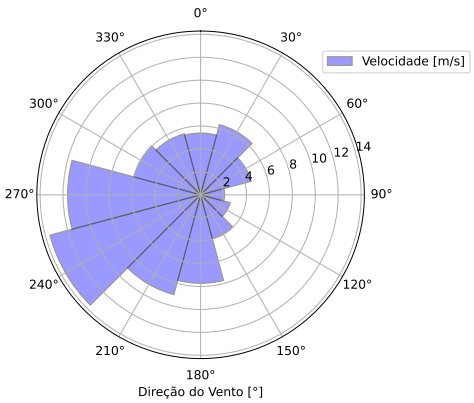
(a) Turbina VA3



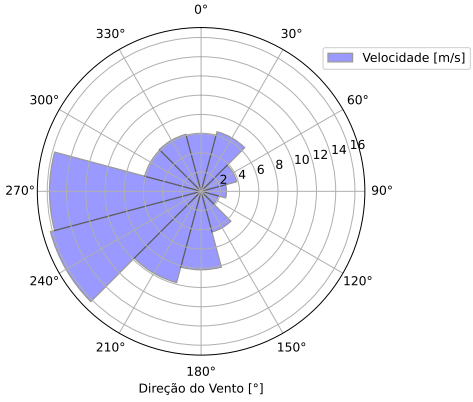
(b) Turbina VA5



(c) Turbina VA8



(d) Turbina VA10



(e) Turbina VA12

Figura 4.3 – Gráficos polares de velocidade por direção obtidos nas localizações das turbinas

4.3.3 Velocidade do vento numa grelha definida de pontos

Também foi efetuado o transporte da série medida pelo mastro 1 a 81 m ANS para uma grelha definida de pontos, utilizando as simulações acopladas (ver figura 4.4). Na tabela 4.8 está indicada a velocidade média da série transportada para cada uma das localizações das turbinas.

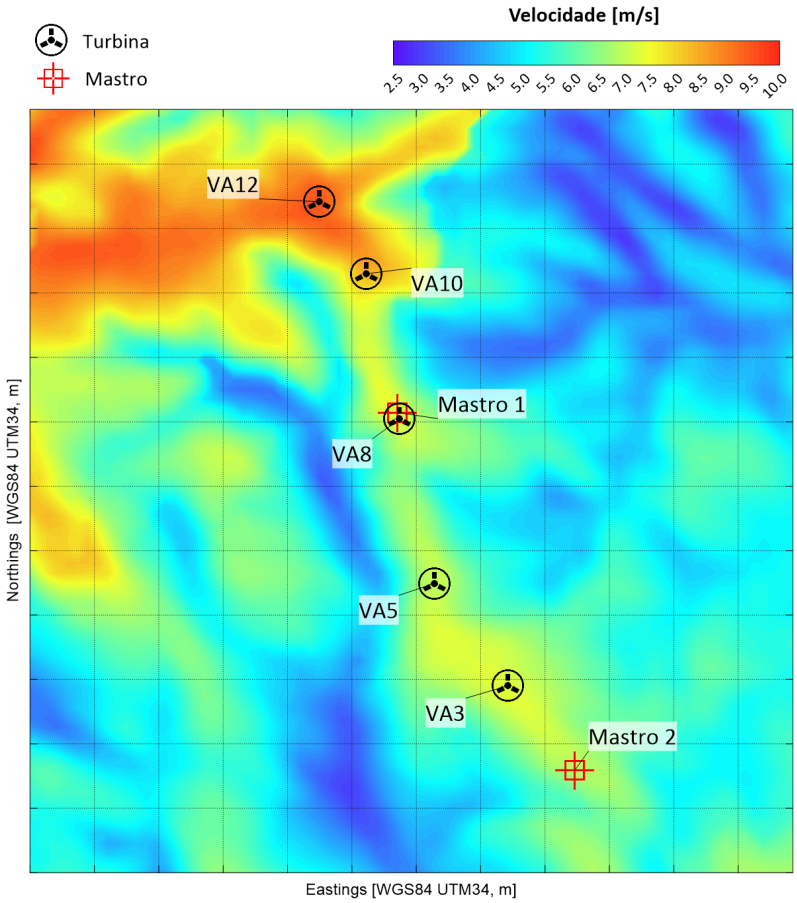


Figura 4.4 – Mapa de velocidades numa grelha de pontos (resolução de 75 m)

Tabela 4.8 – Valores da velocidade média da série transportada para cada turbina

Turbina	Altura do terreno [m]	Velocidade [m/s]
VA3	1170	7.3965
VA5	1298	7.0674
VA8	1400	7.3004
VA10	1470	8.4127
VA12	1520	9.4860

4.4 Discussão de resultados

Por análise aos resultados das validações do modelo aos dois tipos de análise neste caso de estudo, concluiu-se que os ambos os modelos revelaram alguma dificuldade em prever o perfil vertical de velocidades em alturas mais próximas do terreno. No modelo com acoplamento, verificou-se uma amplificação deste erro. A diferença observada entre os resultados medidos e os resultados das simulações em cotas ANS mais baixas pode dever-se à ausência de um modelo de floresta que, se incluído, iria permitir uma modelação mais precisa do comportamento do escoamento junto ao terreno. A diferença mais significativa entre os modelos é que o uso de acoplamento revelou uma melhor capacidade de transportar resultados medidos na horizontal, por análise às tabelas 3.11 e 4.7.

Para uma avaliação mais assertiva tendo em conta o que se pretende com o desenvolvimento destes modelos, foi efetuada uma outra análise aos valores de desvio percentual obtidos, introduzindo o conceito matemático de RMS (*Root Mean Square*). O objetivo foi comparar um valor representativo do erro global para os dois tipos de análise, tendo ou não em consideração as medições a 35 m ANS, uma vez que o erro no transporte a cotas mais elevadas é mais representativo do que se espera obter no transporte da série medida para a localização das turbinas. Também foi calculado o RMS para as validações horizontais e para os dois tipos de validações, este último também tendo ou não em conta as medições a 35 m ANS.

Tabela 4.9 – Valor RMS dos desvios percentuais das validações verticais

	Sem acoplamento	Com acoplamento
RMS (incluindo medições a 35 m ANS)	4.0536 %	8.6930 %
RMS (excluindo medições a 35 m ANS)	2.4371 %	1.5857 %

Tabela 4.10 – Valor RMS dos desvios percentuais das validações horizontais

	Sem acoplamento	Com acoplamento
RMS	8.1049 %	2.2084 %

Tabela 4.11 – Valor RMS dos desvios percentuais dos dois tipos de validações

	Sem acoplamento	Com acoplamento
RMS (incluindo medições a 35 m ANS)	4.8445 %	8.0913 %
RMS (excluindo medições a 35 m ANS)	4.5691 %	1.7622 %

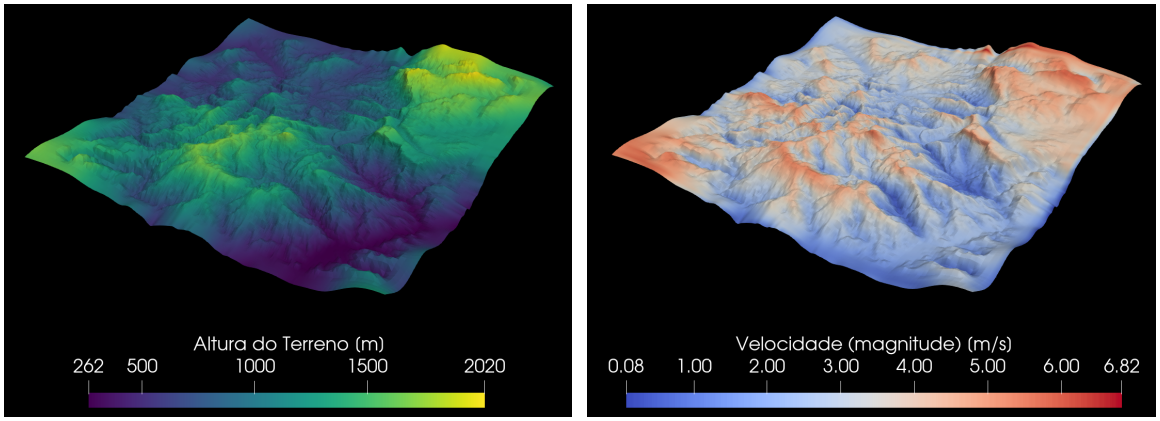
Esta análise leva à conclusão de que o modelo com acoplamento também tem melhor capacidade quando são transportados resultados da série medida na vertical em cotas mais elevadas, um aspeto importante tendo em conta que o último objetivo é transportar a série medida pelos anemómetros mais altos para a altura do eixo das turbinas.

Para facilitar a comparação entre os dois modelos, a tabela 4.12 contém a velocidade média obtida das séries transportadas para as turbinas com os dois tipos de análise:

Tabela 4.12 – Valores da velocidade média da série transportada para cada turbina e para os dois tipos de análise

Turbina	Altura do terreno [m]	Velocidade [m/s]	
		Sem acoplamento	Com acoplamento
VA3	1170	7.4635	7.3965
VA5	1298	7.1909	7.0674
VA8	1400	7.3352	7.3004
VA10	1470	6.7456	8.4127
VA12	1520	6.4474	9.4860

Por contraste ao modelo sem acoplamento, o modelo com acoplamento revela que a velocidade do vento é superior nas turbinas VA10 e VA12, o que seria expectável por estarem localizadas a cotas superiores. Conclui-se ainda que no modelo com acoplamento existe uma forte correlação entre a velocidade e a cota do terreno, como é possível visualizar nas figuras 4.5 e 4.6, retiradas dos resultados do modelo numérico.



(a) (b)

Figura 4.5 – Altura do terreno (a) e magnitude de velocidade numa superfície 102.5 m ANS (b)

No modelo com acoplamento, verificou-se velocidades significativamente baixas nas direções 90°, 120° e 150°, o que é coerente com os dados medidos mas levou a alguma incerteza da solução produzida pelo modelo numérico para essas direções. Por esse motivo, no âmbito da avaliação do recurso eólico, foi efetuada uma análise adicional que consistiu em substituir, para essas direções, os resultados da análise com acoplamento pelos resultados da análise sem acoplamento. Nessa análise mista, foram obtidos desvios percentuais superiores nas validações do modelo. Assim, optou-se pela utilização dos resultados do modelo com acoplamento para todas as direções.

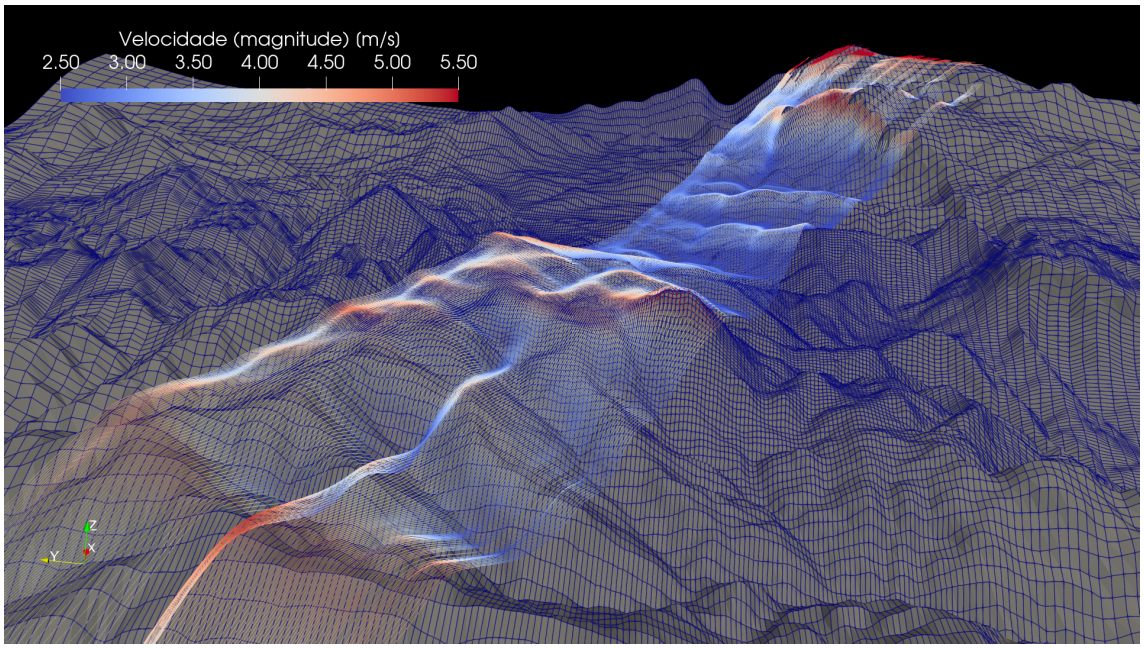


Figura 4.6 – Linhas de corrente na cumeada das turbinas

Mesmo no modelo com acoplamento verificou-se uma velocidade superior na turbina com cota de terreno mais baixa (VA3), quando comparada com as turbinas VA5 e VA8, o que reforça a conclusão já retirada no modelo sem acoplamento de que pode existir um efeito de canalização do escoamento de vento para a turbina VA3, devido à orografia do terreno.

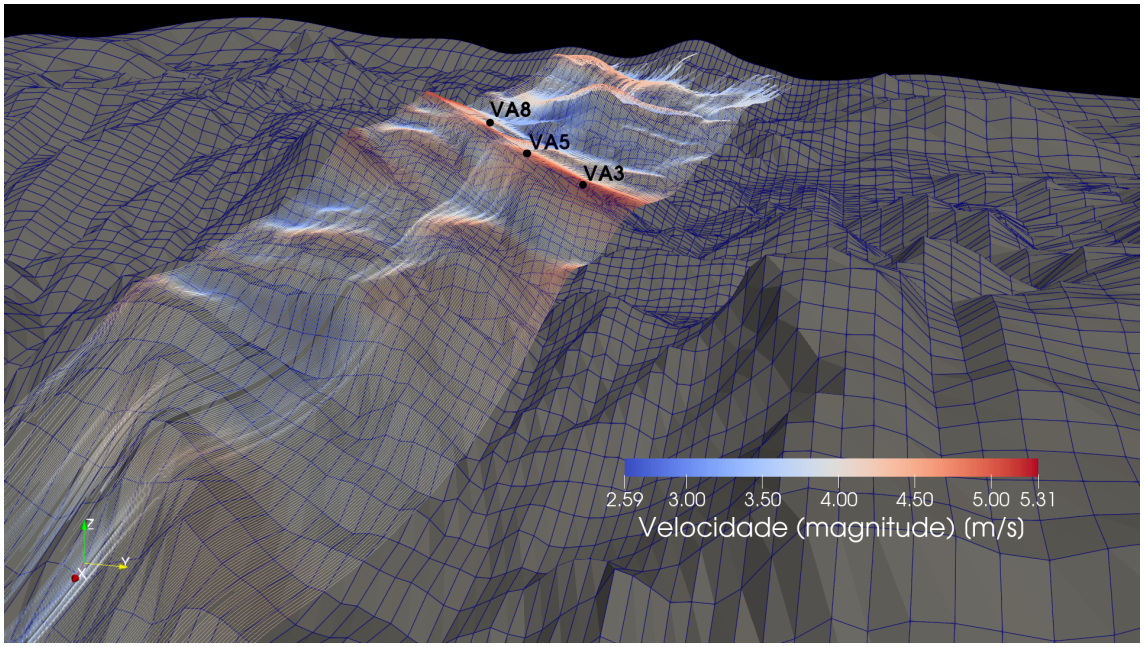


Figura 4.7 – Linhas de corrente nas turbinas VA3, VA5 e VA8

5 Conclusão

5.1 Conclusões finais

Nesta dissertação foram desenvolvidas novas metodologias para a simulação de escoamentos atmosféricos sobre terreno real com condições de fronteira de modelos de mesoescala e avaliação do recurso eólico, através do desenvolvimento de código em *Python*. Os procedimentos desenvolvidos foram implementados num caso de estudo real onde foram simulados escoamentos atmosféricos sobre terreno de complexidade muito elevada.

O modelo com acoplamento trouxe uma considerável redução no erro do transporte da série medida na horizontal com uma redução em cerca de 75 % no erro RMS das previsões cruzadas (de 8.1 % para 2.2 %). Indica-se que para efetuar as validações horizontais, foram usados períodos de medição diferentes que podem influenciar os resultados do desvio percentual obtido. Contudo, uma análise dos desvios face ao longo-termo dos períodos de medição dos mastros, sugere que a redução do erro produzido pelo acoplamento seria até amplificada, se os períodos fossem simultâneos.

Quanto à previsão do perfil vertical por parte dos modelos, ambos produziram erros significativos quando usada a altura de medição mais baixa, algo que provavelmente é devido ao facto de a floresta ter sido modelada usando apenas rugosidade. Contudo, usando os anemómetros mais altos, o acoplamento também trouxe reduções no erro dessas previsões (descida do erro RMS de 2.4 % para 1.6 %), sugerindo que o erro no transporte para a altura de eixo das turbinas terá uma redução de magnitude similar.

Observou-se que, neste caso, o uso de acoplamento teria inflacionado o valor das estimativas de produção do parque eólico e trazido uma redução na incerteza dessas estimativas. Contudo, não se pode assumir que as conclusões seriam necessariamente as mesmas noutro local. Para avaliar esse facto, teria sido necessário implementar os procedimentos de acoplamento desenvolvidos em outros casos de estudo, como

era pretendido nas fases iniciais do desenvolvimento deste trabalho.

A execução do código *Python* desenvolvido neste trabalho revelou elevada demanda computacional, especialmente na leitura das malhas de cálculo de maior resolução com recurso ao módulo `fluidfoam` e no transporte dos resultados de uma série medida para grelhas com um número significativamente elevado de pontos, mesmo com a implementação de paralelização no código. Tais factos sugerem a implementação e teste de outra metodologia para leitura das malhas de cálculo ou o desenvolvimento de código em outras linguagens de programação.

5.2 Trabalho futuro

Na presente secção, são propostos temas a desenvolver no que diz respeito à utilização do OpenFOAM para simulação de escoamentos atmosféricos sobre terreno real e avaliação de recurso eólico.

1. Incluir um modelo de floresta com o objetivo de obter um perfil de velocidades mais representativo do escoamento junto ao terreno;
2. Avaliar o desempenho dos dois modelos no transporte da turbulência medida para as turbinas e estudar alternativas ao modelo $k - \varepsilon$;
3. Estender a produção de resultados nos aerogeradores a quantidades de *site assessment* como intensidade de turbulência, *shear factor* ou inclinação vertical do escoamento;
4. No que diz respeito à avaliação do recurso eólico, estimar a produção nos aerogeradores através da análise da respetiva curva de desempenho e considerar o efeito de esteira das turbinas no OpenFOAM ou em pós-processamento;
5. Considerar os efeitos térmicos em simulações CFD de escoamentos atmosféricos sobre terreno real, com recurso aos *solvers* `buoyantBoussinesqSimpleFoam` para simulações em regime estacionário ou `buoyantBoussinesqPimpleFoam` para simulações em regime transiente e do acoplamento com resultados provenientes de uma simulação precursora de mesoescala como a temperatura do ar e fluxo de calor no terreno, permitindo assim a simulação de escoamentos atmosféricos estratificados e a modelação da instabilidade atmosférica;
6. Elaborar um modelo de acoplamento que permita a simulação de escoamentos atmosféricos em regime transiente;
7. Aplicar os procedimentos desenvolvidos neste trabalho a outros casos de estudo.

