## 1 Contents
1.1 Contents ................................................................. 3

## 2 Community Guide
2.1 Community Guide ...................................................... 5

## 3 Modeling Communities
3.1 Wind Conditions ....................................................... 9

## 4 Indices and tables

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Contents</td>
<td>3</td>
</tr>
<tr>
<td>1.1 Contents</td>
<td>3</td>
</tr>
<tr>
<td>2 Community Guide</td>
<td>5</td>
</tr>
<tr>
<td>2.1 Community Guide</td>
<td>5</td>
</tr>
<tr>
<td>3 Modeling Communities</td>
<td>9</td>
</tr>
<tr>
<td>3.1 Wind Conditions</td>
<td>9</td>
</tr>
<tr>
<td>4 Indices and tables</td>
<td>63</td>
</tr>
</tbody>
</table>
WEMEP is a community project for the documentation of a Wind Energy Model Evaluation Protocol, developed within the IEA-Wind, with the following objectives:

- To develop an international framework to guide model developers and users on methodologies and best practices to conduct formal verification and validation (V&V).
- To promote collaboration between modeling communities and foster interdisciplinary research and development towards integrated models.
- To link to published open-access repositories of models, validation cases and data analysis scripts.

Modeling communities are welcomed to implement the MEP and contribute with open access repositories that can be interoperable with those from other communities.
1.1 Contents

Todo: Provide MEP content

1.1.1 Introduction
1.1.2 Scope and Objectives
1.1.3 Terminology
1.1.4 Quick Guide

Identify the Challenge
Scientific review
Find a Suitable Validation Database
Assess the Validation Hierarchy
A/B Testing
Quantify Performance for Intended Use
Identify Knowledge Gaps

1.1.5 Validation Directed Program

Setting goals based on quantities of interest and metrics
Defining priorities based on Phenomena Identification Ranking Table (PIRT) gap analysis

1.1.6 Verification

Code Verification
Solution Verification

1.1.7 Validation

Defining a validation strategy using a building-block hierarchy
Design of experiment
Setting up of a validation benchmark
Validation assessment at system level

1.1.8 Uncertainty Quantification

Aleatory and epistemic uncertainty
Sources of uncertainty
Experimental uncertainty
Computational model uncertainty

1.1.9 Blind Testing

Blind Testing
Model Calibration

1.1.10 Documenting

1.1.11 Data Provision

Data Provision
Licensing

1.1.12 References
2.1 Community Guide

This section describes the community behind WEMEP and provides guidance on how to contribute to sustain the project.

2.1.1 Community Sponsors

WEMEP is primarily developed under the umbrella of the IEA-Wind based on the following research Tasks:

- IEA Task 31 “Wakebench”: modeling external wind conditions and wind farm wakes
- IEA Task XX

with support from the following research projects:

- New European Wind Atlas (NEWA)
- etc

with institutional support from the following organizations:

- National Renewable Energy Centre (CENER), Spain
- National Renewable Energy Laboratory (NREL), USA
- Sandia National Laboratories (SNL), USA
- etc

2.1.2 How to Contribute
Todo: Community contribute content.

Guide on hot to support the project and how to contribute through github (installation guide, brief git tutorial, etc)

**TODO list**

Todo: Community contribute content.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/wemep/checkouts/latest/community/contribute.rst, line 4.)

Todo: Provide license.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/wemep/checkouts/latest/community/license.rst, line 4.)

Todo: Community support content.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/wemep/checkouts/latest/community/support.rst, line 4.)

Todo: Provide MEP content

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/wemep/checkouts/latest/mep/mep.rst, line 4.)

Todo: CWind conditions content.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/wemep/checkouts/latest/windconditions/windconditions.rst, line 4.)

### 2.1.3 Frequently Asked Questions

FAQ

### 2.1.4 Support

Todo: Community support content.

How to ask questions File an issue in github E-mail to helpdesk
2.1.5 Release Process and History

Release Process

Major releases Minor releases

Release History

1.0

2.1.6 License

Todo: Provide license.
CHAPTER 3

Modeling Communities

3.1 Wind Conditions

The IEA-Wind Task 31 “Wakebench” is rolling out the validation strategy for wind assessment models originating from the New European Wind Atlas (NEWA) project in compliance with the Wind Energy Model Evaluation Protocol (MEP). The validation strategy address the needs of the following wind energy applications:

- Wind resource assessment
- Wind turbine site suitability
- Numerical site calibration

3.1.1 MEP Implementation

External Wind Conditions

Todo: CWind conditions content.

Applications

Wind Resource Assessment

Scope and Objectives Quantities of Interest Quality Acceptance Criteria

Site Suitability

Scope and Objectives Quantities of Interest Quality Acceptance Criteria
Background

The GEWEX Atmospheric Boundary Layer Study (GABLS) series of benchmarks have been developed to improve the representation of the atmospheric boundary layer in regional and large-scale atmospheric models. The model intercomparison studies have been organized for single-column models (SCM) and large-eddy simulation models (LES). The cases are based on observations over flat terrain in the Artic, Kansas (USA), Cabauw (The Netherlands) and Dome C (Antarctica).

GABLS1 simulated a quasi-steady stable boundary layer resulting from 9 hours of uniform surface cooling [1][2][3]. GABLS2 simulated a diurnal cycle, still with idealized forcing, by simplifying measurements from the CASES-99 experiment in Kansas [4][5]. GABLS3 simulated a real diurnal case with a strong nocturnal low-level jet (LLJ) at the Cabauw met tower in the Netherlands [6][7][8][9][10]. In GABLS4, the aim is to study the interaction of a boundary layer with strong stability in relatively simple surface coupling characteristics.