Referências Bibliográficas

[1] "Wind energy industry-standard software - WAsP," acedido a 17 de Dezembro de 2022. [Online]. Available: https://www.wasp.dk/

[2] F. A. Castro, C. S. Santos, and J. C. L. da Costa, "One-way mesoscale-microscale coupling for the simulation of atmospheric flows over complex terrain," *Wind Energy*, vol. 18, pp. 1251–1272, 7 2015. [Online]. Available: https://doi.org/10.1002/we.1758

[3] C. S. Santos, "Energias Renováveis e Ambiente (MEM) : Módulo de Energia Eólica. Apontamentos das Aulas Teóricas," Mestrado em Engenharia Mecânica, ISEP, 2022.

[4] "Weather Research & Forecasting Model (WRF) | Mesoscale & Microscale Meteorology Laboratory," acedido a 30 de Dezembro de 2022. [Online]. Available: https://www.mmm.ucar.edu/models/wrf

[5] "OpenFOAM | Free CFD Software | The OpenFOAM Foundation," acedido a 26 de Dezembro de 2022. [Online]. Available: https://openfoam.org/

[6] J. M. Azevedo, "Development of procedures For The Simulation of Atmospheric Flows Over Complex Terrain, using OpenFOAM (MSc thesis)," ISEP, Portugal, 2013.

[7] R. B. Stull, *An Introduction to Boundary Layer Metereology*. Kluwer Academic Publishers, 1998.

[8] J. Berg, M. Kelly, J. Mann, and M. Nielsen, *DTU 46100: Introduction to Micrometeorology for WindEnergy. DTU Wind Energy. DTU Wind Energy E No. E-0232*. DTU, Risø Campus/Lab, 2022.

[9] M. Z. Jacobson, *Fundamentals of Atmospheric Modeling*. Cambridge University Press, 2005.

[10] E. Leblebici, "Unsteady Atmospheric Flow Solutions With OpenFOAM Coupled With the Numerical Weather Prediction Software WRF (PhD thesis)," Middle East Technical University, Turkey, 2018.

[11] T. R. Oke, *Boundary Layer Climates, 2nd ed.* Methuen: London and New York, 1987.

[12] S. P. Arya, *Introduction to Micrometeorology Second Edition.* Academic Press, 2001.

[13] F. A. Castro, "Métodos Numéricos Para A Simulação De Escoamentos Atmosféricos Sobre Topografia Complexa (PhD thesis)," FEUP, Portugal, 1997.

[14] F. R. Martins, R. A. Guarnieri, and E. B. Pereira, "O aproveitamento da energia eólica," *Revista Brasileira de Ensino de Física*, vol. 30, 2008.

[15] G. L. Johnson, *Wind Energy Systems.* Kansas State University, 2006.

[16] I. Hussain, A. Haider, Z. Ullah, M. Russo, G. M. Casolino, and B. Azeem, "Comparative analysis of eight numerical methods using weibull distribution to estimate wind power density for coastal areas in pakistan," *Energies*, vol. 16, 2 2023.

[17] P. Jackson and J. Hunt, "Turbulent wind flow over a low hill," *Quarterly Journal of the Royal Meteorological Society*, vol. 101, pp. 929–955, 1975. [Online]. Available: https://doi.org/10.1002/qj.49710143015

[18] P. Sanderhoff, *PARK- User's Guide. A PC-program for calculation of wind turbine park performance.* Risø National Laboratory. Risø-I No. 668(EN), 1993.

[19] J. H. Ferziger and M. Perić, *Computational Methods for Fluid Dynamics, 3rd ed.* Springer, 2002. [Online]. Available: https://doi.org/10.1007/978-3-642-56026-2

[20] F. A. Castro and C. S. Santos, *Mecânica dos Fluidos Computacional.* ISEP, 2021.

[21] F. M. White, *Fluid Mechanics, 7th ed.* Mcgraw-Hill, 2011.

[22] S. B. Pope, *Turbulent Flows.* Cambridge University Press, 2000.

[23] P. Moin and K. Mahesh, "Direct numerical simulation: A tool in turbulence research," *Annual Review of Fluid Mechanics*, vol. 30, pp. 539–578, 1998.

[24] A. S. Lopes, J. M. L. M. Palma, and F. A. Castro, "Simulation of the Askervein flow. Part 2: Large-eddy simulations," *Boundary-Layer Meteorology*, vol. 125, pp. 85–108, 10 2007. [Online]. Available: https://doi.org/10.1007/s10546-007-9195-4

[25] J. Boussinesq, *Essai sur la théorie des eaux courantes, Mémoires présentés par divers savants à l'Académie des Sciences XXIII (1).* Imprimerie Nationale, 1877.

[26] F. G. Schmitt, "About Boussinesq's turbulent viscosity hypothesis: historical remarks and a direct evaluation of its validity," *Comptes Rendus - Mecanique*, vol. 335, pp. 617–627, 9 2007. [Online]. Available: https://doi.org/10.1016/j.crme.2007.08.004

[27] B. E. Launder and D. B. Spalding, "The numerical computation of turbulent flows," *Computer Methods in Applied Mechanics and Engineering*, vol. 3, pp. 269–289, 1974.

[28] B. E. Launder and B. I. Sharma, "Application of the energy-dissipation model of turbulence to the calculation of flow near a spinning disc," *Letter In Heat And Mass Transfer*, vol. 1, pp. 131–138, 1974.

[29] 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*, pp. 38:273–303, 1987.

[30] T. H. S. Shih, W. W. Liou, A. Shabbir, and Z. Yang, "A new k-ϵ eddy viscosity model for high reynolds number turbulent flows," *Computers and Fluids*, pp. 24(3):227–238, 1995.

[31] V. Yakhot and S. A. Orszag, "Renormalization Group Analysis of Turbulence. I - Basic Theory," *Journal of Scientific Computing*, pp. 1:3–51, 1986.

[32] D. Wilcox, "Reassessment of the scale-determining equation for advanced turbulence models," *AIAA Journal*, vol. 26, pp. 1299–1310, 11 1988. [Online]. Available: https://doi.org/10.2514/3.10041

[33] F. R. Menter, "Zonal two equation k-ω turbulence models for aerodynamic flows," *AIAA Fluid Dynamics Conference*, 1993. [Online]. Available: https://doi.org/10.2514/6.1993-2906

[34] "Meteodyn | numerical wind engineering – software and consultancy," acedido a 29 de Agosto de 2023. [Online]. Available: https://meteodyn.com/

[35] R. Laprise, "The Euler equations of motion with hydrostatic pressure as an independent variable," *Monthly Weather Review*, vol. 120, pp. 197–207, 1992. [Online]. Available: https://doi.org/10.1175/1520-0493(1992)120<0197:teeomw>2.0.co;2

[36] 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*. National Center for Atmospheric Research, Mesoscale and Microscale Meteorology Laboratory, 2019.

[37] O. Temel, L. Bricteux, and J. van Beeck, "Coupled wrf-openfoam study of wind flow over complex terrain," *Journal of Wind Engineering and*

Industrial Aerodynamics, vol. 174, pp. 152–169, 3 2018. [Online]. Available: https://doi.org/10.1016/j.jweia.2018.01.002

[38] "Datasets | ECMWF," acedido a 10 de Agosto de 2023. [Online]. Available: https://www.ecmwf.int/en/forecasts/datasets

[39] F. A. Castro, C. S. Santos, and J. C. L. da Costa, "Development of a meso-microscale coupling procedure for site assessment in complex terrain," *EWEA - European Wind Energy Association*, 2010. [Online]. Available: http://hdl.handle.net/10400.22/4255

[40] "OpenFOAM: API Guide: src/TurbulenceModels/turbulenceModels/-derivedFvPatchFields/wallFunctions/kqRWallFunctions/kqRWallFuncti-on/kqRWallFunctionFvPatchField.H Source File," acedido a 17 de Maio de 2023. [Online]. Available: https://www.openfoam.com/documentation/guides/latest/api/kqRWallFunctionFvPatchField_8H_source.html

[41] F. A. Castro, J. M. L. M. Palma, and A. S. Lopes, "Simulation of the Askervein Flow. Part 1: Reynolds Averaged Navier Stokes Equations (k-ϵ Turbulence Model)," *Boundary-Layer Meteorology*, pp. 501–530, 2003. [Online]. Available: https://doi.org/10.1023/A:1022818327584

[42] D. Hargreaves and N. Wright, "On the use of the k– model in commercial CFD software to model the neutral atmospheric boundary layer," *Journal of Wind Engineering and Industrial Aerodynamics*, vol. 95, no. 5, pp. 355–369, 2007. [Online]. Available: https://doi.org/10.1016/j.jweia.2006.08.002

[43] "OpenFOAM: User Guide: atmNutkWallFunction," acedido a 27 de Dezembro de 2022. [Online]. Available: https://www.openfoam.com/documentation/guides/v2112/doc/guide-bcs-wall-turbulence-atmNutkWallFunction.html

[44] "ParaView - Open-source, multi-platform data analysis and visualization application," acedido a 06 de Junho de 2023. [Online]. Available: https://www.paraview.org/

[45] "fluidfoam.readof — fluidfoam 0.2.5 documentation," acedido a 10 de Janeiro de 2023. [Online]. Available: https://fluidfoam.readthedocs.io/en/latest/generated/fluidfoam.readof.html

[46] National Centers for Environmental Prediction, National Weather Service, NOAA, U.S. Department of Commerce, "NCEP FNL Operational Model Global Tropospheric Analyses, continuing from July 1999," Boulder CO, 2000. [Online]. Available: https://doi.org/10.5065/D6M043C6

[47] S. Y. Hong, "Hongandlim-JKMS-2006," *Journal of the Korean Meteorological Society*, vol. 42, pp. 129–151, 2006.

[48] E. J. Mlawer, S. J. Taubman, P. D. Brown, M. J. Iacono, and S. A. Clough, "Radiative transfer for inhomogeneous atmospheres: RRTM, a validated correlated-k model for the longwave," *Journal of Geophysical Research Atmospheres*, vol. 102, pp. 16 663–16 682, 7 1997. [Online]. Available: https://doi.org/10.1029/97jd00237

[49] J. Dudhia, "Numerical study of convection observed during the winter monsoon experiment using a mesoscale two-dimensional model," *Journal of the Atmospheric Sciences*, vol. 46, pp. 3077–3107, 1989. [Online]. Available: https://doi.org/10.1175/1520-0469(1989)046<3077:NSOCOD>2.0.CO;2

[50] Z. I. Janjić, "The Step-Mountain Eta Coordinate Model: Further Developments of the Convection, Viscous Sublayer, and Turbulence Closure Schemes," *Monthly Weather Review*, pp. 122:927–945, 1994. [Online]. Available: https://doi.org/10.1175/1520-0493(1994)122<0927:TSMECM>2.0.CO;2

[51] J. S. Kain and J. M. Fritsch, "Convective Parameterization for Mesoscale Models: The Kain-Fritsch Scheme," *American Meteorological Society*, pp. 165–170, 1993. [Online]. Available: https://doi.org/10.1007/978-1-935704-13-3_16

[52] "OpenFOAM: API Guide: src/finiteVolume/cfdTools/general/adjustPhi/adjustPhi.C Source File," acedido a 10 de Maio de 2023. [Online]. Available: https://www.openfoam.com/documentation/guides/latest/api/adjustPhi_8C_source.html

[53] "scipy.interpolate.griddata — SciPy v1.11.2 Manual," acedido a 8 de Março de 2023. [Online]. Available: https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.griddata.html

A Códigos *Python* para processamento dos dados medidos

A.1 WindData_Read.py

```
#############################################################
## ------------------------------------------------------- ##
## -- WindData_Read.py python script                     ##
## ------------------------------------------------------- ##
#############################################################

import numpy as np
import matplotlib.pyplot as plt
from statistics import mean
import pandas as pd

#############################################################
###                    FUNCTIONS                          ###
#############################################################

def StringParser(string, file_path):
    with open(file_path,'r') as f:
        for line in f.readlines():
            if string in line:
                return line.strip().split()[2]

def StringParser_equal_sep(string, file_path):
    with open(file_path,'r') as f:
        for line in f.readlines():
            if string in line:
                return line.strip().split('= ')[1]

#############################################################
#############################################################

file_name = StringParser("file_name ","../WindData_Read.cfg")

invalid_vel_data = float(StringParser("invalid_vel_data ","../WindData_Read.cfg"))
invalid_dir_data = int(StringParser("invalid_dir_data ","../WindData_Read.cfg"))

mast_name = StringParser_equal_sep("mast_name","../WindData_Read.cfg")
nDirs = int(StringParser("nDirs ","../WindData_Read.cfg"))
skip = int(StringParser("skip =","../WindData_Read.cfg"))

PATH = "../WindData_Read.inputs/%s" %file_name

#############################################################
#############################################################
### READ MAST DATA

mast_content = [i.strip().split(",") for i in
open("%s" %PATH).readlines()]

step = int(360/nDirs)

lower = []
upper = []
```

```
53
54 ### Define filtering intervals
55 for dd in range(0,360, step):
56     low = dd - step/2
57     up = dd + step/2
58
59     if low < 0:
60         low = low + 360
61
62     lower.append(low)
63     upper.append(up)
64
65 #### Est, Dir[1], Vavg[2], Vmax, Vmin, Vstd[5], YYYYMMDDhhmm ####
66
67 dir_vals = [[] for _ in range(0, nDirs)]
68 V_vals = [[] for _ in range(0, nDirs)]
69 I_vals = [[] for _ in range(0, nDirs)]
70
71 Vmean_vals = []
72 Imean_vals = []
73 freq_vals = []
74
75 for d, low, up in zip(range(0, nDirs), lower, upper):
76     for i in range(skip, len(mast_content)):
77
78         line = mast_content[i]
79         dir = float(line[1])
80         V = float(line[2])
81         vstd = float(line[5])
82         I = vstd/V
83
84         if low > up:
85             if dir >= low or dir < up:
86                 if (V != invalid_vel_data and V != invalid_dir_data and
87                     dir != invalid_vel_data and dir != invalid_dir_data):
88                     V_vals[d].append(V)
89                     I_vals[d].append(I)
90         else:
91             if dir >= low and dir < up:
92                 if (V != invalid_vel_data and V != invalid_dir_data and
93                     dir != invalid_vel_data and dir != invalid_dir_data):
94                     V_vals[d].append(V)
95                     I_vals[d].append(I)
96
97 sumlen = 0
98 for d in range(0, nDirs):
99     sumlen = sumlen + len(V_vals[d])
100
101 for d in range(0,nDirs):
102     freq_vals.append(len(V_vals[d])/sumlen*100)
103     Vmean_vals.append(mean(V_vals[d]))
104     Imean_vals.append(mean(I_vals[d])*100)
105
106 ############################################################
107 ############################################################
108 ### OUTPUT TABLE
109
110 print("--- AVERAGE MEASURED DATA: STATION %s ---" %mast_name)
111
112 direcs = np.arange(0, 360, step)
113
114 index_str = []
115 for d in direcs:
116     index_str.append("")
117
118 data_dict = {"DIR": direcs, "freq [%]": freq_vals, "v [m/s]": Vmean_vals,
119         "I [%]": Imean_vals}
120
121 df = pd.DataFrame(data=data_dict, index=index_str)
122 df_str = df.to_string(index=False)
123 print(df)
124
125 f = open("../WindData_Read.outputs/%s_WindData_res.txt" %(mast_name), "w")
```

```
f.write(df_str)
f.close()
print("")
print("%s_WindData_res.txt saved on WindData_Read.outputs/" %mast_name)

############################################################
############################################################
### OUTPUT PLOTS

# Wind Frequency
dir_rad = []

for d in direcs:
    dir_rad.append(d*np.pi/180)
width = 2*np.pi/nDirs

ax0 = plt.subplot(111,polar=True)
ax0.bar(dir_rad,freq_vals,color ='maroon',width = width,bottom = 0,
            edgecolor = 'k', align = 'center',alpha=0.4,label='Frequência [%]')
ax0.set_xticks(dir_rad)
ax0.set_rlabel_position(75)
ax0.legend()
angle = np.radians(45)
ax0.legend(loc="lower left",
        bbox_to_anchor=(.5 + np.cos(angle)/2, .5 + np.sin(angle)/2))
ax0.set_theta_zero_location("N")
ax0.set_theta_direction(-1)
#ax0.set_title('Estação %s' %mast_name)
ax0.set_xlabel('Direção do Vento [°]')
plt.savefig("../WindData_Read.outputs/%s_freq.png" %mast_name)
print("%s_freq.png saved on WindData_Read.outputs/" %mast_name)
plt.close()

# Velocity
dir_rad = []

for d in direcs:
    dir_rad.append(d*np.pi/180)
width = 2*np.pi/nDirs

ax1 = plt.subplot(111,polar=True)
ax1.bar(dir_rad,Vmean_vals,color ='blue',width = width,bottom = 0,
            edgecolor = 'k', align = 'center',alpha=0.4,label='Velocidade [m/s]')
ax1.set_xticks(dir_rad)
ax1.set_rlabel_position(75)
ax1.legend()
angle = np.radians(45)
ax1.legend(loc="lower left",
        bbox_to_anchor=(.5 + np.cos(angle)/2, .5 + np.sin(angle)/2))
ax1.set_theta_zero_location("N")
ax1.set_theta_direction(-1)
#ax1.set_title('Estação %s' %mast_name)
ax1.set_xlabel('Direção do Vento [°]')
plt.savefig("../WindData_Read.outputs/%s_V.png" %mast_name)
print("%s_V.png saved on WindData_Read.outputs/" %mast_name)
plt.close()

# Turbulence Intensity
dir_rad = []

for d in direcs:
    dir_rad.append(d*np.pi/180)
width = 2*np.pi/nDirs

ax2 = plt.subplot(111,polar=True)
ax2.bar(dir_rad,Imean_vals,color ='blue',width = width,bottom = 0,
            edgecolor = 'k', align = 'center',alpha=0.4,label='Intensidade da
    Turbulência [%]')
ax2.set_xticks(dir_rad)
ax2.set_rlabel_position(75)
ax2.legend()
angle = np.radians(45)
ax2.legend(loc="lower left",
```

```
        bbox_to_anchor=(.5 + np.cos(angle)/2, .5 + np.sin(angle)/2))
ax2.set_theta_zero_location("N")
ax2.set_theta_direction(-1)
#ax2.set_title('Estação %s' %mast_name)
ax2.set_xlabel('Direção do Vento [°]')
plt.savefig("../WindData_Read.outputs/%s_I.png" %mast_name)
print("%s_I.png saved on WindData_Read.outputs/" %mast_name)
plt.close()
```

A.2 weibull.py

```
############################################################
## ------------------------------------------------------ ##
## -- weibull.py python script                          ##
## ------------------------------------------------------ ##
############################################################

import numpy as np
import matplotlib.pyplot as plt
from statistics import mean

############################################################
############################################################
### FUNCTIONS

def StringParser(string, file_path):
    with open(file_path,'r') as f:
        for line in f.readlines():
            if string in line:
                return line.strip().split()[2]

def StringParser_equal_sep(string, file_path):
    with open(file_path,'r') as f:
        for line in f.readlines():
            if string in line:
                return line.strip().split('= ')[1]

def weibull(u, k, A):
    return (k/A)*(u/A)**(k-1)*np.exp(-(u/A)**k)

def weibull_moments_estimation(data, frequencies):

    """
    Estimate k and c parameters of a Weibull distribution using the moments method (MM)
    from frequency data.

    Inputs:
        data (numpy array): 1-dimensional array containing the data points.
        frequencies (numpy array): 1-dimensional array containing the frequencies of
    each data point.
    Outputs:
        k : Estimated shape parameter.
        c : Estimated scale parameter.
    """

    data = np.array(data)
    frequencies = np.array(frequencies)
    n = np.sum(frequencies)
    mean = np.sum(data * frequencies) / n
    var = np.sum((data - mean) ** 2 * frequencies) / n
    std = np.sqrt(var)

    # Estimate shape parameter k:
    k = (std / mean) ** (-1.086)

    # Estimate scale parameter c:
    c = mean / np.math.gamma(1 + 1 / k)

    return k, c

############################################################
############################################################
```

```
60
61 file_name = StringParser("file_name ","../WindData_Read.cfg")
62 mast_name = StringParser_equal_sep("mast_name","../WindData_Read.cfg")
63 vmax = int(StringParser("vmax =","../WindData_Read.cfg"))
64 step = float(StringParser("step =","../WindData_Read.cfg"))
65 skip = int(StringParser("skip =","../WindData_Read.cfg"))
66
67 PATH = "../WindData_Read.inputs/%s" %file_name
68
69 #### READ MAST DATA ####
70 mast_content = [i.strip().split(",") for i in
71     open("%s" %PATH).readlines()]
72
73 Vel = [[] for _ in range(0, vmax)]
74 Vvals = []
75 freq = []
76
77 #### Est, Dir[1], Vavg[2], Vmax, Vmin, Vstd[5], YYYYMMDDhhmm, ####
78 for i in range(skip,len(mast_content)):
79     line = mast_content[i]
80     dir = float(line[1])
81     V = float(line[2])
82
83     Vvals.append(V)
84
85     for v in range(0, vmax):
86         if v < V <= v + 2*step:
87             Vel[v].append(V)
88
89 Vmean = mean(Vvals)
90 print("Global mean velocity for mast %s is :: %.4f m/s\n" %(mast_name, Vmean))
91
92 sumlen = 0
93 for v in range(0, vmax):
94     sumlen = sumlen + len(Vel[v])
95
96 vx = []
97 for v in range(0, vmax):
98     vx.append(v+1)
99     freq.append(len(Vel[v])/sumlen*100)
100
101 vWeibull = []
102 freqWeibull = []
103
104 kWeibull, AWeibull = weibull_moments_estimation(vx, freq)
105
106 print(":: Estimated k and A parameters ::")
107 print("k = %.4f" %kWeibull)
108 print("A = %.4f" %AWeibull)
109
110 for v in np.arange(0, vmax, 0.001):
111     vWeibull.append(v+1)
112     freqWeibull.append(weibull(v+1, kWeibull, AWeibull)*100)
113
114 fig, ax1 = plt.subplots(figsize=(10,7))
115 ax1.bar(vx, freq, edgecolor = "black", linewidth = 1.5,
116     color="maroon")
117 ax1.plot(vWeibull, freqWeibull, "--", color = "black",
118     label="Weibull\nc = %.2f m/s | k = %.2f" %(AWeibull, kWeibull) )
119 ax1.set_xticks(vx)
120 ax1.set_xlabel("Velocidade [m/s]")
121 ax1.set_ylabel("Frequência [%]")
122 ax1.grid(axis='y')
123 ax1.legend()
124 plt.savefig("../WindData_Read.outputs/%s_weibull.png" %mast_name)
125 print("%s_weibull.png saved on WindData_Read.outputs/" %mast_name)
126 plt.close()
```

B Códigos *Python* para estudo do recurso eólico

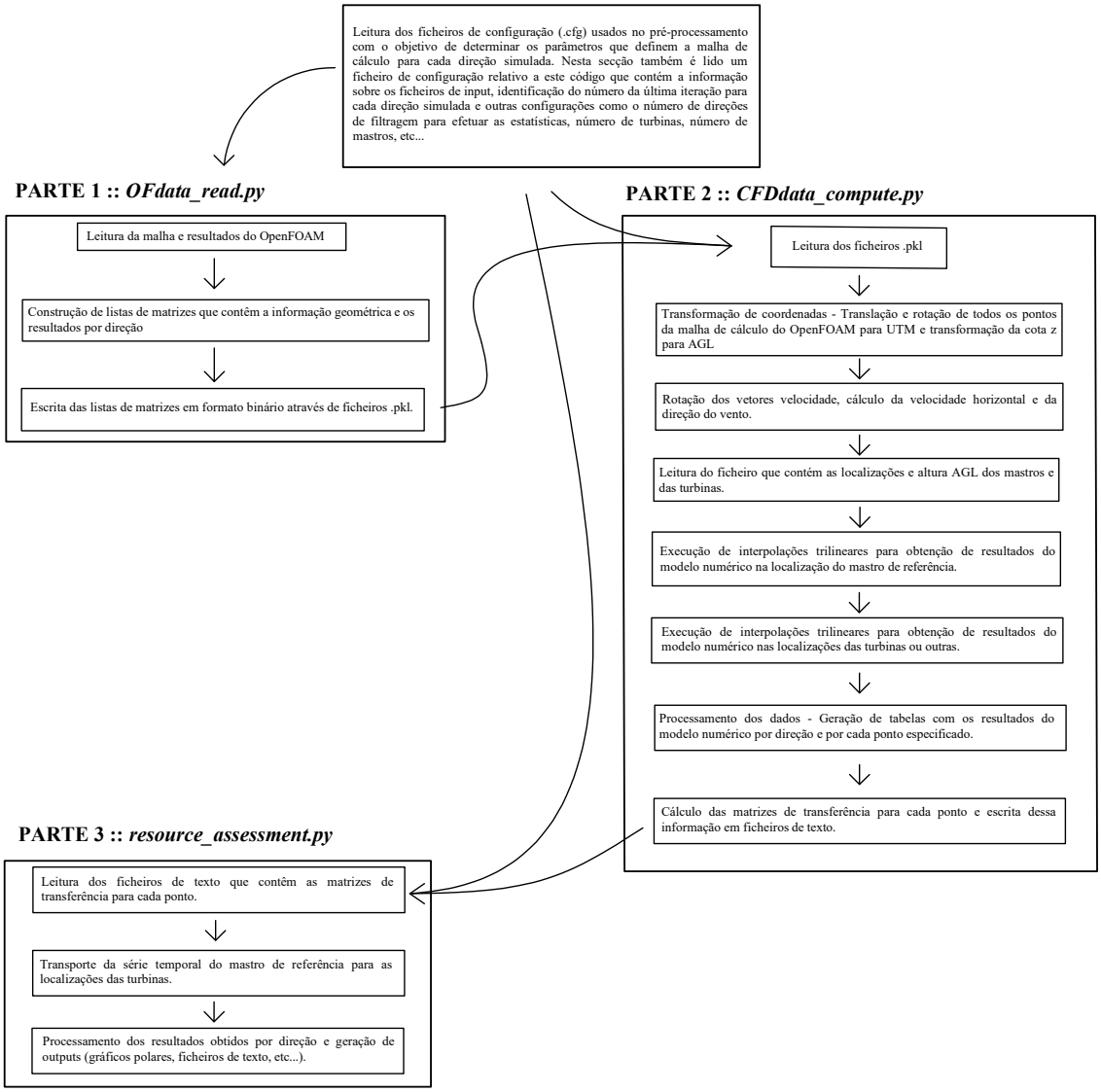


Figura B.1 – Esquema do algoritmo

B.1 Parte 1: OFdata_read.py

```
#############################################################################
# ------------------------------------------------------------------------- #
#                          WIND_ASSESSMENT PROGRAM                          #
# ------------------------------------------------------------------------- #
#   -- File_name: OFdata_read.py                          Version: 1.0  #
# ------------------------------------------------------------------------- #
#############################################################################
#%% CELL
import numpy as np
from fluidfoam import readmesh, readvector, readscalar
import time
import pickle

t_start = time.time()

### HEADING
print("#### ------------------------------------------------------------------------
    ####")
print("####                          WIND_ASSESSMENT PROGRAM
    ####")
print("#### ------------------------------------------------------------------------
    ####")
print("####  -- Script: OFdata_read.py
    ####")
print("#### ------------------------------------------------------------------------
    ####\n")

print("Script execution begun.\n")

#############################################################################
############                    FUNCTIONS                    ############
#############################################################################

def StringParser(string, file_path):
    with open(file_path,'r') as f:
        for line in f.readlines():
            if string in line:
                return line.strip().split()[2]

def StringParser_eq(string, file_path, col):
    with open(file_path,'r') as f:
        for line in f.readlines():
            if string in line:
                return line.strip().split(" = ")[col]

#############################################################################
## O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O ##
#############################################################################

#%% CELL
#############################################################################
#############################################################################
### WIND_ASSESSMENT.CFG FILE READ

nDirs = int(StringParser("nDirs ","../wind_assessment.cfg"))

Dirs = []
latestTimes = []

for i in range(0, nDirs):
    Dirs.append(StringParser_eq("dir%i =" %i,"../wind_assessment.cfg",1))
    latestTimes.append(StringParser_eq("dir%i =" %i,"../wind_assessment.cfg",2))

nMasts = int(StringParser("nMasts ","../wind_assessment.cfg"))
nTurbines = int(StringParser("nTurbines ","../wind_assessment.cfg"))
refMast_name = StringParser("refMast_name ","../wind_assessment.cfg")
mastsWTGS_fileName = StringParser("mastsWTGS_fileName ","../wind_assessment.cfg")
mast_records_file = StringParser("mast_records_file ","../wind_assessment.cfg")
skip = int(StringParser("skip ","../wind_assessment.cfg"))
output_plots = int(StringParser("output_plots ","../wind_assessment.cfg"))

```

```
sol_path = []
for d in range(0, len(Dirs)):
    sol_path.append("../../%s/" %Dirs[d])

#########################################################################
#########################################################################
### .CFG FILES READ

fPATH = []
fPATH_2 = []
fPATH_3 = []
xcentre = []
ycentre = []
cofx = []
cofy = []
rot_deg = []
nz = []
nig = []
njg = []
tolerance_x = []
tolerance_y = []

for i in range(0, len(Dirs)):
    fPATH.append('../../%s/_preproc/_gsurf_WINDIE/preproc.cfg' %Dirs[i])
    fPATH_2.append('../../%s/_preproc/_preprocDict' %Dirs[i])
    fPATH_3.append('../../%s/_preproc/_gsurf_WINDIE/windie.cfg' %Dirs[i])

for i in range(0, len(Dirs)):
    # preproc.cfg
    xcentre.append(float(StringParser('xcentre ', fPATH[i])))
    ycentre.append(float(StringParser('ycentre ', fPATH[i])))
    rot_deg.append(float(StringParser('rot ', fPATH[i])))
    cofx.append(float(StringParser('cofx ', fPATH[i])))
    cofy.append(float(StringParser('cofy ', fPATH[i])))

    # _preprocDict
    nz.append(int(StringParser('nz ',fPATH_2[i])))

    # windie.cfg
    nig.append(int(StringParser('nig ', fPATH_3[i])))
    njg.append(int(StringParser('njg ', fPATH_3[i])))

# Tolerance for nearest points lookup in trilinear interpolation
for i in range(0, len(Dirs)):
    tolerance_x.append(5*cofx[i])
    tolerance_y.append(5*cofy[i])

#%% CELL
#########################################################################
#########################################################################
### READ OPENFOAM MESH AND RESULTS

## INIT. NESTED LISTS
xof = [[] for _ in range(0,len(Dirs))]
yof = [[] for _ in range(0,len(Dirs))]
zof = [[] for _ in range(0,len(Dirs))]
UVEC = [[] for _ in range(0,len(Dirs))]
kof = [[] for _ in range(0,len(Dirs))]
x_ground = [[] for _ in range(0,len(Dirs))]
y_ground = [[] for _ in range(0,len(Dirs))]
z_ground = [[] for _ in range(0,len(Dirs))]

print("### --- ### Reading OpenFOAM internal mesh and results ### --- ###\n")
for i in range(0, len(Dirs)):

    t1 = time.time()
    print("### >>> rot = %i <<< ###\n" %rot_deg[i])

    print("     >> (nig-2) = %i" %(nig[i]-2))
    print("     >> (njg-2) = %i" %(njg[i]-2))
    print("     >> nz = %i\n" %nz[i])

    xof[i], yof[i], zof[i] = readmesh(sol_path[i])
```

```
    UVEC[i] = readvector(sol_path[i],"%s" %latestTimes[i], "U")
    kof[i] = readscalar(sol_path[i],"%s" %latestTimes[i], "k")

    print("\n   -- Reading ground surface --\n")
    ## Ground Surface
    x_ground[i], y_ground[i], z_ground[i] = readmesh(sol_path[i], boundary="ground")

    t2 = time.time()
    print("\nDone in %.2f s.\n" %(t2-t1))

#%% CELL
##############################################################################
##############################################################################
##### RESHAPE FLUIDFOAM LISTS TO GSRF3 IRREGULAR STRUCTURED GRID

X_matrix = [[] for _ in range(0,len(Dirs))]
Y_matrix = [[] for _ in range(0,len(Dirs))]
Z_matrix = [[] for _ in range(0,len(Dirs))]
U_matrix = [[] for _ in range(0,len(Dirs))]
V_matrix = [[] for _ in range(0,len(Dirs))]
W_matrix = [[] for _ in range(0,len(Dirs))]
k_matrix = [[] for _ in range(0,len(Dirs))]
zsurf_ground = [[] for _ in range(0,len(Dirs))]

print("### --- #### Building Results Matrixes ### --- ###\n")

for d in range(0, len(Dirs)):
    print("     >> xcentre = %.1f" %xcentre[d])
    print("     >> ycentre = %.1f" %ycentre[d])
    print("     >> rot = %i \n" %rot_deg[d])

    X_matrix[d] = np.reshape(xof[d], (nig[d]-2, njg[d]-2, nz[d]))
    Y_matrix[d] = np.reshape(yof[d], (nig[d]-2, njg[d]-2, nz[d]))
    Z_matrix[d] = np.reshape(zof[d], (nig[d]-2, njg[d]-2, nz[d]))
    U_matrix[d] = np.reshape(UVEC[d][0,:], (nig[d]-2, njg[d]-2, nz[d]))
    V_matrix[d] = np.reshape(UVEC[d][1,:], (nig[d]-2, njg[d]-2, nz[d]))
    W_matrix[d] = np.reshape(UVEC[d][2,:], (nig[d]-2, njg[d]-2, nz[d]))
    k_matrix[d] = np.reshape(kof[d], (nig[d]-2, njg[d]-2, nz[d]))
    zsurf_ground[d] = np.reshape(z_ground[d], (nig[d]-2, njg[d]-2))

##############################################################################
##############################################################################
##### WRITE OPENFOAM DATA IN BINARY .PKL FILES

print('Saving OpenFOAM data ...\n')

with open('../outputs/OF_data/X_matrix.pkl','wb') as f:
    pickle.dump(X_matrix, f)
print("X_matrix.pkl saved on outputs/OF_data/")

with open('../outputs/OF_data/Y_matrix.pkl','wb') as f:
    pickle.dump(Y_matrix, f)
print("Y_matrix.pkl saved on outputs/OF_data/")

with open('../outputs/OF_data/Z_matrix.pkl','wb') as f:
    pickle.dump(Z_matrix, f)
print("Y_matrix.pkl saved on outputs/OF_data/")

with open('../outputs/OF_data/U_matrix.pkl','wb') as f:
    pickle.dump(U_matrix, f)
print("U_matrix.pkl saved on outputs/OF_data/")

with open('../outputs/OF_data/V_matrix.pkl','wb') as f:
    pickle.dump(V_matrix, f)
print("V_matrix.pkl saved on outputs/OF_data/")

with open('../outputs/OF_data/W_matrix.pkl','wb') as f:
    pickle.dump(W_matrix, f)
print("W_matrix.pkl saved on outputs/OF_data/")

with open('../outputs/OF_data/k_matrix.pkl','wb') as f:
    pickle.dump(k_matrix, f)
print("k_matrix.pkl saved on outputs/OF_data/")
```

```
213
214 with open('../outputs/OF_data/zsurf_ground.pkl','wb') as f:
215     pickle.dump(zsurf_ground, f)
216 print("zsurf_ground.pkl saved on outputs/OF_data/")
217
218 t_end = time.time()
219
220 print("\nDone.\n")
221 print("\nTotal Execution time: %.2f s.\n" %(t_end-t_start))
```