Challenges

The challenges of stable boundary layers and diurnal cycles are reviewed byHotlslag et al. [11], notably: the relation between enhanced mixing in operational weather models performance, investigate the role of land-surface heterogeneity in the coupling with the atmosphere, develop LES models with interactive land-surface schemes, create a climatology of boundary-layer parameters (stability classes, boundary-layer depth, and surface fluxes) and develop parameterizations for the very stable boundary layer when turbulence is not the dominant driver. These challenges are ultimately shared by wind energy applications that are embeded in atmospheric models.

Benchmarks

The GABLS benchmarks are revisited in Windbench as baseline for the design of microscale wind farm flow models that progressively incorporate more complex atmospheric physics. Wind energy specific quantities of interest are used in order to focus the model evaluation on the application of interest.

The GABLS activities are coordinated within the GEWEX Modelling and Prediction Panel (GMPP). The cases have been througly documented in the literature and in the following websites:

- **GABLS1**: SCM, LES
- **GABLS2**
- **GABLS3**: SCM, LES
• GABLS4

In the context of developing a downscaling methodology to link mesoscale wind climate processes and microscale wind farm flow models, the GABLS3 case is particularly relevant. GABLS 1 and 2 can be considered as precursor cases dealing with turbulence modelling of the atmospheric boundary layer under idealized forcing conditions. These cases are suitable for the design of RANS-based SCMs by comparison with LES simulations, that have shown high consistency [3][5].

GABLS3 incorporates realistic forcing by using mesoscale model simulations and synoptic stations [6][7][8][9][10]. This benchmark has been revisited by the wind energy community in the context of the NEWA project to develop meso-micro methodologies for wind resource assessment applications:

**GABLS3 Diurnal Cycle in Flat Terrain Leading to a Nocturnal Low-Level Jet**

Javier Sanz Rodrigo, CENER

June 2017

**Status**

The GABLS3 benchmark carried out jointly in the NEWA, Wakebench and MesoWake projects finished in June 2017. You can find the results and data open-access in the following links:

- **Observational data**: [http://projects.knmi.nl/gabls/evaluation.html](http://projects.knmi.nl/gabls/evaluation.html)
- **Simulation data**: [http://doi.org/10.23728/b2share.f5d5a492d8aa4b7998b70abd68f5eae4](http://doi.org/10.23728/b2share.f5d5a492d8aa4b7998b70abd68f5eae4)
- **Evaluation scripts and docker image**: [https://github.com/windbench/gabls3](https://github.com/windbench/gabls3)

Additionally, you can also find individual evaluation results and data from CFDWind1D:

- **Simulation data**: [http://doi.org/10.23728/b2share.22e419b663cb4ffca8107391b6716c1b](http://doi.org/10.23728/b2share.22e419b663cb4ffca8107391b6716c1b)

**Notebooks**

**GABLS 3 Diurnal-Cycle Benchmark for Wind Energy Applications**

Javier Sanz Rodrigo, National Renewable Energy Centre (CENER), Sarriguren, Spain, jsrodrigo@cener.com

May 2017
Introduction

This benchmark report analyzes the simulations submitted to the GABLS3 benchmark for wind energy, organized within the NEWA project and the IEA Task 31 Wakebench. The results of this benchmark have been published in:


Benchmark Set-Up

Background information and benchmark set-up can be found in: http://windbench.net/gabls-3

Simulations

The following simulations participate in the benchmark. Notice that WRF-YSU (ref) is the reference mesoscale simulation that was used to generate the mesoscale tendencies that are used as forcings for the microscale models. Additional WRF simulations have been run to test the sensitivity of mesoscale simulations to input forcing and planetary boundary layer (PBL) scheme. The ensemble mean of these WRF simulations is also used in the model intercomparison.

Table 1. Summary of model simulations. Monin Obukhov similarity theory (Monin-Obukhov (MOST)) surface boundary conditions use either heat flux (:math:`'H'`), 2-m (:math:`'T_2'`) or skin temperature (:math:`'T_{SK}'`) from WRF.

<table>
<thead>
<tr>
<th>Name</th>
<th>Input</th>
<th>Turbulence</th>
<th>z-Levels</th>
<th>Surface B.C.</th>
</tr>
</thead>
<tbody>
<tr>
<td>WRF-YSU (ref)</td>
<td>ERA-Interim</td>
<td>YSU</td>
<td>46</td>
<td>Noah</td>
</tr>
<tr>
<td>WRF</td>
<td>ERA-Interim, GFS</td>
<td>MJY, MYNN, QNSE, TEMF, YSU</td>
<td>46</td>
<td>Noah</td>
</tr>
<tr>
<td>WRF-YSU_LES</td>
<td>ERA-Interim</td>
<td>LES-TKE</td>
<td>101</td>
<td>Noah</td>
</tr>
<tr>
<td>WRF-VentosM_ke</td>
<td>ERA-Interim</td>
<td>YSU/k – ( \epsilon )</td>
<td>70</td>
<td>MOST, ( H )</td>
</tr>
<tr>
<td>CFDWind1D_ke</td>
<td>WRF (ref)</td>
<td>( k – \epsilon )</td>
<td>301</td>
<td>MOST, ( T_2 )</td>
</tr>
<tr>
<td>Alya-CFDWind1D_ke</td>
<td>WRF (ref)</td>
<td>( k – \epsilon )</td>
<td>500</td>
<td>MOST, ( T_2 )</td>
</tr>
<tr>
<td>Ellipsys1D_ke</td>
<td>WRF (ref)</td>
<td>( k – \epsilon )</td>
<td>512</td>
<td>MOST, ( T_2 )</td>
</tr>
<tr>
<td>Ellipsys3D_ke</td>
<td>WRF (ref)</td>
<td>( k – \epsilon )</td>
<td>192</td>
<td>MOST, ( T_{SK} )</td>
</tr>
<tr>
<td>Ellipsys3D_LES</td>
<td>WRF (ref)</td>
<td>Smagorinsky</td>
<td>128</td>
<td>MOST, ( T_{SK} )</td>
</tr>
<tr>
<td>SP-Wind_LES</td>
<td>WRF (ref)</td>
<td>LES-TKE</td>
<td>500</td>
<td>MOST, ( T_2 )</td>
</tr>
</tbody>
</table>