B.2 Parte 2: CFDdata_compute.py

```
1  ###############################################################################
2  # ------------------------------------------------------------------------- #
3  #                          WIND_ASSESSMENT PROGRAM                          #
4  # ------------------------------------------------------------------------- #
5  #   -- File_name: CFDdata_compute.py                          Version: 3.2  #
6  # ------------------------------------------------------------------------- #
7  ###############################################################################
8
9  #%% CELL
10 import numpy as np
11 from scipy.interpolate import griddata, interp1d, LinearNDInterpolator
12 import pandas as pd
13 import matplotlib.pyplot as plt
14 import multiprocessing as mp
15 import pickle
16 import time
17
18 t_start = time.time()
19
20 ### HEADING
21 print("#### ------------------------------------------------------------------------
       ####")
22 print("####                            WIND_ASSESSMENT PROGRAM
       ####")
23 print("#### ------------------------------------------------------------------------
       ####")
24 print("####  -- Script: CFDdata_compute.py
       ####")
25 print("#### ------------------------------------------------------------------------
       ####\n")
26
27 print("Script execution begun.\n")
28
29 ###############################################################################
30 #############                    FUNCTIONS                    #############
31 ###############################################################################
32
33 def StringParser(string, file_path):
34     with open(file_path,'r') as f:
35         for line in f.readlines():
36             if string in line:
37                 return line.strip().split()[2]
38
39 def StringParser_eq(string, file_path, col):
40     with open(file_path,'r') as f:
41         for line in f.readlines():
42             if string in line:
43                 return line.strip().split(" = ")[col]
44
45 def trilinear_interpolation(xp, yp, zp, X_coords, Y_coords, Z_coords, Values, Nx, Ny, Nz
       , tolx, toly):
46
47     ### Filter data near xp, yp and zp otherwise griddata will be slow
48     xfiltered = []
49     yfiltered = []
50     zfiltered = []
51     valfiltered = []
52
53     for i in range(0, Nx):
54         for j in range(0, Ny):
```

```
            for k in range(0, Nz):
                if (X_coords[i,j,k] > xp - tolx and X_coords[i,j,k] < xp + tolx and
                    Y_coords[i,j,k] > yp - toly and Y_coords[i,j,k] < yp + toly and
                    Z_coords[i,j,k] > zp - 25 and Z_coords[i,j,k] < zp + 25):

                    xfiltered.append(X_coords[i,j,k])
                    yfiltered.append(Y_coords[i,j,k])
                    zfiltered.append(Z_coords[i,j,k])
                    valfiltered.append(Values[i,j,k])

    ### Define griddata inputs
    request = np.stack((xp, yp, zp)).T
    points = np.stack((xfiltered, yfiltered, zfiltered)).T
    values = valfiltered

    ### Perform trilinear interpolation via scipy griddata function
    result = griddata(points, values, request, method="linear", rescale=True)

    return result

##########################################################################
## O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O ##
##########################################################################

#%% CELL
##########################################################################
##########################################################################
### WIND_ASSESSMENT.CFG FILE READ

nDirs = int(StringParser("nDirs ","../wind_assessment.cfg"))

Dirs = []
latestTimes = []

for i in range(0, nDirs):
    Dirs.append(StringParser_eq("dir%i =" %i,"../wind_assessment.cfg",1))
    latestTimes.append(StringParser_eq("dir%i =" %i,"../wind_assessment.cfg",2))

nMasts = int(StringParser("nMasts ","../wind_assessment.cfg"))
nTurbines = int(StringParser("nTurbines ","../wind_assessment.cfg"))
refMast_name = StringParser("refMast_name ","../wind_assessment.cfg")
mastsWTGS_fileName = StringParser("mastsWTGS_fileName ","../wind_assessment.cfg")
mast_records_file = StringParser("mast_records_file ","../wind_assessment.cfg")
skip = int(StringParser("skip ","../wind_assessment.cfg"))
output_plots = int(StringParser("output_plots ","../wind_assessment.cfg"))
num_processes = int(StringParser("num_processes ","../wind_assessment.cfg"))
print_tables = int(StringParser("print_tables ","../wind_assessment.cfg"))
write_csv = int(StringParser("write_csv ","../wind_assessment.cfg"))

sol_path = []
for d in range(0, len(Dirs)):
    sol_path.append("../../%s/" %Dirs[d])

##########################################################################
##########################################################################
### .CFG FILES READ

fPATH = []
fPATH_2 = []
fPATH_3 = []
xcentre = []
ycentre = []
cofx = []
cofy = []
rot_deg = []
nz = []
nig = []
njg = []
tolerance_x = []
tolerance_y = []

for i in range(0, len(Dirs)):
    fPATH.append('../../%s/_preproc/_gsurf_WINDIE/preproc.cfg' %Dirs[i])
```

```
    fPATH_2.append('../../%s/_preproc/_preprocDict' %Dirs[i])
    fPATH_3.append('../../%s/_preproc/_gsurf_WINDIE/windie.cfg' %Dirs[i])

for i in range(0, len(Dirs)):
    # preproc.cfg
    xcentre.append(float(StringParser('xcentre ', fPATH[i])))
    ycentre.append(float(StringParser('ycentre ', fPATH[i])))
    rot_deg.append(float(StringParser('rot ', fPATH[i])))
    cofx.append(float(StringParser('cofx ', fPATH[i])))
    cofy.append(float(StringParser('cofy ', fPATH[i])))

    # _preprocDict
    nz.append(int(StringParser('nz ',fPATH_2[i])))

    # windie.cfg
    nig.append(int(StringParser('nig ', fPATH_3[i])))
    njg.append(int(StringParser('njg ', fPATH_3[i])))

# Tolerance for nearest points lookup in trilinear interpolation
for i in range(0, len(Dirs)):
    tolerance_x.append(5*cofx[i])
    tolerance_y.append(5*cofy[i])

#%% CELL
##############################################################################
##############################################################################
##### LOAD OpenFOAM DATA

print('Loading OpenFOAM data ...')
with open('../outputs/OF_data/X_matrix.pkl', 'rb') as f:
    X_matrix = pickle.load(f)
with open('../outputs/OF_data/Y_matrix.pkl', 'rb') as f:
    Y_matrix = pickle.load(f)
with open('../outputs/OF_data/Z_matrix.pkl', 'rb') as f:
    Z_matrix = pickle.load(f)

with open('../outputs/OF_data/U_matrix.pkl', 'rb') as f:
    U_matrix = pickle.load(f)
with open('../outputs/OF_data/V_matrix.pkl', 'rb') as f:
    V_matrix = pickle.load(f)
with open('../outputs/OF_data/W_matrix.pkl', 'rb') as f:
    W_matrix = pickle.load(f)
with open('../outputs/OF_data/k_matrix.pkl', 'rb') as f:
    k_matrix = pickle.load(f)

with open('../outputs/OF_data/zsurf_ground.pkl', 'rb') as f:
    zsurf_ground = pickle.load(f)

#%% CELL
##############################################################################
##############################################################################
##### COORDINATES TRANSFORMATION
rot = []
xr = [[] for _ in range(0,len(Dirs))]
yr = [[] for _ in range(0,len(Dirs))]
X_UTM = [[] for _ in range(0,len(Dirs))]
Y_UTM = [[] for _ in range(0,len(Dirs))]
Z_agl = [[] for _ in range(0,len(Dirs))]

print("\nPerforming coordinates transformations ...\n")

## ROTATION
for d in range(0,len(Dirs)):
    rot_deg[d] = -(180-((450-rot_deg[d]) % 360))  # DEG
    rot.append(rot_deg[d] * np.pi/180)        # TO RAD

for d in range(0, len(Dirs)):
    xr[d] = X_matrix[d]*np.cos(rot[d]) - Y_matrix[d]*np.sin(rot[d])
    yr[d] = X_matrix[d]*np.sin(rot[d]) + Y_matrix[d]*np.cos(rot[d])

## TRANSLATION
for d in range(0, len(Dirs)):
    X_UTM[d] = xr[d] + xcentre[d]
```

```
    Y_UTM[d] = yr[d] + ycentre[d]

## BUILD ZAGL MATRIX
for d in range(0, len(Dirs)):
    Z_agl[d] = np.zeros([nig[d]-2, njg[d]-2, nz[d]])

for d in range(0, len(Dirs)):
    for k in range(0, nz[d]):
        Z_agl[d][:,:,k] = Z_matrix[d][:,:,k] - zsurf_ground[d][:,:]

#%% CELL
########################################################################
########################################################################
## VELOCITY VECTORS ROTATION
## HORIZONTAL VELOCITY CALCULATION
## WIND DIRECTION CALCULATION

Ur = [[] for _ in range(0,len(Dirs))]
Vr = [[] for _ in range(0,len(Dirs))]
hVel = [[] for _ in range(0,len(Dirs))]

phic = [[] for _ in range(0,len(Dirs))]
phi_deg = [[] for _ in range(0,len(Dirs))]
phi = [[] for _ in range(0,len(Dirs))]

for d in range(0, len(Dirs)):
    Ur[d] = U_matrix[d]*np.cos(rot[d]) - V_matrix[d]*np.sin(rot[d])
    Vr[d] = U_matrix[d]*np.sin(rot[d]) + V_matrix[d]*np.cos(rot[d])

for d in range(0, len(Dirs)):
    hVel[d] = np.sqrt(Ur[d]**2 + Vr[d]**2)

for d in range(0, len(Dirs)):

    phic[d] = np.zeros([nig[d]-2, njg[d]-2, nz[d]])

    for i in range(0, nig[d]-2):
        for j in range(0, njg[d]-2):
            for k in range(0, nz[d]):

                ### 1st Quadrant
                if Vr[d][i,j,k] > 0 and Ur[d][i,j,k] > 0:
                    phic[d][i,j,k] = 90 - np.arctan(abs(Vr[d][i,j,k])/abs(Ur[d][i,j,k]))
    *180/np.pi

                ### 2nd Quadrant
                elif Vr[d][i,j,k] > 0 and Ur[d][i,j,k] < 0:
                    phic[d][i,j,k] = 270 + np.arctan(abs(Vr[d][i,j,k])/abs(Ur[d][i,j,k])
    )*180/np.pi

                ### 3rd Quadrant
                elif Vr[d][i,j,k] < 0 and Ur[d][i,j,k] < 0:
                    phic[d][i,j,k] = 270 - np.arctan(abs(Vr[d][i,j,k])/abs(Ur[d][i,j,k])
    )*180/np.pi

                ### 4th Quadrant
                elif Vr[d][i,j,k] < 0 and Ur[d][i,j,k] > 0:
                    phic[d][i,j,k] = 90 + np.arctan(abs(Vr[d][i,j,k])/abs(Ur[d][i,j,k]))
    *180/np.pi

    phi_deg[d] = phic[d] - 180

for d in range(0, len(Dirs)):

    phi[d] = np.zeros([nig[d]-2, njg[d]-2, nz[d]])

    for i in range(0, nig[d]-2):
        for j in range(0, njg[d]-2):
            for k in range(0, nz[d]):

                if phi_deg[d][i,j,k] < 0:
                    phi[d][i,j,k] = phi_deg[d][i,j,k] + 360
                else:
```

```
                    phi[d][i,j,k] = phi_deg[d][i,j,k]

#%% CELL
########################################################################
########################################################################
##### MASTS AND WTGS FILE DATA READ

label_masts = []
label_turbs = []

xMasts = []
yMasts = []
zMasts = []

xTurbs = []
yTurbs = []
zTurbs = []

cont = 0

with open("../inputs/%s" %(mastsWTGS_fileName),"r") as f:
    for line in f.readlines():

        if cont > 0:
            label = line.strip().split()[0]
            xcoord = float(line.strip().split()[1])
            ycoord = float(line.strip().split()[2])
            zcoord = float(line.strip().split()[3])

        if 0 < cont <= nMasts:
            label_masts.append(label)
            xMasts.append(xcoord)
            yMasts.append(ycoord)
            zMasts.append(zcoord)

        if nMasts < cont <= nTurbines + nMasts:
            label_turbs.append(label)
            xTurbs.append(xcoord)
            yTurbs.append(ycoord)
            zTurbs.append(zcoord)

        cont = cont + 1

## Find reference mast values
for l, xm, ym, zm in zip(label_masts, xMasts, yMasts, zMasts):
    if l == refMast_name:
        x_refMast = xm
        y_refMast = ym
        z_refMast = zm

#%% CELL
########################################################################
########################################################################
##### TRILINEAR INTERPOLATIONS :: MASTS

hVel_mast = []
dir_mast = []

print("### --- ### Performing trilinear interpolations ### --- ###\n")

#### REFERENCE MAST

print(">> Selected reference mast is \'%s\' \n" %refMast_name)

for d in range(0, len(Dirs)):

    hVel_mast.append(trilinear_interpolation(x_refMast, y_refMast, z_refMast,
        X_UTM[d], Y_UTM[d], Z_agl[d], hVel[d], nig[d]-2, njg[d]-2, nz[d],
        tolerance_x[d], tolerance_y[d]))

    Um = trilinear_interpolation(x_refMast, y_refMast, z_refMast,
            X_UTM[d], Y_UTM[d], Z_agl[d], Ur[d], nig[d]-2, njg[d]-2, nz[d],
            tolerance_x[d], tolerance_y[d])
```

```
343
344     Vm = trilinear_interpolation(x_refMast, y_refMast, z_refMast,
345         X_UTM[d], Y_UTM[d], Z_agl[d], Vr[d], nig[d]-2, njg[d]-2, nz[d],
346         tolerance_x[d], tolerance_y[d])
347
348     ### 1st Quadrant
349     if Vm > 0 and Um > 0:
350         dir = 90 - np.arctan(abs(Vm)/abs(Um))*180/np.pi
351
352     ### 2nd Quadrant
353     elif Vm > 0 and Um < 0:
354         dir = 270 + np.arctan(abs(Vm)/abs(Um))*180/np.pi
355
356     ### 3rd Quadrant
357     elif Vm < 0 and Um < 0:
358         dir = 270 - np.arctan(abs(Vm)/abs(Um))*180/np.pi
359
360     ### 4th Quadrant
361     elif Vm < 0 and Um > 0:
362         dir = 90 + np.arctan(abs(Vm)/abs(Um))*180/np.pi
363
364     dir_m = dir - 180
365
366     if dir_m < 0:
367         dir_m = dir_m + 360
368
369     dir_mast.append(float(dir_m))
370
371     print("%s :: rot = %s :: Horizontal Velocity hVel = %.4f m/s :: Wind Direction = %.4
    f degrees"
372         %(refMast_name,Dirs[d],hVel_mast[d], dir_mast[d]))
373
374 #%% CELL
375 ##########################################################################
376 ##########################################################################
377 ##### TRILINEAR INTERPOLATIONS :: WIND TURBINES
378
379 # local options
380 parallelize = True
381
382 # INIT.
383 hVel_turbs = [[] for _ in range(0,len(Dirs))]
384 dir_turbs = [[] for _ in range(0,len(Dirs))]
385 ratio = [[] for _ in range(0,len(Dirs))]
386
387 print("\nPerforming trilinear interpolations for wind turbines ...")
388
389 tinterp_i = time.time()
390
391 if parallelize == True:
392     ### PARALLEL CODE
393     print("Number of processes = %i\n" %num_processes)
394
395     def calculate_turbine(d, turb):
396
397         print(f"Dir %s -- Turbine %s" % (Dirs[d], label_turbs[turb]))
398
399         hVel_t = trilinear_interpolation(xTurbs[turb], yTurbs[turb], zTurbs[turb],
400             X_UTM[d], Y_UTM[d], Z_agl[d], hVel[d], nig[d]-2, njg[d]-2, nz[d],
401             tolerance_x[d], tolerance_y[d])
402
403         Ut = trilinear_interpolation(xTurbs[turb], yTurbs[turb], zTurbs[turb],
404             X_UTM[d], Y_UTM[d], Z_agl[d], Ur[d], nig[d]-2, njg[d]-2, nz[d],
405             tolerance_x[d], tolerance_y[d])
406
407         Vt = trilinear_interpolation(xTurbs[turb], yTurbs[turb], zTurbs[turb],
408             X_UTM[d], Y_UTM[d], Z_agl[d], Vr[d], nig[d]-2, njg[d]-2, nz[d],
409             tolerance_x[d], tolerance_y[d])
410
411         ### 1st Quadrant
412         if Vt > 0 and Ut > 0:
413             dir = 90 - np.arctan(abs(Vt)/abs(Ut))*180/np.pi
414
```

```
        ### 2nd Quadrant
        elif Vt > 0 and Ut < 0:
            dir = 270 + np.arctan(abs(Vt)/abs(Ut))*180/np.pi

        ### 3rd Quadrant
        elif Vt < 0 and Ut < 0:
            dir = 270 - np.arctan(abs(Vt)/abs(Ut))*180/np.pi

        ### 4th Quadrant
        elif Vt < 0 and Ut > 0:
            dir = 90 + np.arctan(abs(Vt)/abs(Ut))*180/np.pi

        dir_t = dir - 180

        if dir_t < 0:
            dir_t = dir_t + 360

        return int(np.round(dir_t)), float(hVel_t), float(hVel_t/hVel_mast[d])

    if __name__ == "__main__":

        # Create a multiprocessing.Pool with the specified number of processes
        with mp.Pool(processes=num_processes) as pool:
            results = []

            for d in range(len(Dirs)):
                for turb in range(nTurbines):

                    # Apply the function asynchronously using pool
                    result = pool.apply_async(calculate_turbine, (d, turb))
                    results.append(result)

            # Wait for all results to be ready
            for result in results:
                result.wait()

        for d in range(len(Dirs)):
            for turb in range(nTurbines):

                ## Access results from parallel processing
                res = results[d * nTurbines + turb].get()
                dir_t, hVel_t, ratio_t = res

                dir_turbs[d].append(dir_t)
                hVel_turbs[d].append(hVel_t)
                ratio[d].append(ratio_t)
else:
    ### SERIAL CODE ###
    hVel_turbs = [[] for _ in range(0,len(Dirs))]
    dir_turbs = [[] for _ in range(0,len(Dirs))]
    ratio = [[] for _ in range(0,len(Dirs))]

    print("")
    for d in range(0, len(Dirs)):
        print("Performing trilinear interpolations for wind turbines, rot = %s" %(Dirs[d
    ]))
        for turb in range(0, nTurbines):

            print("     %i -- Turbine %s" %(turb, label_turbs[turb]))

            hVel_t = trilinear_interpolation(xTurbs[turb], yTurbs[turb], zTurbs[turb],
                X_UTM[d], Y_UTM[d], Z_agl[d], hVel[d], nig[d]-2, njg[d]-2, nz[d],
                tolerance_x[d], tolerance_y[d])

            Ut = trilinear_interpolation(xTurbs[turb], yTurbs[turb], zTurbs[turb],
                X_UTM[d], Y_UTM[d], Z_agl[d], Ur[d], nig[d]-2, njg[d]-2, nz[d],
                tolerance_x[d], tolerance_y[d])

            Vt = trilinear_interpolation(xTurbs[turb], yTurbs[turb], zTurbs[turb],
                X_UTM[d], Y_UTM[d], Z_agl[d], Vr[d], nig[d]-2, njg[d]-2, nz[d],
                tolerance_x[d], tolerance_y[d])

            ### 1st Quadrant
```

```
            if Vt > 0 and Ut > 0:
                dir = 90 - np.arctan(abs(Vt)/abs(Ut))*180/np.pi

            ### 2nd Quadrant
            elif Vt > 0 and Ut < 0:
                dir = 270 + np.arctan(abs(Vt)/abs(Ut))*180/np.pi

            ### 3rd Quadrant
            elif Vt < 0 and Ut < 0:
                dir = 270 - np.arctan(abs(Vt)/abs(Ut))*180/np.pi

            ### 4th Quadrant
            elif Vt < 0 and Ut > 0:
                dir = 90 + np.arctan(abs(Vt)/abs(Ut))*180/np.pi

            dir_t = dir - 180

            if dir_t < 0:
                dir_t = dir_t + 360

            dir_turbs[d].append(int(np.round(dir_t)))
            hVel_turbs[d].append(float(hVel_t))
            ratio[d].append(float(hVel_t/hVel_mast[d]))

tinterp_f = time.time()

print("\n Interpolations for wind turbines done in %.2f s \n" %(tinterp_f-tinterp_i))

#%% CELL
########################################################################
########################################################################
### COMBINE DATA

print("Processing data ...\n")
data_label_turbs = [[] for _ in range(0,nTurbines)]
data_hVel_turbs = [[] for _ in range(0,nTurbines)]
data_ratio = [[] for _ in range(0,nTurbines)]
data_dir_turbs = [[] for _ in range(0,nTurbines)]
data_delta_phi = [[] for _ in range(0,nTurbines)]

table1_dict = [[] for _ in range(0,nTurbines)]

for n in range(0, nTurbines):
    for d in range(0, len(Dirs)):
        data_label_turbs[n].append(label_turbs[n])
        data_hVel_turbs[n].append(hVel_turbs[d][n])
        data_ratio[n].append(ratio[d][n])
        data_dir_turbs[n].append(dir_turbs[d][n])

        dphi = dir_turbs[d][n]-dir_mast[d]

        if 360 > dphi > 180:
            dphi_corr = dphi - 360
        elif -180 > dphi > -360:
            dphi_corr = dphi + 360
        else:
            dphi_corr = dphi

        data_delta_phi[n].append(dphi_corr)

### Output tables
indext_str = []
for ind in range(0, len(Dirs)):
    indext_str.append("")

for n in range(0, nTurbines):
    table1_dict[n] = {"WT_LABEL": data_label_turbs[n], "HORZ._VEL_[m/s]":
    data_hVel_turbs[n],
        "RATIO_turb/mast": data_ratio[n], "Wind_Dir._(deg)": data_dir_turbs[n],
        "Delta_phi": data_delta_phi[n]}

    table1_df = pd.DataFrame(data = table1_dict[n], index=indext_str)

```

```
559     if print_tables == 1:
560         print(table1_df)
561 
562     # Write tables to files
563     table1_str = table1_df.to_string(index = "False")
564     f = open("../outputs/txt/CFDdata_WT_%s.txt" %label_turbs[n], "w")
565     f.write(table1_str)
566     f.close()
567     print("CFDdata_WT_%s.txt written to outputs/txt/ " %label_turbs[n])
568 
569     if write_csv == 1:
570         table1_df.to_csv("../outputs/csv/CFDdata_WT_%s.csv" %label_turbs[n], index=False
    )
571         print("CFDdata_WT_%s.csv written to outputs/csv/\n " %label_turbs[n])
572 
573 print("\nDone.\n")
574 
575 #%% CELL
576 ########################################################################
577 ########################################################################
578 ##### GENERATE TRANSFER MATRIX FOR CFD DATA
579 
580 print("Generating Transfer Matrix for CFD DATA ...\n")
581 
582 ratio_allDir = [[] for _ in range(0,nTurbines)]
583 delta_phi_allDir = [[] for _ in range(0,nTurbines)]
584 
585 extended_dir_turbs = [[] for _ in range(0,nTurbines)]
586 extended_ratio = [[] for _ in range(0,nTurbines)]
587 extended_delta_phi = [[] for _ in range(0,nTurbines)]
588 
589 table2_dict = [[] for _ in range(0,nTurbines)]
590 
591 all_dirs = np.arange(0,360,1)
592 
593 for n in range(0, nTurbines):
594 
595     ## Extend data to handle cyclic boundary
596     extended_dir_turbs[n] = np.concatenate((np.array(data_dir_turbs[n]) - 360,
597         np.array(data_dir_turbs[n]), np.array(data_dir_turbs[n]) + 360))
598 
599     extended_ratio[n] = np.concatenate((data_ratio[n], data_ratio[n], data_ratio[n]))
600     extended_delta_phi[n] = np.concatenate((data_delta_phi[n], data_delta_phi[n],
601         data_delta_phi[n]))
602 
603     ## Perform radial interpolation
604     ratio_interp_func = interp1d(extended_dir_turbs[n], extended_ratio[n],
605         fill_value="extrapolate")
606 
607     delta_phi_interp_func = interp1d(extended_dir_turbs[n], extended_delta_phi[n],
608         fill_value="extrapolate")
609 
610     ratio_allDir[n] = ratio_interp_func(all_dirs)
611     delta_phi_allDir[n] = delta_phi_interp_func(all_dirs)
612 
613 # Output table
614 for n in range(0, nTurbines):
615 
616     table2_dict[n] = {"Wind_Dir._(deg)": all_dirs,
617         "RATIO_turb/mast": ratio_allDir[n], "Delta_phi": delta_phi_allDir[n]}
618 
619     table2_df = pd.DataFrame(data = table2_dict[n])
620 
621     # Write tables to files
622     table2_str = table2_df.to_string(index = "False")
623     f = open("../outputs/txt/MATRIX_WT_%s.txt" %label_turbs[n], "w")
624     f.write(table2_str)
625     f.close()
626     print("MATRIX_WT_%s.txt written to outputs/txt/ " %label_turbs[n])
627 
628     if write_csv == 1:
629         table2_df.to_csv("../outputs/csv/MATRIX_WT_%s.csv" %label_turbs[n], index=False)
630         print("MATRIX_WT_%s.csv written to outputs/csv/ " %label_turbs[n])
```

```
631
632 # Output plots
633 if output_plots == 1:
634     for index in range(0, nTurbines):
635         fig, ax1 = plt.subplots(figsize=(10,7))
636
637         ax1.plot(all_dirs, ratio_allDir[index], "--", color = "tab:blue",
638             label="Transfer Matrix")
639         ax1.plot(data_dir_turbs[index], data_ratio[index], "o", color = "tab:red",
640             label="CFD Data")
641
642         ax1.set_xticks(np.arange(0,360+30,30))
643         ax1.set_xlim([0,360])
644         ax1.set_title("WIND TURBINE %s - RATIO" %label_turbs[index])
645         ax1.set_xlabel("Wind Direction (°)")
646         ax1.set_ylabel("Ratio TURBINE / MAST")
647         ax1.grid()
648         ax1.legend()
649         plt.savefig("../outputs/png/WT_%s.png" %label_turbs[index])
650         print("WT_%s.png saved on outputs/png/" %label_turbs[index])
651         plt.close()
652
653     for index in range(0, nTurbines):
654         fig, ax2 = plt.subplots(figsize=(10,7))
655
656         ax2.plot(all_dirs, delta_phi_allDir[index], "--", color = "tab:blue",
657             label="Transfer Matrix")
658         ax2.plot(data_dir_turbs[index], data_delta_phi[index], "o", color = "tab:red",
659             label="CFD Data")
660
661         ax2.set_xticks(np.arange(0,360+30,30))
662         ax2.set_xlim([0,360])
663         ax2.set_title("WIND TURBINE %s - DELTA PHI" %label_turbs[index])
664         ax2.set_xlabel("Wind Direction (°)")
665         ax2.set_ylabel("Delta Phi = phi_turb - phi_mast (°)")
666         ax2.grid()
667         ax2.legend()
668         plt.savefig("../outputs/png/WT_%s_deltaPhi.png" %label_turbs[index])
669         print("WT_%s_deltaPhi.png saved on outputs/png/" %label_turbs[index])
670         plt.close()
671
672 t_end = time.time()
673
674 print("\nDone.\n")
675 print("\nTotal Execution time: %.2f s.\n" %(t_end-t_start))
676
677 # %%
```

B.3 Parte 3: resource_assessment.py

```
1  ###############################################################################
2  # ------------------------------------------------------------------------- #
3  #                          WIND_ASSESSMENT PROGRAM                          #
4  # ------------------------------------------------------------------------- #
5  #   -- File_name: resource_assessment.py                    Version: 2.3  #
6  # ------------------------------------------------------------------------- #
7  ###############################################################################
8  #%% CELL
9  import numpy as np
10 import pandas as pd
11 import matplotlib.pyplot as plt
12 import time
13 from statistics import mean
14
15 t_start = time.time()
16
17 ### HEADING
18 print("#### ------------------------------------------------------------------------
       ####")
19 print("####                          WIND_ASSESSMENT PROGRAM
       ####")
20 print("#### ------------------------------------------------------------------------
```

```
    ####")
print("####   -- Script: resource_assessment.py
    ####")
print("#### ------------------------------------------------------------------------
    ####\n")

print("Script execution begun.\n")

########################################################################
############                    FUNCTIONS                    #############
########################################################################

def StringParser(string, file_path):
    with open(file_path,'r') as f:
        for line in f.readlines():
            if string in line:
                return line.strip().split()[2]

########################################################################
## O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O ##
########################################################################

### LOCAL OPTIONS
rho = 1.225

########################################################################
########################################################################
### WIND_ASSESSMENT.CFG FILE READ

mast_records_file = StringParser("mast_records_file ","../wind_assessment.cfg")
mastsWTGS_fileName = StringParser("mastsWTGS_fileName ","../wind_assessment.cfg")
skip = int(StringParser("skip ","../wind_assessment.cfg"))
nTurbines = int(StringParser("nTurbines ","../wind_assessment.cfg"))
nMasts = int(StringParser("nMasts ","../wind_assessment.cfg"))
stats_nDirs = int(StringParser("stats_nDirs ","../wind_assessment.cfg"))
output_plots = int(StringParser("output_plots ","../wind_assessment.cfg"))

########################################################################
########################################################################
##### MASTS AND WTGS FILE DATA READ

print("# -- Reading input files ... -- #\n")
print("     >> Number of Turbines = %i\n" %nTurbines)

label_masts = []
label_turbs = []
d_turbs = []
A_turbs = []

cont = 0

print(" -- Reading %s file ... --\n" %(mastsWTGS_fileName))
with open("../inputs/%s" %(mastsWTGS_fileName),"r") as f:
    for line in f.readlines():

        if cont > 0:
            label = line.strip().split()[0]
            d = float(line.strip().split()[4])

        if 0 < cont <= nMasts:
            label_masts.append(label)

        if nMasts < cont <= nTurbines + nMasts:
            label_turbs.append(label)
            d_turbs.append(d)
            A_turbs.append(np.pi*d**2/4)
        cont = cont + 1

########################################################################
########################################################################
### READ TRANSFER MATRIX OUTPUT FROM CFDdata_compute.py

ratio_WT = [[] for _ in range(0,nTurbines)]
```

```
delta_phi_WT = [[] for _ in range(0,nTurbines)]

for n in range(0, nTurbines):
    cont = 0
    print(" -- Reading output from /outputs/txt/MATRIX_WT_%s.txt file ... --" %
    label_turbs[n])
    with open("../outputs/txt/MATRIX_WT_%s.txt" %label_turbs[n]) as f:
        for line in f.readlines():
            if cont > 0:
                ratio_WT[n].append(float(line.strip().split()[2]))
                delta_phi_WT[n].append(float(line.strip().split()[3]))
            cont = cont + 1

#%% CELL
########################################################################
########################################################################
### TRANSPORT MAST TIME SERIES TO TURBINES

print("\n# -- Transporting mast time series to wind turbines locations -- #\n")

rec_dirs_round = []

# Read mast data as dataframe
rec_df = pd.read_csv("../inputs/%s" %mast_records_file, header=skip-1)

rec_dirs = pd.to_numeric(rec_df.iloc[:,1])
vel = pd.to_numeric(rec_df.iloc[:,2])

for rec_dir in rec_dirs:
    rec_dirs_round.append(int(np.round(rec_dir)))

vel_turbs = [[] for _ in range(0,nTurbines)]
P_turbs = [[] for _ in range(0,nTurbines)]
dir_turbs = [[] for _ in range(0,nTurbines)]

for turbine in range(0, nTurbines):
    for rec_dir, rec_vel in zip(rec_dirs_round, vel):

        if rec_dir == 360:
            rec_dir = 0

        ### DIRECTION SERIES FOR EACH TURBINE
        dir_turbs[turbine].append(int(np.round(rec_dir +
            delta_phi_WT[turbine][rec_dir])))

        ### VELOCITY SERIES FOR EACH TURBINE
        vel_turbs[turbine].append(rec_vel*ratio_WT[turbine][rec_dir])

        ### WIND POWER SERIES FOR EACH TURBINE
        P_turbs[turbine].append(0.5*rho*A_turbs[turbine]*
            (rec_vel*ratio_WT[turbine][rec_dir])**3)

    print("turbine %s done." %label_turbs[turbine])

#%% CELL
########################################################################
########################################################################
### STATISTICS BY DIRECTION

print("\nFiltering data by direction ...")
print("     >> Number of directions for wind turbines statistics : %i\n"
    %(stats_nDirs))

dirVals = [[] for _ in range(0,stats_nDirs)]
velDirTurbs = [[] for _ in range(0, nTurbines)]
freqTurbs = [[] for _ in range(0, nTurbines)]

step = int(360/stats_nDirs)

lower = []
upper = []

### Define filtering intervals
```

```
for dd in range(0,360, step):
    low = dd - step/2
    up = dd + step/2

    if low < 0:
        low = low + 360

    lower.append(low)
    upper.append(up)

### Filtering data by Turbine and by Direction
for turbine in range(0, nTurbines):

    velDirVals = [[] for _ in range(0, stats_nDirs)]
    vmean = [[] for _ in range(0, stats_nDirs)]

    wd = [[] for _ in range(0, stats_nDirs)]
    freq = [[] for _ in range(0, stats_nDirs)]

    for index, low, up in zip(range(0,stats_nDirs), lower, upper):
        for i in range(0, len(rec_dirs_round)):
            if low > up:
                if dir_turbs[turbine][i] >= low or dir_turbs[turbine][i] < up:
                    dirVals[index].append(rec_dirs_round[i])
                    velDirVals[index].append(vel_turbs[turbine][i])

                    wd[index].append(dir_turbs[turbine][i])
            else:
                if dir_turbs[turbine][i] >= low and dir_turbs[turbine][i] < up:
                    dirVals[index].append(rec_dirs_round[i])
                    velDirVals[index].append(vel_turbs[turbine][i])

                    wd[index].append(dir_turbs[turbine][i])

        if len(velDirVals[index]) == 0:
            vmean[index] = 0
            freq[index] = 0
        else:
            vmean[index] = mean(velDirVals[index])
            freq[index] = len(wd[index])/len(rec_dirs_round)*100

    print("turbine %s done." %label_turbs[turbine])

    velDirTurbs[turbine] = vmean
    freqTurbs[turbine] = freq

#%% CELL
########################################################################
########################################################################
### FILTERING AVERAGE VELOCITY BY TURBINE

vmean_turbs = []

for turbine in range(0, nTurbines):
    vmean_turbs.append(mean(vel_turbs[turbine]))

f = open("../outputs/txt/turbines_avg_velocity.txt", 'w')
for turbine in range(0, nTurbines):
    f.write("%s   %.4f\n" %(label_turbs[turbine], vmean_turbs[turbine]))
f.close()

########################################################################
########################################################################
### OUTPUTS TABLES AND PLOTS

table_dict = [[] for _ in range(0,nTurbines)]

direcs = np.arange(0,360, step)

# force the hiding of table indexes
indext_str = []
for ind in range(0, stats_nDirs):
    indext_str.append("")
```

```
236
237 for n in range(0, nTurbines):
238     table_dict[n] = {"DIR": direcs, "freq [%]": freqTurbs[n], "v [m/s]": velDirTurbs[n]}
239     table_df = pd.DataFrame(data = table_dict[n], index=indext_str)
240
241     # Write tables to files
242     table_str = table_df.to_string(index = "False")
243     f = open("../outputs/txt/transportedResults_WT_%s.txt" %label_turbs[n], "w")
244     f.write(table_str)
245     f.close()
246     print("transportedResults_WT_%s.txt written to outputs/txt/ " %label_turbs[n])
247
248 if output_plots == 1:
249     print("")
250     dir_rad = []
251
252     for direc in range(0,360, step):
253         dir_rad.append(direc*np.pi/180)
254     width = 2*np.pi/stats_nDirs
255
256     for t in range(0, nTurbines):
257         ax0 = plt.subplot(111,polar=True)
258         ax0.bar(dir_rad,velDirTurbs[t],color ='blue',width = width,bottom = 0,
259                     edgecolor = 'k', align = 'center',alpha=0.4,label='Velocidade [m/s]'
    )
260
261         ax0.set_xticks(dir_rad)
262         ax0.set_rlabel_position(75)
263         ax0.legend()
264
265         angle = np.radians(45)
266         ax0.legend(loc="lower left",
267                 bbox_to_anchor=(.5 + np.cos(angle)/2, .5 + np.sin(angle)/2))
268
269         ax0.set_theta_zero_location("N")
270         ax0.set_theta_direction(-1)
271         ax0.set_title('Velocity by Direction on Turbine %s' %label_turbs[t])
272         ax0.set_xlabel('Direção do Vento [°]')
273         plt.savefig("../outputs/png/velocity_WT_%s.png" %label_turbs[t])
274         print("velocity_WT_%s.png saved on outputs/png/" %label_turbs[t])
275         plt.close()
276
277     for t in range(0, nTurbines):
278         ax1 = plt.subplot(111,polar=True)
279         ax1.bar(dir_rad,freqTurbs[t],color ='maroon',width = width,bottom = 0,
280                     edgecolor = 'k', align = 'center',alpha=0.4,label='Frequência [%]')
281
282         ax1.set_xticks(dir_rad)
283         ax1.set_rlabel_position(75)
284         ax1.legend()
285
286         angle = np.radians(45)
287         ax1.legend(loc="lower left",
288                 bbox_to_anchor=(.5 + np.cos(angle)/2, .5 + np.sin(angle)/2))
289
290         ax1.set_theta_zero_location("N")
291         ax1.set_theta_direction(-1)
292         ax1.set_title('Frequency by Direction on Turbine %s' %label_turbs[t])
293         ax1.set_xlabel('Direção do Vento [°]')
294         plt.savefig("../outputs/png/freq_WT_%s.png" %label_turbs[t])
295         print("freq_WT_%s.png saved on outputs/png/" %label_turbs[t])
296         plt.close()
297
298 ########################################################################
299
300 t_end = time.time()
301
302 print("\nDone.\n")
303 print("\nTotal Execution time: %.2f s.\n" %(t_end-t_start))
```