Load libraries and define input data

```python
import matplotlib as plt
import numpy as np
from scipy import spatial
import datetime
import netCDF4
import pandas as pd
from scipy import interpolate
import statsmodels.api as sm
```

(continues on next page)
import os

# Constants
K = 0.41  # von Karman constant
g = 9.81  # [m s-2]
R_air = 287.058  # Specific gas constant for dry air [J kg-1 K-1]
Cp_air = 1005  # Specific heat of air [J kg-1 K-1]
P0 = 100000.  # Reference pressure [Pa]

# Site ID
siteID = 'GABLS3'
lat_s = 51.971  # degrees N
lon_s = 4.927  # degrees E

datefrom = datetime.datetime(2006, 7, 1, 12, 0, 0)  # Origin of evaluation period
dateto = datetime.datetime(2006, 7, 2, 12, 0, 0)  # End of evaluation period
Hhub = 120.  # hub-height
Drot = 160.  # rotor diameter
ts = 10  # sampling frequency to evaluate [min]

# Mesoscale tendencies averaging settings
tav = 60.0  # Time averaging time used in simulations [min]
Lav = 9000.0  # Spatial averaging [m]
ws = int(tav/ts)  # rolling window size

Load Simulation data

[12]: dirsim = '
filesim = [dirsim + '/GABLS3_tendencies_d02_YSU_w60_L9000.nc',
dirsim + '/WRF-LES_d8.nc',
dirsim + '/VentosM.nc',
dirsim + '/CFDWindSCM_ke.nc',
dirsim + '/Alya-CFDWind1D.nc',
dirsim + '/Ellipsys1D_TskWRF.nc',
dirsim + '/Ellipsys3D_TskWRF.nc',
dirsim + '/Ellipsys3D_LES_TskWRF.nc',
dirsim + '/SP-Wind.nc',
dirsim + '/WRF-YSU_GFS.nc',
dirsim + '/WRF-YSU ERA.nc',
dirsim + '/WRF-TEMP_GFS.nc',
dirsim + '/WRF-TEMP ERA.nc',
dirsim + '/WRF-QNSE_GFS.nc',
dirsim + '/WRF-QNSE ERA.nc',
dirsim + '/WRF-MYNN_GFS.nc',
dirsim + '/WRF-MYNN ERA.nc',
dirsim + '/WRF-MJY_GFS.nc',
dirsim + '/WRF-MJY ERA.nc',
]
simID = ['WRF-YSU (ref)', 'WRF-YSULES', 'VentosM',
'CFDWind1D_ke', 'Alya-CFDWind1D_ke', 'Ellipsys1D_ke',
']

(continues on next page)
'Ellipsys3D_ke', 'Ellipsys3D_LES', 'SP-Wind_LES',
'WRF-YSU_GFS', 'WRF-YSU_ERA',
'WRF-TEMF_GFS', 'WRF-TEMF_ERA',
'WRF-QNSE_GFS', 'WRF-QNSE_ERA',
'WRF-MYNN_GFS', 'WRF-MYNN_ERA',
'WRF-MJY_GFS', 'WRF-MJY_ERA',
]

simtype = ['meso', 'micro', 'micro',
            'micro', 'micro', 'micro',
            'micro', 'micro',
            'meso', 'meso', 'meso',
            'meso', 'meso', 'meso',
            'meso', 'meso', 'meso',
            'meso', 'meso',
            ]

Nsim = len(filesim)

# Create a list with the simulation datasets
for isim in range(0, Nsim):
    f = netCDF4.Dataset(filesim[isim], 'r')
    times = f.variables['time'][:]
    idates = np.where(np.logical_and(times >= mdates.date2num(datefrom), times <
   mdates.date2num(dateto)))[0]
    t.append(pd.DataFrame(f.variables['time'][idates], index = mdates.num2date(f.
   variables['time'][idates])).resample(str(ts)+'Min').mean())
    U.append(pd.DataFrame(f.variables['U'][idates, :], index = mdates.num2date(f.
   variables['time'][idates])).resample(str(ts)+'Min').mean().rolling(window = ws).
   mean())
    V.append(pd.DataFrame(f.variables['V'][idates, :], index = mdates.num2date(f.
   variables['time'][idates])).resample(str(ts)+'Min').mean().rolling(window = ws).
   mean())
    Th.append(pd.DataFrame(f.variables['Th'][idates, :], index = mdates.num2date(f.
   variables['time'][idates])).resample(str(ts)+'Min').mean().rolling(window = ws).
   mean())
    z.append(f.variables['z'][:])
    S.append((U[isim]**2 + V[isim]**2)**0.5)
    WD.append(180 + np.arctan2(U[isim], V[isim])*180/np.pi)
    us.append(pd.DataFrame(f.variables['ust'][idates], index = mdates.num2date(f.
   variables['time'][idates])).resample(str(ts)+'Min').mean().rolling(window = ws).
   mean())
    wt.append(pd.DataFrame(f.variables['wt'][idates], index = mdates.num2date(f.
   variables['time'][idates])).resample(str(ts)+'Min').mean().rolling(window = ws).
   mean())
    T2.append(pd.DataFrame(f.variables['T2'][idates], index = mdates.num2date(f.
   variables['time'][idates])).resample(str(ts)+'Min').mean().rolling(window = ws).
   mean())
    if simtype[isim] == 'micro':
        TKE.append(pd.DataFrame(f.variables['TKE'][idates, :], index = mdates.
    num2date(f.variables['time'][idates])).resample(str(ts)+'Min').mean().
    rolling(window = ws).mean())

(continues on next page)
else:
    TKE.append(pd.DataFrame(data=None, columns=U[isim].columns, index=U[isim].index))
    f.close()

# Create ensemble from WRF simulations
simID.append('WRF-Ensemble')
simtype.append('meso')
Nsim = Nsim + 1
U.append(pd.DataFrame(data=np.mean(np.dstack(U[9:18]), axis = 2), columns=U[18].columns, index=U[18].index))
V.append(pd.DataFrame(data=np.mean(np.dstack(V[9:18]), axis = 2), columns=U[18].columns, index=U[18].index))
S.append((U[len(U)-1]**2 + V[len(U)-1]**2)**0.5)
WD.append(180 + np.arctan2(U[len(U)-1],V[len(U)-1])*180/np.pi)
Th.append(pd.DataFrame(data=np.mean(np.dstack(Th[9:18]), axis = 2), columns=U[18].columns, index=U[18].index))
TKE.append(pd.DataFrame(data=np.mean(np.dstack(TKE[9:18]), axis = 2), columns=U[18].columns, index=U[18].index))
us.append(pd.DataFrame(data=np.mean(np.dstack(us[9:18]), axis = 1), index=U[18].index))
wj.append(pd.DataFrame(data=np.mean(np.dstack(wj[9:18]), axis = 1), index=U[18].index))
T2.append(pd.DataFrame(data=np.mean(np.dstack(T2[9:18]), axis = 1), index=U[18].index))
t.append(t[18])
z.append(z[18])

Load Observations

[13]: # Note that the file 'gabls3_scm_cabauw_obs_v33.nc' can be obtained from the KNMI GABLS3 website
    # http://projects.knmi.nl/gabls/gabls3_scm_cabauw_obs_v33.nc
    # Alternatively, it is also provided in the Windbench/GABLS3 input dataset
    dirobs = '.'
    fileobs = dirobs + '/gabls3_scm_cabauw_obs_v33.nc'
    nodata = -9999.0 # missing data flag
    date0 = datetime.datetime(2006,7,1,0,0,0) # origin of time_obs
    f1 = netCDF4.Dataset(fileobs, 'r')
    dates_obs = f1.variables['date'][:]
    time_obs = f1.variables['time'][:] # 'hours since 2006-07-01 00:00:00 0:00'
    date_obs = []
    for i in range (0,len(time_obs)):
        date_obs.append(date0 + datetime.timedelta(seconds = np.int(time_obs[i]*3600.0)))
    iffrom_obs=0
    for j in range(0,len(date_obs)):
        if date_obs[j] <= datefrom:
            iffrom_obs = j
    ito_obs=0
    for j in range(0,len(date_obs)):
if date_obs[j] <= dateto:
    ito_obs = j+1

date_obs = mdates.date2num(date_obs[ifrom_obs:ito_obs])
time_obs = 24.0*(date_obs - date_obs[0])

# Surface variables
ustar_obs = pd.DataFrame(np.ma.array(f1.variables['ustar'][ifrom_obs:ito_obs], mask=(f1.variables['ustar'][ifrom_obs:ito_obs] == nodata)), index = mdates.num2date(date_obs))
U10_obs = pd.DataFrame(np.ma.array(f1.variables['u10m'][ifrom_obs:ito_obs], mask=(f1.variables['u10m'][ifrom_obs:ito_obs] == nodata)), index = mdates.num2date(date_obs))
V10_obs = pd.DataFrame(np.ma.array(f1.variables['v10m'][ifrom_obs:ito_obs], mask=(f1.variables['v10m'][ifrom_obs:ito_obs] == nodata)), index = mdates.num2date(date_obs))
S10_obs = pd.DataFrame(np.ma.array(f1.variables['f10m'][ifrom_obs:ito_obs], mask=(f1.variables['f10m'][ifrom_obs:ito_obs] == nodata)), index = mdates.num2date(date_obs))
WD10_obs = pd.DataFrame(np.ma.array(f1.variables['d10m'][ifrom_obs:ito_obs], mask=(f1.variables['d10m'][ifrom_obs:ito_obs] == nodata)), index = mdates.num2date(date_obs))
T2_obs = pd.DataFrame(np.ma.array(f1.variables['t2m'][ifrom_obs:ito_obs], mask=(f1.variables['t2m'][ifrom_obs:ito_obs] == nodata)), index = mdates.num2date(date_obs))
HFX_obs = pd.DataFrame(np.ma.array(f1.variables['shf'][ifrom_obs:ito_obs], mask=(f1.variables['shf'][ifrom_obs:ito_obs] == nodata)), index = mdates.num2date(date_obs))
Ts_obs = pd.DataFrame(np.ma.array(f1.variables['ts'][ifrom_obs:ito_obs], mask=(f1.variables['ts'][ifrom_obs:ito_obs] == nodata)), index = mdates.num2date(date_obs))
zs_obs = f1.variables['zs'][ifrom_obs:ito_obs][0,:]
# soil temperature heights [m]
Tsk_obs = pd.DataFrame(np.ma.array(f1.variables['tsk'][ifrom_obs:ito_obs], mask=(f1.variables['tsk'][ifrom_obs:ito_obs] == nodata)), index = mdates.num2date(date_obs))
Tsk1_obs = pd.DataFrame(np.ma.array(f1.variables['tsk1'][ifrom_obs:ito_obs], mask=(f1.variables['tsk1'][ifrom_obs:ito_obs] == nodata)), index = mdates.num2date(date_obs))