C Código *Python* de acoplamento

C.1 coupling_wrfmean.py

```
#############################################################################
# ------------------------------------------------------------------------- #
#                      OPENFOAM+WRF COUPLING PROGRAM                        #
# ------------------------------------------------------------------------- #
#   -- File_name: coupling_wrfmean.py                        Version: 3.0  #
# ------------------------------------------------------------------------- #
#############################################################################
#%% CELL
import netCDF4
import numpy as np
from scipy.interpolate import griddata
from sklearn.linear_model import LinearRegression
import time
import os
import utm

### LOCAL IMPORTS
from readof_mod import readmesh_mod

print('###
    ------------------------------------------------------------------------------------------
     ###')
print('### ---------------------------------- COUPLING PROGRAM
    ---------------------------------- ###')
print('###    -- SCRIPT: coupling_wrfmean.py
          ###')
print('###       -- Extract and process WRFMEAN data and build boundary conditions to
    OpenFOAM  ###')
print('###
    ------------------------------------------------------------------------------------------
     ###\n')
print('Execution Begun.\n')

timer_start = time.time()

#####################################################################
#############                FUNCTIONS                ##############
#####################################################################

def StringParser(string, file_path):
    with open(file_path,'r') as f:
        for line in f.readlines():
            if string in line:
                return line.strip().split()[2]

def planar_trilinear_interpolation(x_WRF, y_WRF, z_WRF, VAR, Nx_WRF, Ny_WRF, Nz_WRF,
    x_OF, y_OF, z_OF, N_OF1, N_OF2):
    """ x_WRF, y_WRF, z_WRF    :: WRF X, Y and Z coordinates matrixes\n
        VAR                    :: WRF variable to interpolate\n
        Nx_WRF, Ny_WRF, Nz_WRF :: WRF mesh dimensions\n
        x_OF, y_OF, z_OF       :: OpenFOAM X, Y and Z coordinates lists (fluidfoam
    output as it is)\n
        N_OF1, N_OF            :: Target plane dimensions\n """

```

```
    ### WRF MESH MATRIXES TO VECTORS
    xvec = np.reshape(x_WRF,(Nx_WRF*Ny_WRF*Nz_WRF))
    yvec = np.reshape(y_WRF,(Nx_WRF*Ny_WRF*Nz_WRF))
    zvec = np.reshape(z_WRF,(Nx_WRF*Ny_WRF*Nz_WRF))

    ### WRF VARIABLE MATRIX TO VECTOR
    values = np.reshape(VAR,(Nx_WRF*Ny_WRF*Nz_WRF))

    ### CONCATENATE WRF MESH VECTORS
    points = np.stack((xvec, yvec, zvec)).T

    ### CONCATENATE OPENFOAM MESH VECTORS
    request = np.stack((x_OF, y_OF, z_OF)).T

    ### INTERPOLATION
    VAR_INTERP = griddata(points, values, request)

    ### HANDLE OPENFOAM POINTS OUTSIDE WRF DOMAIN
    outside_indices = np.isnan(VAR_INTERP)

    if np.any(outside_indices):
        # Points outside exist, perform extrapolation
        outside_points = request[outside_indices]
        inside_points = points

        # Linear regression for extrapolation, choice of a different method is possible
        regressor = LinearRegression()
        regressor.fit(inside_points, values)

        extrapolated_values = regressor.predict(outside_points)

        # Combine interpolated and extrapolated values
        VAR_INTERP[outside_indices] = extrapolated_values

    return np.reshape(VAR_INTERP, (N_OF1, N_OF2)).T

######################################################################
# O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O-O #
######################################################################

######################################################################
######################################################################
### LOCAL OPTIONS

time_index = 0
g = 9.81

######################################################################
######################################################################
### .CFG FILES READING ###

fPATH = '../../_preproc/_gsurf_WINDIE/preproc.cfg'
fPATH_2 = '../../_preproc/_preprocDict'
fPATH_3 = '../../_preproc/_gsurf_WINDIE/windie.cfg'
fPATH_4 = '../coupling.cfg'

# preproc.cfg
xcentre = float(StringParser('xcentre ', fPATH))
ycentre = float(StringParser('ycentre ', fPATH))
rot = float(StringParser('rot ', fPATH))
xmin = float(StringParser('xmin ', fPATH))
xmax = float(StringParser('xmax ', fPATH))
ymin = float(StringParser('ymin ', fPATH))
ymax = float(StringParser('ymax ', fPATH))

# _preprocDict
sky = float(StringParser('sky ',fPATH_2))
nz = int(StringParser('nz ',fPATH_2))

# windie.cfg
nig = int(StringParser('nig ', fPATH_3))
njg = int(StringParser('njg ', fPATH_3))