# Vertical profiles
zf_obs = f1.variables['zf'][ifrom_obs:ito_obs][0,:]
# velocity profile heights [m]
zT_obs = f1.variables['zt'][ifrom_obs:ito_obs][0,:]
# temperature profile heights [m]
U_obs = pd.DataFrame(np.ma.array(f1.variables['u'][ifrom_obs:ito_obs], mask=(f1.variables['u'][ifrom_obs:ito_obs] == nodata)), index = mdates.num2date(date_obs))
V_obs = pd.DataFrame(np.ma.array(f1.variables['v'][ifrom_obs:ito_obs], mask=(f1.variables['v'][ifrom_obs:ito_obs] == nodata)), index = mdates.num2date(date_obs))
S_obs = pd.DataFrame(np.ma.array(f1.variables['f'][ifrom_obs:ito_obs], mask=(f1.variables['f'][ifrom_obs:ito_obs] == nodata)), index = mdates.num2date(date_obs))
WD_obs = pd.DataFrame(np.ma.array(f1.variables['d'][ifrom_obs:ito_obs], mask=(f1.variables['d'][ifrom_obs:ito_obs] == nodata)), index = mdates.num2date(date_obs))
Th_obs = pd.DataFrame(np.ma.array(f1.variables['th'][ifrom_obs:ito_obs], mask=(f1.variables['th'][ifrom_obs:ito_obs] == nodata)), index = mdates.num2date(date_obs))
q_obs = pd.DataFrame(np.ma.array(f1.variables['q'][ifrom_obs:ito_obs], mask=(f1.variables['q'][ifrom_obs:ito_obs] == nodata)), index = mdates.num2date(date_obs))

# fluxes
zh_obs = f1.variables['zh'][ifrom_obs:ito_obs][0,:]
# fluxes heights
wt_obs = pd.DataFrame(np.ma.array(f1.variables['wt'][ifrom_obs:ito_obs], mask=(f1.variables['wt'][ifrom_obs:ito_obs] == nodata)), index = mdates.num2date(date_obs))
uw_obs = pd.DataFrame(np.ma.array(f1.variables['uw'][ifrom_obs:ito_obs], mask=(f1.variables['uw'][ifrom_obs:ito_obs] == nodata)), index = mdates.num2date(date_obs))
vw_obs = pd.DataFrame(np.ma.array(f1.variables['vw'][ifrom_obs:ito_obs], mask=(f1.variables['vw'][ifrom_obs:ito_obs] == nodata)), index = mdates.num2date(date_obs))
us_obs = (uw_obs**2 + vw_obs**2)**0.25

# Obukhov length
T0_obs = np.tile(Th_obs[36].values,(5,1)).T
L_obs = -us_obs**3/(K*(g/T0_obs)*wt_obs)

# Resample and apply tav to observations
ws = int(tav/ts)  # rolling window size
ustar_obs_w = ustar_obs.resample(str(ts)+'Min').mean().bfill().rolling(window = ws).mean()
HFX_obs_w = HFX_obs.resample(str(ts)+'Min').mean().bfill().rolling(window = ws).mean()
U10_obs_w = U10_obs.resample(str(ts)+'Min').mean().bfill().rolling(window = ws).mean()
V10_obs_w = V10_obs.resample(str(ts)+'Min').mean().bfill().rolling(window = ws).mean()
S10_obs_w = S10_obs.resample(str(ts)+'Min').mean().bfill().rolling(window = ws).mean()
WD10_obs_w = WD10_obs.resample(str(ts)+'Min').mean().bfill().rolling(window = ws).mean()
T2_obs_w = T2_obs.resample(str(ts)+'Min').mean().bfill().rolling(window = ws).mean()
Ts_obs_w = Ts_obs.resample(str(ts)+'Min').mean().bfill().rolling(window = ws).mean()
U_obs_w = U_obs.resample(str(ts)+'Min').mean().bfill().rolling(window = ws).mean()
V_obs_w = V_obs.resample(str(ts)+'Min').mean().bfill().rolling(window = ws).mean()
S_obs_w = S_obs.resample(str(ts)+'Min').mean().bfill().rolling(window = ws).mean()
WD_obs_w = WD_obs.resample(str(ts)+'Min').mean().bfill().rolling(window = ws).mean()
Th_obs_w = Th_obs.resample(str(ts)+'Min').mean().bfill().rolling(window = ws).mean()
q_obs_w = q_obs.resample(str(ts)+'Min').mean().bfill().rolling(window = ws).mean()
wt_obs_w = wt_obs.resample(str(ts)+'Min').mean().bfill().rolling(window = ws).mean()
uw_obs_w = uw_obs.resample(str(ts)+'Min').mean().bfill().rolling(window = ws).mean()
vw_obs_w = vw_obs.resample(str(ts)+'Min').mean().bfill().rolling(window = ws).mean()
us_obs_w = us_obs.resample(str(ts)+'Min').mean().bfill().rolling(window = ws).mean()
L_obs_w = L_obs.resample(str(ts)+'Min').mean().bfill().rolling(window = ws).mean()
TKE_obs_w = pd.DataFrame(data=None, columns=U_obs_w.columns,index=U_obs_w.index)

Plot tz contours

[14]: # List simulations for index reference
for isim in range(0,len(simID)):
    print isim, simID[isim]

0 WRF-YSU (ref)
1 WRF-YSU_LES
2 VentosM
3 CFDWind1D_ke
4 Alya-Wind1D_ke
5 Ellipsys1D_ke
6 Ellipsys3D_ke
7 Ellipsys3D_LES
8 SP-Wind_LES
9 WRF-YSU_GFS
10 WRF-YSU_ERA
11 WRF-TEMF_GFS
12 WRF-TEMF_ERA
13 WRF-QNSE_GFS
14 WRF-QNSE_ERA
15 WRF-MYNN_GFS
16 WRF-MYNN_ERA

3.1. Wind Conditions
# Select the simulations you want to plot
plotsim = np.array([0, 19, 1, 2, 3, 4, 5, 6, 7, 8])

# Plot settings
zlim = 2000  # height limit [m]
Zcmap = plt.get_cmap('jet')  # colormap
figname = siteID + '_tzfields.png'  # output filename
Zlevels = np.array([np.linspace(0, 16, 17),
                     np.linspace(30, 210, 13),
                     np.linspace(285, 305, 11),
                     np.linspace(0, 5, 11)])  # range for TKE
X = []; Y = []; Z = []; zaxis = []; Ztitle = []
Z = [S_obs_w.T, WD_obs_w.T, Th_obs_w.T, TKE_obs_w.T]
[Xf, Yf] = np.meshgrid(date_obs, zf_obs)
[XT, YT] = np.meshgrid(date_obs, zT_obs)
X = [Xf, Xf, XT, Xf]
Y = [Yf, Yf, YT, Yf]
taxis = [date_obs, date_obs, date_obs, date_obs]
zaxis = [zf_obs, zf_obs, zT_obs, zf_obs]
Ztitle.append('Observations')
for ip in range(0, len(plotsim)):
  isim = plotsim[ip]
  Z.append(S[isim].T)
  [X0, Y0] = np.meshgrid(t[isim], z[isim])
  taxis.append(t[isim]); zaxis.append(z[isim]); X.append(X0); Y.append(Y0)
  Z.append(WD[isim].T)
  [X0, Y0] = np.meshgrid(t[isim], z[isim])
  taxis.append(t[isim]); zaxis.append(z[isim]); X.append(X0); Y.append(Y0)
  Z.append(Th[isim].T)
  [X0, Y0] = np.meshgrid(t[isim], z[isim])
  taxis.append(t[isim]); zaxis.append(z[isim]); X.append(X0); Y.append(Y0)
  Z.append(TKE[isim].T)
  [X0, Y0] = np.meshgrid(t[isim], z[isim])
  taxis.append(t[isim]); zaxis.append(z[isim]); X.append(X0); Y.append(Y0)
  Ztitle.append(simID[isim])

Zlabel = ('$S$ [$m s^{-1}$']', '$WD$ ['+'u'\N{DEGREE SIGN}']', '$\Theta$ [$K$]', '$TKE$ [\rightarrow $m^{2} s^{-2}$]')
taxis_label = 'UTC time in hours since ' + datefrom.strftime('%Y-%m-%d %H:%M') + ', L_{avg} = ' + '%.0f' % (Lav/1000) + ' km, t_{avg} = ' + '%.0f' % (tav) + ' min'
zaxis_label = '$z$ [$m$]
hoursFmt = mdates.DateFormatter('’%H’')
Zlevels = np.tile(Zlevels, len(Ztitle))

hoursFmt = mdates.DateFormatter('’%H’')
tticks = [datefrom + datetime.timedelta(hours = 3*x) for x in range(0, 9)]

fig, ax = plt.subplots(len(Ztitle), 4, sharex='col', sharey='row', figsize=(11, 17), dpi=300)
xrotor = np.array([mdates.date2num(datefrom), mdates.date2num(dateto)])
yrotor1 = np.array([Hhub - 0.5*Drot, Hhub - 0.5*Drot])
yrotor2 = np.array([Hhub + 0.5*Drot, Hhub + 0.5*Drot])
cbleft = np.array([0.125,0.33,0.53,0.73])

for iax in range (0,len(Ztitle)*4):
    ix, iy = np.unravel_index(iax,(len(Ztitle),4))
    CF0 = ax[ix,iy].contourf(X[ix],Y[ix],Z[ix], Zlevels[ix], cmap=Zcmap)
    ax[ix,iy].plot(xrotor,yrotor1,'--k')
    ax[ix,iy].plot(xrotor,yrotor2,'--k')
    ax[ix,iy].set_ylim([10, zlim])
    ax[ix,iy].set_yscale('log')
    ax[ix,iy].xaxis.set_major_formatter(hoursFmt)
    ax[ix,iy].set_xticks(tticks)
    if iy == 0:
        ax[ix,iy].set_ylabel(zaxis_label)
    if iy == 3:
        ax[ix,iy].yaxis.set_label_position("right")
        ax[ix,iy].set_ylabel(Ztitle[ix], fontsize = 9, fontweight='bold')
    if ix == len(Ztitle)-1:
        fig.subplots_adjust(top=0.87)
        cbar_ax = fig.add_axes([cbleft[iy],0.885, 0.165, 0.012])
        cbar = fig.colorbar(CF0, cax=cbar_ax, orientation='horizontal')
        cbar.ax.set_xlabel(Zlabel[iy],labelpad = -42, x = 0.5)

ax[len(Ztitle)-1,1].set_xlabel(taxis_label,x=1.2);
plt.setp([a.get_xticklabels() for a in ax[0, :]], visible=False)
plt.setp([a.get_yticklabels() for a in ax[:, 1]], visible=False)

3.1. Wind Conditions

from IPython.display import Markdown, display
figcaption = ("**Fig 5. Time-height contour of horizontal wind speed $S$, direction $\rightarrow WD$, potential temperature $\Theta$ and turbulent kinetic energy $TKE$. Dotted lines denote a rotor diameter of 160 m at 120 m hub-height.**")
display(Markdown(figcaption))
Fig 5. Time-height contour of horizontal wind speed \( S \), direction \( WD \), potential temperature \( \Theta \) and turbulent kinetic energy \( TKE \). Dotted lines denote a rotor diameter of 160 m at 120 m hub-height.