```

```
# coupling.cfg
force_utm_zone = bool(StringParser('force_zone ', fPATH_4))
utm_zone = int(StringParser('utm_zone ', fPATH_4))
wrf_file = StringParser('wrf_file ', fPATH_4)

######################################################################
######################################################################
### WRF FILE READING ###

print('-- Reading variables from WRFMEAN NETCDF (.nc) file --')
f = netCDF4.Dataset("../wrf_files/%s" %wrf_file)

# COORDINATES
xlat = f.variables['latm']
xlong = f.variables['lonm']

# VELOCITY
u = f.variables['umm']
v = f.variables['vmm']
w = f.variables['wmm']

# TURBULENCE KINETIC ENERGY
tke = f.variables['tkemm']

# Z values
zmm = f.variables['zmm']

print('-- WRF variables read --\n')

# MESOSCALE GRID
nim = int(f.dimensions['nimm1'].size)
njm = int(f.dimensions['njmm1'].size)
nkm = int(f.dimensions['nkm'].size) - 1

print('### MESOSCALE GRID DIMENSIONS :: wrfmean ###')
print('nim = %i' %nim)
print('njm = %i' %njm)
print('nkm = %i' %nkm)

### Force number of time steps
ntm = 1

######################################################################
######################################################################
### CONVERT LAT LONG TO UTM

### INIT.
xlongUTM = np.zeros([nim,njm])
xlatUTM = np.zeros([nim,njm])

XLAT = np.zeros([nim,njm])
XLONG = np.zeros([nim,njm])

XLAT[:,:] = xlat[time_index,:,:]
XLONG[:,:] = xlong[time_index,:,:]

print('\n-- Coverting LAT LONG coordinates to UTM --')
if force_utm_zone == True:
    print('>> Forcing utm zone to %i <<' %utm_zone)

for j, jj in zip( range(0,nim) , range(0,njm) ):
    for i, ii in zip( range(0,nim) , range(0,njm) ):
        if force_utm_zone == True:
            utm_tuple = utm.from_latlon(XLAT[ii,jj], XLONG[i,j],utm_zone)
        else:
            utm_tuple = utm.from_latlon(XLAT[ii,jj], XLONG[i,j])

        xlongUTM[i,j] = list(utm_tuple)[0]
        xlatUTM[i,j] = list(utm_tuple)[1]

print('-- Coverting LAT LONG coordinates to UTM --')

######################################################################
```

```
192 ######################################################################
193 ### WRF COORDINATES TRANSFORMATION
194
195 ### INIT.
196 xt = np.zeros([nim,nim])
197 yt = np.zeros([njm,njm])
198
199 xr = np.zeros([nim,nim])
200 yr = np.zeros([njm,njm])
201
202 ### ROT CALC.
203 rot = 180-((450-rot) % 360)  # DEG
204 rot = rot * np.pi/180        # TO RAD
205
206 ### TRANSLATION
207 for i in range(0,nim):
208     for j in range(0,njm):
209         xt[i,j] = xlongUTM[i,j] - xcentre
210         yt[i,j] = xlatUTM[i,j] - ycentre
211
212 ### ROTATION
213 xr[:,:] = xt[:,:]*np.cos(rot) - yt[:,:]*np.sin(rot)
214 yr[:,:] = xt[:,:]*np.sin(rot) + yt[:,:]*np.cos(rot)
215
216 ######################################################################
217 ######################################################################
218 ### ASSIGN WRFMEAN VARIABLES TO NUMPY MATRIXES
219
220 Um = np.zeros([nkm, nim, njm])
221 Vm = np.zeros([nkm, nim, njm])
222
223 Uvals = np.zeros([nkm, nim, njm])
224 Vvals = np.zeros([nkm, nim, njm])
225 Wvals = np.zeros([nkm, njm, nim])
226 kvals = np.zeros([nkm, njm, nim])
227 z = np.zeros([nkm, njm, nim])
228
229 Um[:,:,:] = u[1:,:,:]
230 Vm[:,:,:] = v[1:,:,:]
231 Wvals[:,:,:] = w[1:,:,:]
232 kvals[:,:,:] = tke[1:,:,:]
233 z[:,:,:] = zmm[1:,:,:]
234 kvals[:,:,:] = tke[1:,:,:]
235
236 ######################################################################
237 ######################################################################
238 ### U AND V VECTORS ROTATION
239
240 Uvals[:,:,:] = Um[:,:,:]*np.cos(rot) - Vm[:,:,:]*np.sin(rot)
241 Vvals[:,:,:] = Um[:,:,:]*np.sin(rot) + Vm[:,:,:]*np.cos(rot)
242
243 #%% CELL
244 ######################################################################
245 ######################################################################
246 #### READ OpenFOAM MESH (ALL BOUNDARIES)
247
248 print('\n-- Reading OpenFOAM Mesh --')
249 print('nig = %i' %nig)
250 print('njg = %i' %njg)
251 print('nz = %i \n' %nz)
252
253 sol_path = '../..'
254
255 ### WEST
256 print('-- Reading OpenFOAM Mesh for west boundary --')
257 xof, yof, zof, area = readmesh_mod('%s' %sol_path, boundary="west")
258 xof_west = np.reshape(xof, (njg-2,nz)).T
259 yof_west = np.reshape(yof, (njg-2,nz)).T
260 zof_west = np.reshape(zof, (njg-2,nz)).T
261 area_west = np.reshape(area, (njg-2,nz)).T
262
263 ### EAST
264 print('\n-- Reading OpenFOAM Mesh for east boundary --')
```

```
xof, yof, zof, area = readmesh_mod('%s' %sol_path, boundary="east")
xof_east = np.reshape(xof, (njg-2,nz)).T
yof_east = np.reshape(yof, (njg-2,nz)).T
zof_east = np.reshape(zof, (njg-2,nz)).T
area_east = np.reshape(area, (njg-2,nz)).T

### NORTH
print('\n-- Reading OpenFOAM Mesh for north boundary --')
xof, yof, zof, area = readmesh_mod('%s' %sol_path, boundary="north")
xof_north = np.reshape(xof, (nig-2,nz)).T
yof_north = np.reshape(yof, (nig-2,nz)).T
zof_north = np.reshape(zof, (nig-2,nz)).T
area_north = np.reshape(area, (nig-2,nz)).T

### SOUTH
print('\n-- Reading OpenFOAM Mesh for south boundary --')
xof, yof, zof, area = readmesh_mod('%s' %sol_path, boundary="south")
xof_south = np.reshape(xof, (nig-2,nz)).T
yof_south = np.reshape(yof, (nig-2,nz)).T
zof_south = np.reshape(zof, (nig-2,nz)).T
area_south = np.reshape(area, (nig-2,nz)).T

### SKY
print('\n-- Reading OpenFOAM Mesh for sky boundary --')
xof, yof, zof, area = readmesh_mod('%s' %sol_path, boundary="sky")
xof_top = np.reshape(xof, (nig-2,njg-2)).T
yof_top = np.reshape(yof, (nig-2,njg-2)).T
zof_top = np.reshape(zof, (nig-2,njg-2)).T
area_top = np.reshape(area, (nig-2,njg-2)).T

print('\nWest Area: %.9e' %(np.sum(area_west[:,:])))
print('East Area: %.9e' %(np.sum(area_east[:,:])))
print('North Area: %.9e' %(np.sum(area_north[:,:])))
print('South Area: %.9e' %(np.sum(area_south[:,:])))
print('Sky Area: %.9e\n' %(np.sum(area_top[:,:])))

#%% CELL
######################################################################
######################################################################
#### TRILINEAR INTERPOLATION

### INIT.
Uof_west = np.zeros([ntm,nz,njg-2])
Vof_west = np.zeros([ntm,nz,njg-2])
Wof_west = np.zeros([ntm,nz,njg-2])
kof_west = np.zeros([ntm,nz,njg-2])

Uof_east = np.zeros([ntm,nz,njg-2])
Vof_east = np.zeros([ntm,nz,njg-2])
Wof_east = np.zeros([ntm,nz,njg-2])
kof_east = np.zeros([ntm,nz,njg-2])

Uof_north = np.zeros([ntm,nz,nig-2])
Vof_north = np.zeros([ntm,nz,nig-2])
Wof_north = np.zeros([ntm,nz,nig-2])
kof_north = np.zeros([ntm,nz,nig-2])

Uof_south = np.zeros([ntm,nz,nig-2])
Vof_south = np.zeros([ntm,nz,nig-2])
Wof_south = np.zeros([ntm,nz,nig-2])
kof_south = np.zeros([ntm,nz,nig-2])

Uof_top = np.zeros([ntm,nig-2,njg-2])
Vof_top = np.zeros([ntm,nig-2,njg-2])
Wof_top = np.zeros([ntm,nig-2,njg-2])
kof_top = np.zeros([ntm,nig-2,njg-2])

### CONVERT 2D COORDINATE MATRIXES TO 3D
xr_matrix = np.zeros([nkm,njm,nim])
yr_matrix = np.zeros([nkm,njm,nim])

for k in range(0,nkm):
    xr_matrix[k,:,:] = xr[:,:]
```

```
338     yr_matrix[k,:,:] = yr[:,:]
339 
340 print("-- Performing trilinear interpolations ... ")
341 for t in range(0,ntm):
342     print("-- Time step = %i" %t)
343     print("   -- west")
344     Uof_west[t,:,:] = planar_trilinear_interpolation(xr_matrix, yr_matrix, z, Uvals
        [:,:,:],
345         nim, njm, nkm, xof_west, yof_west, zof_west, njg-2, nz)
346     Vof_west[t,:,:] = planar_trilinear_interpolation(xr_matrix, yr_matrix, z, Vvals
        [:,:,:],
347         nim, njm, nkm, xof_west, yof_west, zof_west, njg-2, nz)
348     Wof_west[t,:,:] = planar_trilinear_interpolation(xr_matrix, yr_matrix, z, Wvals
        [:,:,:],
349         nim, njm, nkm, xof_west, yof_west, zof_west, njg-2, nz)
350     kof_west[t,:,:] = planar_trilinear_interpolation(xr_matrix, yr_matrix, z, kvals
        [:,:,:],
351         nim, njm, nkm, xof_west, yof_west, zof_west, njg-2, nz)
352 
353     print("   -- east")
354     Uof_east[t,:,:] = planar_trilinear_interpolation(xr_matrix, yr_matrix, z, Uvals
        [:,:,:],
355         nim, njm, nkm, xof_east, yof_east, zof_east, njg-2, nz)
356     Vof_east[t,:,:] = planar_trilinear_interpolation(xr_matrix, yr_matrix, z, Vvals
        [:,:,:],
357         nim, njm, nkm, xof_east, yof_east, zof_east, njg-2, nz)
358     Wof_east[t,:,:] = planar_trilinear_interpolation(xr_matrix, yr_matrix, z, Wvals
        [:,:,:],
359         nim, njm, nkm, xof_east, yof_east, zof_east, njg-2, nz)
360     kof_east[t,:,:] = planar_trilinear_interpolation(xr_matrix, yr_matrix, z, kvals
        [:,:,:],
361         nim, njm, nkm, xof_east, yof_east, zof_east, njg-2, nz)
362 
363     print("   -- north")
364     Uof_north[t,:,:] = planar_trilinear_interpolation(xr_matrix, yr_matrix, z, Uvals
        [:,:,:],
365         nim, njm, nkm, xof_north, yof_north, zof_north, nig-2, nz)
366     Vof_north[t,:,:] = planar_trilinear_interpolation(xr_matrix, yr_matrix, z, Vvals
        [:,:,:],
367         nim, njm, nkm, xof_north, yof_north, zof_north, nig-2, nz)
368     Wof_north[t,:,:] = planar_trilinear_interpolation(xr_matrix, yr_matrix, z, Wvals
        [:,:,:],
369         nim, njm, nkm, xof_north, yof_north, zof_north, nig-2, nz)
370     kof_north[t,:,:] = planar_trilinear_interpolation(xr_matrix, yr_matrix, z, kvals
        [:,:,:],
371         nim, njm, nkm, xof_north, yof_north, zof_north, nig-2, nz)
372 
373     print("   -- south")
374     Uof_south[t,:,:] = planar_trilinear_interpolation(xr_matrix, yr_matrix, z, Uvals
        [:,:,:],
375         nim, njm, nkm, xof_south, yof_south, zof_south, nig-2, nz)
376     Vof_south[t,:,:] = planar_trilinear_interpolation(xr_matrix, yr_matrix, z, Vvals
        [:,:,:],
377         nim, njm, nkm, xof_south, yof_south, zof_south, nig-2, nz)
378     Wof_south[t,:,:] = planar_trilinear_interpolation(xr_matrix, yr_matrix, z, Wvals
        [:,:,:],
379         nim, njm, nkm, xof_south, yof_south, zof_south, nig-2, nz)
380     kof_south[t,:,:] = planar_trilinear_interpolation(xr_matrix, yr_matrix, z, kvals
        [:,:,:],
381         nim, njm, nkm, xof_south, yof_south, zof_south, nig-2, nz)
382 
383     print("   -- sky")
384     Uof_top[t,:,:] = planar_trilinear_interpolation(xr_matrix, yr_matrix, z, Uvals
        [:,:,:],
385         nim, njm, nkm, xof_top, yof_top, zof_top, nig-2, njg-2)
386     Vof_top[t,:,:] = planar_trilinear_interpolation(xr_matrix, yr_matrix, z, Vvals
        [:,:,:],
387         nim, njm, nkm, xof_top, yof_top, zof_top, nig-2, njg-2)
388     Wof_top[t,:,:] = planar_trilinear_interpolation(xr_matrix, yr_matrix, z, Wvals
        [:,:,:],
389         nim, njm, nkm, xof_top, yof_top, zof_top, nig-2, njg-2)
390     kof_top[t,:,:] = planar_trilinear_interpolation(xr_matrix, yr_matrix, z, kvals
        [:,:,:],
```

```
391         nim, njm, nkm, xof_top, yof_top, zof_top, nig-2, njg-2)
392
393 #%% CELL
394 ######################################################################
395 ######################################################################
396 #### FLUXES CALCULATIONS
397
398 ### INIT.
399 flux_west = np.zeros([ntm,nz,njg-2])
400 flux_east = np.zeros([ntm,nz,njg-2])
401 flux_north = np.zeros([ntm,nz,nig-2])
402 flux_south = np.zeros([ntm,nz,nig-2])
403 flux_top = np.zeros([ntm,nig-2,njg-2])
404
405 Ucorr_west = np.zeros([ntm,nz,njg-2])
406 Ucorr_east = np.zeros([ntm,nz,njg-2])
407
408 Vcorr_north = np.zeros([ntm,nz,nig-2])
409 Vcorr_south = np.zeros([ntm,nz,nig-2])
410
411 Wcorr_top = np.zeros([ntm,nig-2,njg-2])
412
413 print('\n -- Fluxes Correction --')
414
415 for t in range(0,ntm):
416     print('\n ------ Time step = %i ------' %t)
417
418     ### FLUXES CALC.
419     flux_west[t,:,:] = Uof_west[t,:,:]*area_west[:,:]
420     flux_east[t,:,:] = Uof_east[t,:,:]*area_east[:,:]
421     flux_north[t,:,:] = Vof_north[t,:,:]*area_north[:,:]
422     flux_south[t,:,:] = Vof_south[t,:,:]*area_south[:,:]
423     flux_top[t,:,:] = Wof_top[t,:,:]*area_top[:,:]
424
425     ### FLUXES CALC. (BOUNDARIES)
426     sumf_west = np.sum(flux_west[t,:,:])   ## ID 0
427     sumf_east = np.sum(flux_east[t,:,:])   ## ID 1
428     sumf_north = np.sum(flux_north[t,:,:]) ## ID 2
429     sumf_south = np.sum(flux_south[t,:,:]) ## ID 3
430     sumf_top = np.sum(flux_top[t,:,:])     ## ID 4
431
432     ### INITIAL ERROR
433     err_flux = sumf_west - sumf_east - sumf_north + sumf_south - sumf_top
434
435     ### MAX. FLUX
436     max_flux = max(abs(sumf_west), abs(sumf_east), abs(sumf_north), abs(sumf_south), abs
        (sumf_top))
437
438     print('\nWest Flux: %.9e' %(sumf_west))
439     print('East Flux: %.9e' %(sumf_east))
440     print('North Flux: %.9e' %(sumf_north))
441     print('South Flux: %.9e' %(sumf_south))
442     print('Sky Flux: %.9e\n' %(sumf_top))
443
444     print('Total Inflow: %.9e' %(sumf_west + sumf_south))
445     print('Total Outflow: %.9e\n' %(sumf_east + sumf_north + sumf_top))
446
447     print('Initial error : %.9f ' %err_flux)
448
449     ### F CALCULATION
450     if abs(max_flux - abs(sumf_west)) < 1e-9:
451         F = -(-sumf_east+sumf_south-sumf_north-sumf_top)/sumf_west
452         print('Correction Factor F = %.4f \n' %F)
453         Ucorr_west[t,:,:] = Uof_west[t,:,:]*F
454         print("Highest Flux is on West Boundary")
455     else:
456         Ucorr_west[t,:,:] = Uof_west[t,:,:]
457
458     if abs(max_flux - abs(sumf_east)) < 1e-9:
459         F = (sumf_west+sumf_south-sumf_north-sumf_top)/sumf_east
460         print('Correction Factor F = %.4f \n' %F)
461         Ucorr_east[t,:,:] = Uof_east[t,:,:]*F
462         print("Highest Flux is on East Boundary")
```

```
463     else:
464         Ucorr_east[t,:,:] = Uof_east[t,:,:]
465 
466     if abs(max_flux - abs(sumf_north)) < 1e-9:
467         F = (sumf_west-sumf_east+sumf_south-sumf_top)/sumf_north
468         print('Correction Factor F = %.4f \n' %F)
469         Vcorr_north[t,:,:] = Vof_north[t,:,:]*F
470         print("Highest Flux is on North Boundary")
471     else:
472         Vcorr_north[t,:,:] = Vof_north[t,:,:]
473 
474     if abs(max_flux - abs(sumf_south)) < 1e-9:
475         F = -(sumf_west-sumf_east-sumf_north-sumf_top)/sumf_south
476         print('Correction Factor F = %.4f \n' %F)
477         Vcorr_south[t,:,:] = Vof_south[t,:,:]*F
478         print("Highest Flux is on South Boundary")
479     else:
480         Vcorr_south[t,:,:] = Vof_south[t,:,:]
481 
482     if abs(max_flux - abs(sumf_top)) < 1e-9:
483         F = (sumf_west-sumf_east+sumf_south-sumf_north)/sumf_top
484         print('Correction Factor F = %.4f \n' %F)
485         Wcorr_top[t,:,:] = Wof_top[t,:,:]*F
486         print("Highest Flux is on Sky Boundary")
487     else:
488         Wcorr_top[t,:,:] = Wof_top[t,:,:]
489 
490     flux_west[t,:,:] = Ucorr_west[t,:,:]*area_west[:,:]
491     flux_east[t,:,:] = Ucorr_east[t,:,:]*area_east[:,:]
492     flux_north[t,:,:] = Vcorr_north[t,:,:]*area_north[:,:]
493     flux_south[t,:,:] = Vcorr_south[t,:,:]*area_south[:,:]
494     flux_top[t,:,:] = Wcorr_top[t,:,:]*area_top[:,:]
495 
496     sumf_west = np.sum(flux_west[t,:,:])   ## ID 0
497     sumf_east = np.sum(flux_east[t,:,:])   ## ID 1
498     sumf_north = np.sum(flux_north[t,:,:]) ## ID 2
499     sumf_south = np.sum(flux_south[t,:,:]) ## ID 3
500     sumf_top = np.sum(flux_top[t,:,:])     ## ID 4
501 
502     print('\nWest Flux: %.9e' %(sumf_west))
503     print('East Flux: %.9e' %(sumf_east))
504     print('North Flux: %.9e' %(sumf_north))
505     print('South Flux: %.9e' %(sumf_south))
506     print('Sky Flux: %.9e\n' %(sumf_top))
507 
508     print('Total Inflow: %.9e' %(sumf_west + sumf_south))
509     print('Total Outflow: %.9e\n' %(sumf_east + sumf_north + sumf_top))
510 
511     ### FINAL ERROR
512     err_flux = abs(sumf_west - sumf_east - sumf_north + sumf_south - sumf_top)
513     print('Final error : %.9f ' %err_flux)
514 
515 #%% CELL
516 ########################################################################
517 ########################################################################
518 ### WRITE BOUNDARY DATA FILES
519 
520 dt = int(24*3600/72)
521 
522 ###############
523 ### POINTS
524 print('\n-- Writing points files --')
525 
526 ### WEST ###
527 f = open('../../constant/boundaryData/west/points','w')
528 
529 f.write('// Points\n')
530 f.write('%i\n' %(nz*(njg-2)))
531 f.write('(\n\n')
532 f.write('// west boundary\n')
533 
534 for k in range(0,nz):
535     for j in range(0,njg-2):
```

```
536         f.write('(%.20f    %.20f    %.20f)\n' %(xmin,yof_west[k,j],zof_west[k,j]))
537
538 f.write('\n)\n\n')
539 f.write('// ********************************************************************************* //
        ')
540 f.close()
541
542 ### EAST ###
543 f = open('../../constant/boundaryData/east/points','w')
544
545 f.write('// Points\n')
546 f.write('%i\n' %(nz*(nig-2)))
547 f.write('(\n\n')
548 f.write('// east boundary\n')
549
550 for k in range(0,nz):
551     for j in range(0,njg-2):
552         f.write('(%.20f    %.20f    %.20f)\n' %(xmax,yof_east[k,j],zof_east[k,j]))
553
554 f.write('\n)\n\n')
555 f.write('// ********************************************************************************* //
        ')
556 f.close()
557
558 ### NORTH ###
559 f = open('../../constant/boundaryData/north/points','w')
560
561 f.write('// Points\n')
562 f.write('%i\n' %(nz*(nig-2)))
563 f.write('(\n\n')
564 f.write('// north boundary\n')
565
566 for k in range(0,nz):
567     for i in range(0,nig-2):
568         f.write('(%.20f    %.20f    %.20f)\n' %(xof_north[k,i],ymax,zof_north[k,i]))
569
570 f.write('\n)\n\n')
571 f.write('// ********************************************************************************* //
        ')
572 f.close()
573
574 ### SOUTH ###
575 f = open('../../constant/boundaryData/south/points','w')
576
577 f.write('// Points\n')
578 f.write('%i\n' %(nz*(nig-2)))
579 f.write('(\n\n')
580 f.write('// south boundary\n')
581
582 for k in range(0,nz):
583     for i in range(0,nig-2):
584         f.write('(%.20f    %.20f    %.20f)\n' %(xof_south[k,i],ymin,zof_south[k,i]))
585
586 f.write('\n)\n\n')
587 f.write('// ********************************************************************************* //
        ')
588 f.close()
589
590 ### SKY i.e. TOP ###
591 f = open('../../constant/boundaryData/sky/points','w')
592
593 f.write('// Points\n')
594 f.write('%i\n' %((nig-2)*(njg-2)))
595 f.write('(\n\n')
596 f.write('// sky boundary\n')
597
598 for i in range(0,nig-2):
599     for j in range(0,njg-2):
600         f.write('(%.20f    %.20f    %.20f)\n' %(xof_top[i,j],yof_top[i,j],sky))
601
602 f.write('\n)\n\n')
603 f.write('// ********************************************************************************* //
        ')
```

```
f.close()

print('-- Points files written --\n')

#%% CELL
######################################################################
######################################################################
### TIME VARYING MAPPED FIXED VALUES FOR U, V, W and k

sum_area_west = np.sum(area_west)
sum_area_east = np.sum(area_east)

print('-- Writing boundary data for velocity and turbulence kinetic energy --')

for t in range(0,ntm):

    tt = t*dt

    """###############
    #### WEST"""
    exist = os.path.exists('../../constant/boundaryData/west/%i' %(tt))
    if not exist:
        os.mkdir('../../constant/boundaryData/west/%i' %(tt))

    # VELOCITY
    f = open('../../constant/boundaryData/west/%i/U' %(tt),'w')

    f.write('// Data on points\n')
    f.write('%i\n' %(nz*(njg-2)))
    f.write('(\n\n')
    f.write('// velocity west boundary\n')

    for k in range(0,nz):
        for j in range(0,njg-2):
            f.write('(%.20f    %.20f    %.20f)\n' %(Ucorr_west[t,k,j],Vof_west[t,k,j],
    Wof_west[t,k,j]))
            #f.write('(%.20f    %.20f    %.20f)\n' %(1,0,0))

    f.write('\n)\n\n')
    f.write('//
    ********************************************************************** //')
    f.close()

    # TURBULENCE KINETIC ENERGY
    f = open('../../constant/boundaryData/west/%i/k' %(tt),'w')

    f.write('// Data on points\n')
    f.write('%i\n' %(nz*(njg-2)))
    f.write('(\n\n')
    f.write('// k west boundary\n')

    for k in range(0,nz):
        for j in range(0,njg-2):
            f.write('%.8f\n' %(kof_west[t,k,j]))

    f.write('\n)\n\n')
    f.write('//
    ********************************************************************** //')
    f.close()

    """###############
    #### EAST"""
    exist = os.path.exists('../../constant/boundaryData/east/%i' %(tt))
    if not exist:
        os.mkdir('../../constant/boundaryData/east/%i' %(tt))

    # VELOCITY
    f = open('../../constant/boundaryData/east/%i/U' %(tt),'w')

    f.write('// Data on points\n')
    f.write('%i\n' %(nz*(njg-2)))
    f.write('(\n\n')
    f.write('// velocity east boundary\n')
```

```
674
675     for k in range(0,nz):
676         for j in range(0,njg-2):
677             f.write('(%.20f    %.20f    %.20f)\n' %(Ucorr_east[t,k,j],Vof_east[t,k,j],
    Wof_east[t,k,j]))
678             #f.write('(%.12f    %.9f    %.8f)\n' %(sum_area_west/sum_area_east,0,0))
679
680     f.write('\n)\n\n')
681     f.write('//
    ********************************************************************** //')
682     f.close()
683
684     # TURBULENCE KINETIC ENERGY
685     f = open('../../constant/boundaryData/east/%i/k' %(tt),'w')
686
687     f.write('// Data on points\n')
688     f.write('%i\n' %(nz*(njg-2)))
689     f.write('(\n\n')
690     f.write('// k east boundary\n')
691
692     for k in range(0,nz):
693         for j in range(0,njg-2):
694             f.write('%.8f\n' %(kof_east[t,k,j]))
695
696     f.write('\n)\n\n')
697     f.write('//
    ********************************************************************** //')
698     f.close()
699
700     """###############
701     #### NORTH"""
702     exist = os.path.exists('../../constant/boundaryData/north/%i' %(tt))
703     if not exist:
704         os.mkdir('../../constant/boundaryData/north/%i' %(tt))
705
706     # VELOCITY
707     f = open('../../constant/boundaryData/north/%i/U' %(tt),'w')
708
709     f.write('// Data on points\n')
710     f.write('%i\n' %(nz*(njg-2)))
711     f.write('(\n\n')
712     f.write('// velocity north boundary\n')
713
714     for k in range(0,nz):
715         for i in range(0,nig-2):
716             f.write('(%.20f    %.20f    %.20f)\n' %(Uof_north[t,k,i],Vcorr_north[t,k,i],
    Wof_north[t,k,i]))
717             #f.write('(%.20f    %.20f    %.20f)\n' %(0,0,0))
718
719     f.write('\n)\n\n')
720     f.write('//
    ********************************************************************** //')
721     f.close()
722
723     # TURBULENCE KINETIC ENERGY
724     f = open('../../constant/boundaryData/north/%i/k' %(tt),'w')
725
726     f.write('// Data on points\n')
727     f.write('%i\n' %(nz*(nig-2)))
728     f.write('(\n\n')
729     f.write('// k north boundary\n')
730
731     for k in range(0,nz):
732         for i in range(0,nig-2):
733             f.write('%.8f\n' %(kof_north[t,k,i]))
734
735     f.write('\n)\n\n')
736     f.write('//
    ********************************************************************** //')
737     f.close()
738
739     """###############
740     #### SOUTH"""
```

```
741     exist = os.path.exists('../../constant/boundaryData/south/%i' %(tt))
742     if not exist:
743         os.mkdir('../../constant/boundaryData/south/%i' %(tt))
744 
745     # VELOCITY
746     f = open('../../constant/boundaryData/south/%i/U' %(tt),'w')
747 
748     f.write('// Data on points\n')
749     f.write('%i\n' %(nz*(nig-2)))
750     f.write('(\n\n')
751     f.write('// velocity south boundary\n')
752 
753     for k in range(0,nz):
754         for i in range(0,nig-2):
755             f.write('(%.20f    %.20f    %.20f)\n' %(Uof_south[t,k,i],Vcorr_south[t,k,i],
    Wof_south[t,k,i]))
756             #f.write('(%.20f    %.20f    %.20f)\n' %(0,0,0))
757 
758     f.write('\n)\n\n')
759     f.write('//
    ********************************************************************** //')
760     f.close()
761 
762     # TURBULENCE KINETIC ENERGY
763     f = open('../../constant/boundaryData/south/%i/k' %(tt),'w')
764 
765     f.write('// Data on points\n')
766     f.write('%i\n' %(nz*(nig-2)))
767     f.write('(\n\n')
768     f.write('// k south boundary\n')
769 
770     for k in range(0,nz):
771         for i in range(0,nig-2):
772             f.write('%.8f\n' %(kof_south[t,k,i]))
773 
774     f.write('\n)\n\n')
775     f.write('//
    ********************************************************************** //')
776     f.close()
777 
778     """###############
779     #### SKY"""
780     exist = os.path.exists('../../constant/boundaryData/sky/%i' %(tt))
781     if not exist:
782         os.mkdir('../../constant/boundaryData/sky/%i' %(tt))
783 
784     # VELOCITY
785     f = open('../../constant/boundaryData/sky/%i/U' %(tt),'w')
786 
787     f.write('// Data on points\n')
788     f.write('%i\n' %((nig-2)*(njg-2)))
789     f.write('(\n\n')
790     f.write('// velocity sky boundary\n')
791 
792     for i in range(0,nig-2):
793         for j in range(0,njg-2):
794             f.write('(%.20f    %.20f    %.20f)\n' %(Uof_top[t,i,j],Vof_top[t,i,j],
    Wcorr_top[t,i,j]))
795             #f.write('(%.20f    %.20f    %.20f)\n' %(0,0,0))
796 
797     f.write('\n)\n\n')
798     f.write('//
    ********************************************************************** //')
799     f.close()
800 
801     # TURBULENCE KINETIC ENERGY
802     f = open('../../constant/boundaryData/sky/%i/k' %(tt),'w')
803 
804     f.write('// Data on points\n')
805     f.write('%i\n' %((nig-2)*(njg-2)))
806     f.write('(\n\n')
807     f.write('// k sky boundary\n')
808 
```

```
809     for i in range(0,nig-2):
810         for j in range(0,njg-2):
811             f.write('%.8f\n' %(kof_top[t,i,j]))
812 
813     f.write('\n)\n\n')
814     f.write('//
        ********************************************************************** //')
815     f.close()
816 
817 print('-- Boundary data for velocity and turbulence kinetic energy written --\n')
818 
819 #%% CELL
820 ########################################################################
821 ########################################################################
822 ### WRITE p file
823 
824 pathp = "../../0/p"
825 
826 if max(abs(sumf_west), abs(sumf_east), abs(sumf_north), abs(sumf_south), abs(sumf_top))
        == abs(sumf_west):
827 
828     print("Corrected Highest Flux is on West Boundary, writting p file ...")
829 
830     f = open("%s" %pathp, "w")
831 
832     f.write("/*--------------------------------*- C++
        -*----------------------------------*\ \n")
833     f.write("| =========                 |
           | \n")
834     f.write("| \\\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
           | \n")
835     f.write("|  \\\    /   O peration     | Version:  2.1.1
           | \n")
836     f.write("|   \\\  /    A nd           | Web:      www.OpenFOAM.org
           | \n")
837     f.write("|    \\\/     M anipulation  |
           | \n")
838     f.write("
        \*---------------------------------------------------------------------------*/ \n")
839     f.write("FoamFile\n")
840     f.write("{\n")
841     f.write("    version     2.0;\n")
842     f.write("    format      ascii;\n")
843     f.write("    class       volScalarField;\n")
844     f.write("    object      p;\n")
845     f.write("}\n")
846     f.write("// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
        * //\n")
847     f.write("\n")
848     f.write("dimensions      [0 2 -2 0 0 0 0];\n")
849     f.write("\n")
850     f.write("internalField   uniform 0;\n")
851     f.write("\n")
852     f.write("boundaryField\n")
853     f.write("{\n")
854     f.write("    west\n")  ############## WEST ##############
855     f.write("    {\n")
856     f.write("        type            fixedValue;\n")
857     f.write("        value           uniform 0;\n")
858     f.write("    }\n")
859     f.write("\n")
860     f.write("    east\n") ############## EAST ##############
861     f.write("    {\n")
862     f.write("        type            zeroGradient;\n")
863     f.write("    }\n")
864     f.write("\n")
865     f.write("    south\n") ############## SOUTH ##############
866     f.write("    {\n")
867     f.write("        type            zeroGradient;\n")
868     f.write("    }\n")
869     f.write("\n")
870     f.write("    north\n") ############## NORTH ##############
871     f.write("    {\n")
```

```
872     f.write("        type            zeroGradient;\n")
873     f.write("    }\n")
874     f.write("\n")
875     f.write("    ground\n") ############## GROUND #############
876     f.write("    {\n")
877     f.write("        type            zeroGradient;\n")
878     f.write("    }\n")
879     f.write("\n")
880     f.write("    sky\n") ############## SKY #############
881     f.write("    {\n")
882     f.write("        type            zeroGradient;\n")
883     f.write("    }\n")
884     f.write("}\n")
885     f.write("\n")
886     f.write("//
        ********************************************************************** //\n")
887
888     f.close()
889
890 if max(abs(sumf_west), abs(sumf_east), abs(sumf_north), abs(sumf_south), abs(sumf_top))
        == abs(sumf_east):
891
892     print("Corrected Highest Flux is on East Boundary, writting p file ...")
893
894     f = open("%s" %pathp, "w")
895
896     f.write("/*--------------------------------*- C++
        -*----------------------------------*\ \n")
897     f.write("| =========                 |
           | \n")
898     f.write("| \\\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
           | \n")
899     f.write("|  \\\    /   O peration     | Version:  2.1.1
           | \n")
900     f.write("|   \\\  /    A nd           | Web:      www.OpenFOAM.org
           | \n")
901     f.write("|    \\\/     M anipulation  |
           | \n")
902     f.write("
        \*---------------------------------------------------------------------------*/ \n")
903     f.write("FoamFile\n")
904     f.write("{\n")
905     f.write("    version     2.0;\n")
906     f.write("    format      ascii;\n")
907     f.write("    class       volScalarField;\n")
908     f.write("    object      p;\n")
909     f.write("}\n")
910     f.write("// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
        * //\n")
911     f.write("\n")
912     f.write("dimensions      [0 2 -2 0 0 0 0];\n")
913     f.write("\n")
914     f.write("internalField   uniform 0;\n")
915     f.write("\n")
916     f.write("boundaryField\n")
917     f.write("{\n")
918     f.write("    west\n")  ############## WEST #############
919     f.write("    {\n")
920     f.write("        type            zeroGradient;\n")
921     f.write("    }\n")
922     f.write("\n")
923     f.write("    east\n") ############## EAST #############
924     f.write("    {\n")
925     f.write("        type            fixedValue;\n")
926     f.write("        value           uniform 0;\n")
927     f.write("    }\n")
928     f.write("\n")
929     f.write("    south\n") ############## SOUTH #############
930     f.write("    {\n")
931     f.write("        type            zeroGradient;\n")
932     f.write("    }\n")
933     f.write("\n")
934     f.write("    north\n") ############## NORTH #############
```

```
935     f.write("    {\n")
936     f.write("        type            zeroGradient;\n")
937     f.write("    }\n")
938     f.write("\n")
939     f.write("    ground\n") ############## GROUND ##############
940     f.write("    {\n")
941     f.write("        type            zeroGradient;\n")
942     f.write("    }\n")
943     f.write("\n")
944     f.write("    sky\n") ############# SKY #############
945     f.write("    {\n")
946     f.write("        type            zeroGradient;\n")
947     f.write("    }\n")
948     f.write("}\n")
949     f.write("\n")
950     f.write("//
        ********************************************************************** //\n")
951
952     f.close()
953
954 if max(abs(sumf_west), abs(sumf_east), abs(sumf_north), abs(sumf_south), abs(sumf_top))
        == abs(sumf_south):
955
956     print("Corrected Highest Flux is on South Boundary, writting p file ...")
957
958     f = open("%s" %pathp, "w")
959
960     f.write("/*--------------------------------*- C++
        -*----------------------------------*\ \n")
961     f.write("| =========                 |
           | \n")
962     f.write("| \\\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
            | \n")
963     f.write("|  \\\    /   O peration     | Version:  2.1.1
            | \n")
964     f.write("|   \\\  /    A nd           | Web:      www.OpenFOAM.org
            | \n")
965     f.write("|    \\\/     M anipulation  |
            | \n")
966     f.write("
        \*---------------------------------------------------------------------------*/ \n")
967     f.write("FoamFile\n")
968     f.write("{\n")
969     f.write("    version     2.0;\n")
970     f.write("    format      ascii;\n")
971     f.write("    class       volScalarField;\n")
972     f.write("    object      p;\n")
973     f.write("}\n")
974     f.write("// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
        * //\n")
975     f.write("\n")
976     f.write("dimensions      [0 2 -2 0 0 0 0];\n")
977     f.write("\n")
978     f.write("internalField   uniform 0;\n")
979     f.write("\n")
980     f.write("boundaryField\n")
981     f.write("{\n")
982     f.write("    west\n")  ############## WEST ##############
983     f.write("    {\n")
984     f.write("        type            zeroGradient;\n")
985     f.write("    }\n")
986     f.write("\n")
987     f.write("    east\n") ############## EAST ##############
988     f.write("    {\n")
989     f.write("        type            zeroGradient;\n")
990     f.write("    }\n")
991     f.write("\n")
992     f.write("    south\n") ############## SOUTH ##############
993     f.write("    {\n")
994     f.write("        type            fixedValue;\n")
995     f.write("        value           uniform 0;\n")
996     f.write("    }\n")
997     f.write("\n")
```

```
998     f.write("    north\n") ############## NORTH #############
999     f.write("    {\n")
1000    f.write("        type            zeroGradient;\n")
1001    f.write("    }\n")
1002    f.write("\n")
1003    f.write("    ground\n") ############## GROUND #############
1004    f.write("    {\n")
1005    f.write("        type            zeroGradient;\n")
1006    f.write("    }\n")
1007    f.write("\n")
1008    f.write("    sky\n") ############## SKY #############
1009    f.write("    {\n")
1010    f.write("        type            zeroGradient;\n")
1011    f.write("    }\n")
1012    f.write("}\n")
1013    f.write("\n")
1014    f.write("//
        ********************************************************************** //\n")
1015
1016    f.close()
1017
1018 if max(abs(sumf_west), abs(sumf_east), abs(sumf_north), abs(sumf_south), abs(sumf_top))
        == abs(sumf_north):
1019
1020    print("Corrected Highest Flux is on North Boundary, writting p file ...")
1021
1022    f = open("%s" %pathp, "w")
1023
1024    f.write("/*--------------------------------*- C++
        -*----------------------------------*\ \n")
1025    f.write("| =========                 |
           | \n")
1026    f.write("| \\\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
           | \n")
1027    f.write("|  \\\    /   O peration     | Version:  2.1.1
           | \n")
1028    f.write("|   \\\  /    A nd           | Web:      www.OpenFOAM.org
           | \n")
1029    f.write("|    \\\/     M anipulation  |
           | \n")
1030    f.write("
        \*---------------------------------------------------------------------------*/ \n")
1031    f.write("FoamFile\n")
1032    f.write("{\n")
1033    f.write("    version     2.0;\n")
1034    f.write("    format      ascii;\n")
1035    f.write("    class       volScalarField;\n")
1036    f.write("    object      p;\n")
1037    f.write("}\n")
1038    f.write("// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
        * //\n")
1039    f.write("\n")
1040    f.write("dimensions      [0 2 -2 0 0 0 0];\n")
1041    f.write("\n")
1042    f.write("internalField   uniform 0;\n")
1043    f.write("\n")
1044    f.write("boundaryField\n")
1045    f.write("{\n")
1046    f.write("    west\n")  ############## WEST #############
1047    f.write("    {\n")
1048    f.write("        type            zeroGradient;\n")
1049    f.write("    }\n")
1050    f.write("\n")
1051    f.write("    east\n") ############## EAST #############
1052    f.write("    {\n")
1053    f.write("        type            zeroGradient;\n")
1054    f.write("    }\n")
1055    f.write("\n")
1056    f.write("    south\n") ############## SOUTH #############
1057    f.write("    {\n")
1058    f.write("        type            zeroGradient;\n")
1059    f.write("    }\n")
1060    f.write("\n")
```

```
1061      f.write("    north\n") ############## NORTH #############
1062      f.write("    {\n")
1063      f.write("        type            fixedValue;\n")
1064      f.write("        value           uniform 0;\n")
1065      f.write("    }\n")
1066      f.write("\n")
1067      f.write("    ground\n") ############## GROUND #############
1068      f.write("    {\n")
1069      f.write("        type            zeroGradient;\n")
1070      f.write("    }\n")
1071      f.write("\n")
1072      f.write("    sky\n") ############# SKY #############
1073      f.write("    {\n")
1074      f.write("        type            zeroGradient;\n")
1075      f.write("    }\n")
1076      f.write("}\n")
1077      f.write("\n")
1078      f.write("//
         ********************************************************************** //\n")
1079
1080      f.close()
1081
1082  if max(abs(sumf_west), abs(sumf_east), abs(sumf_north), abs(sumf_south), abs(sumf_top))
         == abs(sumf_top):
1083
1084      print("Corrected Highest Flux is on Sky Boundary, writting p file ...")
1085
1086      f = open("%s" %pathp, "w")
1087
1088      f.write("/*--------------------------------*- C++
         -*----------------------------------*\ \n")
1089      f.write("| =========                 |
            | \n")
1090      f.write("| \\\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
             | \n")
1091      f.write("|  \\\    /   O peration     | Version:  2.1.1
             | \n")
1092      f.write("|   \\\  /    A nd           | Web:      www.OpenFOAM.org
             | \n")
1093      f.write("|    \\\/     M anipulation  |
             | \n")
1094      f.write("
         \*---------------------------------------------------------------------------*/ \n")
1095      f.write("FoamFile\n")
1096      f.write("{\n")
1097      f.write("    version     2.0;\n")
1098      f.write("    format      ascii;\n")
1099      f.write("    class       volScalarField;\n")
1100      f.write("    object      p;\n")
1101      f.write("}\n")
1102      f.write("// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
         * //\n")
1103      f.write("\n")
1104      f.write("dimensions      [0 2 -2 0 0 0 0];\n")
1105      f.write("\n")
1106      f.write("internalField   uniform 0;\n")
1107      f.write("\n")
1108      f.write("boundaryField\n")
1109      f.write("{\n")
1110      f.write("    west\n")  ############## WEST #############
1111      f.write("    {\n")
1112      f.write("        type            zeroGradient;\n")
1113      f.write("    }\n")
1114      f.write("\n")
1115      f.write("    east\n") ############# EAST #############
1116      f.write("    {\n")
1117      f.write("        type            zeroGradient;\n")
1118      f.write("    }\n")
1119      f.write("\n")
1120      f.write("    south\n") ############## SOUTH #############
1121      f.write("    {\n")
1122      f.write("        type            zeroGradient;\n")
1123      f.write("    }\n")
```

```
1124    f.write("\n")
1125    f.write("    north\n") ############## NORTH ##############
1126    f.write("    {\n")
1127    f.write("        type            zeroGradient;\n")
1128    f.write("    }\n")
1129    f.write("\n")
1130    f.write("    ground\n") ############## GROUND ##############
1131    f.write("    {\n")
1132    f.write("        type            zeroGradient;\n")
1133    f.write("    }\n")
1134    f.write("\n")
1135    f.write("    sky\n") ############## SKY ##############
1136    f.write("    {\n")
1137    f.write("        type            fixedValue;\n")
1138    f.write("        value           uniform 0;\n")
1139    f.write("    }\n")
1140    f.write("}\n")
1141    f.write("\n")
1142    f.write("//
       ********************************************************************************* //\n")
1143
1144    f.close()
1145
1146 timer_stop = time.time()
1147
1148 print("\nExecution Terminated.")
1149 print('Code total run time: %.2f s' %(timer_stop-timer_start))
1150
1151 # %%
```