Vertical Profiles

```python
# Specify the datetime at which you want to plot the vertical profiles
t0 = datetime.datetime(2006,7,2,0,0,0)

# Plot settings
linespec = ['k-','b.-','r.-','c.-','m-','g-','y-','c-','b-','r--']
lwidth = np.array([2,1,1,1,1,1,1,1,2,2])
Nm = 3  # marker every

Z_obs = (S_obs_w.loc[t0].values, WD_obs_w.loc[t0].values,
         Th_obs_w.loc[t0].values)
z_obs = (zf_obs, zf_obs, zT_obs)

Z_sim = []; z_sim = []
for iplot in range(0,len(plotsim)):
    isim = plotsim[iplot]
    Z_sim.append((S[isim].loc[t0].values, WD[isim].loc[t0].values,
                  Th[isim].loc[t0].values))
    z_sim.append((z[isim], z[isim], z[isim]))

figname = siteID+'_' +t0.strftime('%Y-%m-%d_%H')+'_profiles.png'
zlim = 1000
fig,ax = plt.subplots(1, 3, sharey='row', figsize=(8,6))
for iax in range(0,3):
    ax[iax].plot(Z_obs[iax], z_obs[iax], 'ok', color='grey', label = 'obs ')
    for iplot in range(0,len(plotsim)):
        if (isim == 0 and iax == 2):
            aa = []
        else:
            ax[iax].plot(Z_sim[iplot][iax][1:], z_sim[iplot][iax][1:],
                         linespec[iplot], linewidth = lwidth[iplot], label = simID[isim],
                         markevery=Nm)
    #ax[iax].set_yscale('log')
    xlim = ax[iax].get_xlim()
    ax[iax].plot(xlim,yrotor1,'--k')
    ax[iax].plot(xlim,yrotor2,'--k')
    ax[iax].set_ylim([1., zlim])
    ax[iax].set_yticks(np.linspace(0,zlim,5))
    ax[iax].grid(which='major',color='k',linestyle=':')

#ax[0].set_xlim([0, 16])
ax[1].set_xlim([60, 180])
ax[2].set_xlim([285, 300])
ax[0].set_xlabel('$S$ \ [$m \ s^{-1}$]')
ax[1].set_xlabel('$WD$ \ [°]')
ax[2].set_xlabel(r'$\Theta$ \ [$K$]')
ax[0].set_ylabel('$z$ \ [$m$]')
ax[0].set_title('$t_{0}$ = ' + t0.strftime('%Y-%m-%d %H:%M:%S'))
```

(continues on next page)
Quantities of Interest and Metrics

The performance of the models is based on quantities of interest relevant for wind energy applications. These quantities are evaluated across a reference rotor span of 160 m, between 40 and 200 m, characteristic of an 8-MW large wind turbine. Besides hub-height wind speed $S_{hub}$ and direction $WD_{hub}$, it is relevant to consider the rotor-equivalent wind speed $REW_{S}$, the turbulence intensity (not evaluated here), the wind speed shear $\alpha$, and the wind direction shear or veer $\psi$.

The $REW_{S}$ is especially suitable to account for wind shear in wind turbine power performance tests [14]. The $REW_{S}$ is the wind speed corresponding to the kinetic energy flux through the swept rotor area, when accounting for the
vertical shear:

\[ \text{REW}_S = \left[ \frac{1}{A} \sum_i (A_i S_i^3 \cos \beta_i) \right]^{1/3} \]

where \( A \) is the rotor area and \( A_i \) are the horizontal segments that separate vertical measurement points of horizontal wind speed \( S_i \) across the rotor plane. The \( \text{REW}_S \) is here weighted by the cosine of the angle \( \beta_i \) of the wind direction \( W_D_i \) with respect to the hub-height wind direction to account for the effect of wind veer [15].

Wind shear is defined by fitting a power-law curve across the rotor wind speed points \( S_i \):

\[ S_i = S_{hub} \left( \frac{z_i}{z_{hub}} \right)^\alpha \]

Similarly, wind veer is defined as the slope \( \psi \) of the linear fit of the wind direction difference:

\[ \beta_i = \psi (z_i - z_{hub}) \]

To evaluate simulations and measurements consistently, these quantities are obtained after resampling, by linear interpolation, velocity and wind direction vertical profiles at 10 points across the rotor area and then computing the \( \text{REW}_S \) and the shear functional fits. While these fitting functions are commonly used in wind energy, their suitability in LLJ conditions is questionable. The regression coefficient from the fitting can be used to determine this suitability.

A rolling average with a window size of one hour is applied to simulation and observational data to remove the impact of high frequency fluctuations in the analysis.

Validation results are quantified in terms of the mean absolute error (MAE):

\[ \text{MAE} = \frac{1}{N} \sum_{i=1}^{N} |\chi_{pred} - \chi_{obs}| \]

where \( \chi \) is any of the above mentioned quantities of interest, predicted (\( \text{pred} \)) or observed (\( \text{obs} \)), and \( N \) is the number of samples evaluated in the time series.

```python
# Define rotor-based quantities of interest
zrot = np.linspace(Hhub - 0.5*Drot, Hhub + 0.5*Drot, 1 + Drot/10, endpoint=True)
def rotor(z,Sz,WDz):
    # Rotor QoIs [m s-1]
    # z: heights [m] where the velocity and wind direction are known spanning the rotor diameter
    # Sz, WDz: Wind speed [m s-1] and direction [deg from N] at z levels [tdim,zdim]
    # Returns:
    # REWS: rotor equivalent wind speed [m s-1]
    # alpha: wind shear, power-law exponent from linear fit lg(U/Uhub) = \alpha \log(z/zhub)
    # alpha_R2: R-squared from least squares fit to linear function
    # veer: wind veer, slope of linear function beta = WDz - WDhub = veer*(z - zhub)
    # veer_R2: R-squared from least squares fit to linear function
    tdim = Sz.shape[0]
zdim = Sz.shape[1]
Rrot = 0.5*(z[-1]-z[0])
Hhub = 0.5*(z[-1]+z[0])
ihub = int(0.5*len(zrot))
Arotor = np.pi*(Rrot)**2
Uz = -Sz*np.sin(np.pi*WDz/180.)
Vz = -Sz*np.cos(np.pi*WDz/180.)
Shub = Sz[:,ihub]
(continues on next page)
```