D Mapas de topografia e rugosidade

D.1 Mapa de Topografia

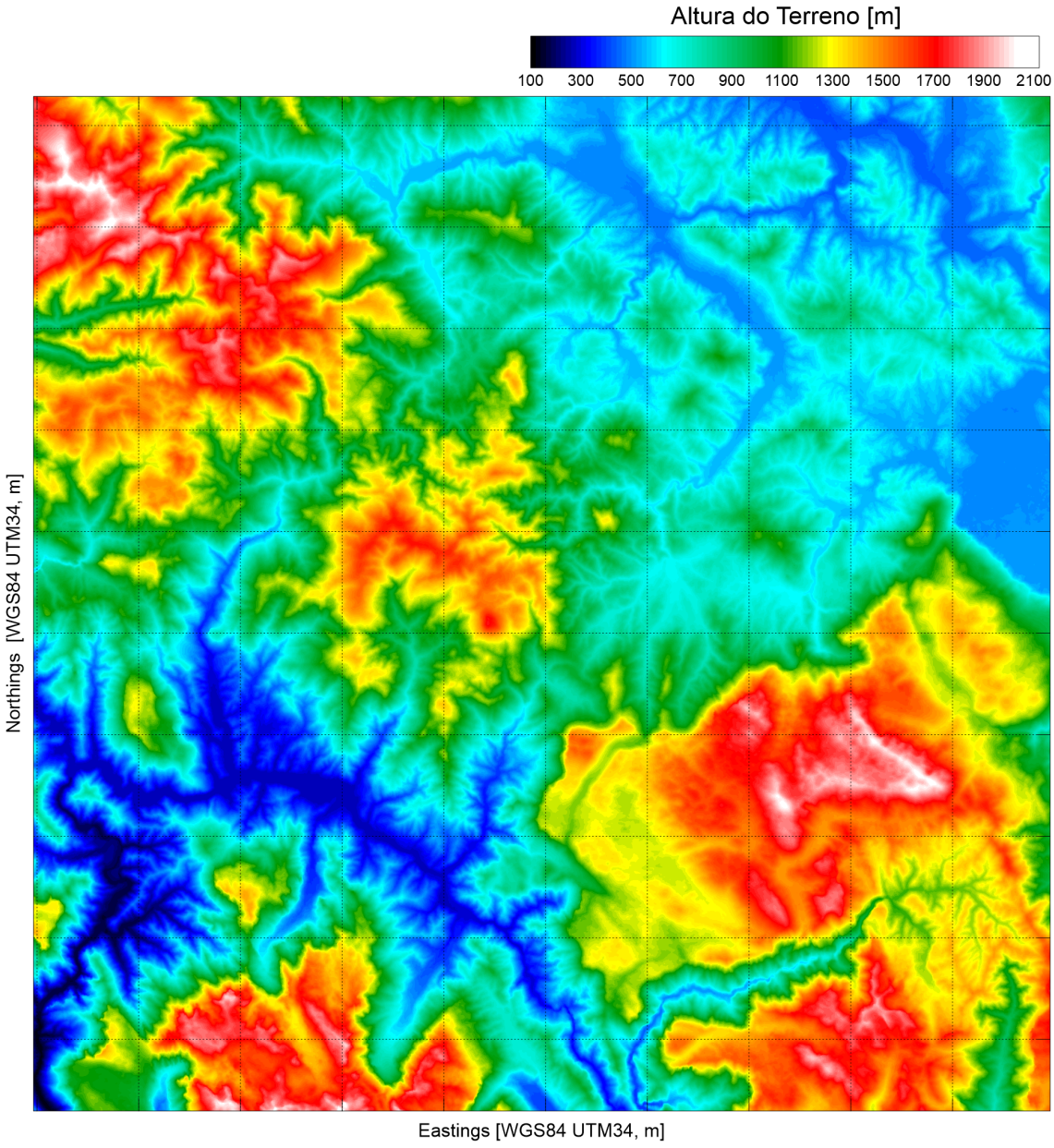


Figura D.1 – Mapa de topografia `topo.dat`

D.2 Mapa de Rugosidade

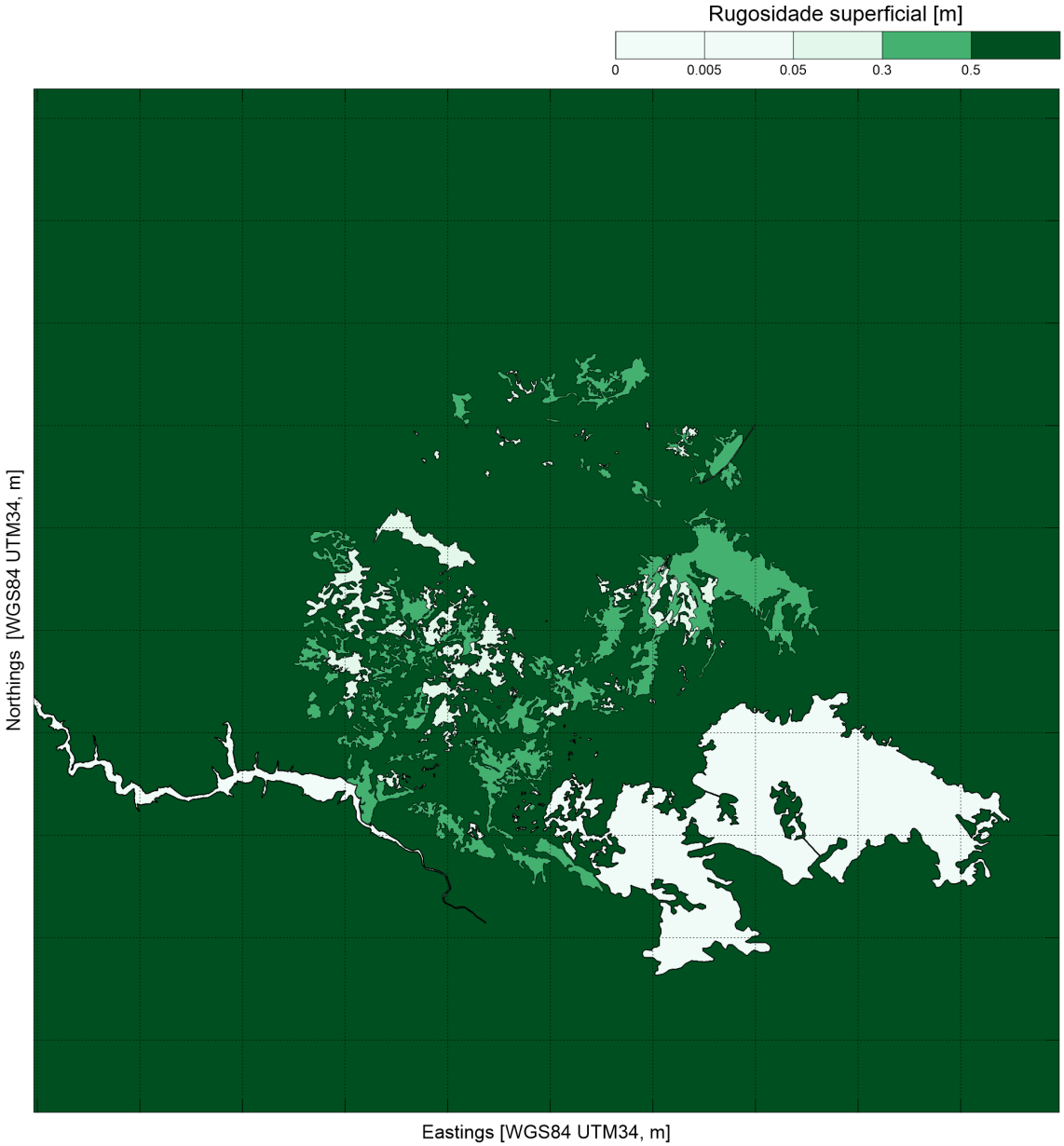


Figura D.2 – Mapa de rugosidade `roug.dat`