3.1. Wind Conditions 23
WDhub = WDz[:,ihub]
def cz(x,R,H):
    return 2.*(R**2 - (x-H)**2)**0.5
sumA = np.zeros((Sz.shape[0]))
veer = np.zeros((Sz.shape[0]))
for i in range(0,zdim-1):
    Ai = integrate.quad(cz, z[i], z[i+1], args = (Rrot,Hhub))
    Si = 0.5*(Sz[:,i+1]+Sz[:,i])
    Ui = 0.5*(Uz[:,i+1]+Uz[:,i])
    Vi = 0.5*(Vz[:,i+1]+Vz[:,i])
    WDi = 180. + np.arctan2(Ui,Vi)*180./np.pi
    betai = WDi - WDhub
    sumA = sumA + Ai[0]*(Si*np.cos(np.pi*betai/180.))**3
REWS = (sumA/Arotor)**(1./3.)
alpha = np.zeros(tdim); alpha_stderr = np.zeros(tdim); alpha_R2 = np.zeros(tdim)
veer = np.zeros(tdim); veer_stderr = np.zeros(tdim); veer_R2 = np.zeros(tdim)
for it in range(0,tdim):
    regmodel = sm.OLS(np.log(Sz[it,:]/Shub[it]), np.log(z/Hhub))
    results = regmodel.fit()
    alpha[it] = results.params[0]
    alpha_stderr[it] = results.bse[0]
    alpha_R2[it] = results.rsquared
    regmodel = sm.OLS(WDz[it,:] - WDhub[it], z - Hhub)
    results = regmodel.fit()
    veer[it] = results.params[0]
    veer_stderr[it] = results.bse[0]
    veer_R2[it] = results.rsquared
return REWS, Shub, WDhub, alpha, alpha_R2, veer, veer_R2

Srews_obs = interpolate.interp2d(zf_obs[30:34],date_obs,S_obs_w.values[:,30:34])(zrot,
                                 date_obs)
Urews_obs = interpolate.interp2d(zf_obs[30:34],date_obs,U_obs_w.values[:,30:34])(zrot,
                                 date_obs)
Vrews_obs = interpolate.interp2d(zf_obs[30:34],date_obs,V_obs_w.values[:,30:34])(zrot,
                                 date_obs)
WDrews_obs = 180. + np.arctan2(Urews_obs,Vrews_obs)*180./np.pi
REWS_obs, Shub_obs, WDhub_obs, alpha_obs, alpha_R2_obs, veer_obs, veer_R2_obs = rotor(zrot,Srews_obs,WDrews_obs)

REWS = []; Shub = []; WDhub = []
alpha = []; alpha_R2 = []; veer = []; veer_R2 = []
for isim in range(0,Nsim):
    Srews_sim = interpolate.interp2d(z[isim],np.ravel(t[isim].values),S[isim].values)
    Urews_sim = interpolate.interp2d(z[isim],np.ravel(t[isim].values),U[isim].values)
    Vrews_sim = interpolate.interp2d(z[isim],np.ravel(t[isim].values),V[isim].values)
    WDrews_sim = 180. + np.arctan2(Urews_sim,Vrews_sim)*180./np.pi
    REWS0, Shub0, WDhub0, alpha0, alpha_R20, veer0, veer_R20 = rotor(zrot,Srews_sim,
                                                                 WDrews_sim)

(continues on next page)
REWS.append(REWS0)
Shub.append(Shub0)
WDhub.append(WDhub0)
alpha.append(alpha0)
alpha_R2.append(alpha_R20)
veer.append(veer0)
veer_R2.append(veer_R20)

[18]: # plot QoIs
figname = siteID+'_QoIs_rotor.png'
taxis_label = 'UTC time in hours since ' + datefrom.strftime('%Y-%m-%d %H:%M') + ', $\Delta D$ = ' + '
fig, ax = plt.subplots(2, 2, sharex='col', figsize=(9,6), dpi=300)
for iplot in range(0,len(plotsim)):
from IPython.display import display
pd.options.display.float_format = '{:,.2f}'.format  # format for output data in tables

# Compute error metrics
def mae(xtrue, xpred, norm):
    # Normalized Mean Absolute Error
    # xtrue: series with true values
    # xpred: series with predicted values
    nonan = ~np.isnan(xtrue)
    if norm == 'True':
        xtrue0 = xtrue[nonan] / np.mean(xtrue[nonan])
        xpred0 = xpred[nonan] / np.mean(xtrue[nonan])
    else:
        xtrue0 = xtrue[nonan]
        xpred0 = xpred[nonan]
N = len(xtrue0)
abserr = np.abs(xpred0 - xtrue0)
MAE = np.sum(abserr)/N
return MAE

# Errors with respecto to observations
Metrics = pd.DataFrame(np.zeros((Nsim, 5)),
columns = ['REWS', 'Shub', 'WDhub', 'alpha', 'veer'],
index = simID)
Metrics.loc['units',:] = np.array(['m s-1', 'm s-1', 'deg', '-', '-'])
for isim in range(0,Nsim):
    Metrics.loc[(simID[isim]),'REWS'] = mae(REWS_obs,REWS[isim],'False')
    Metrics.loc[(simID[isim]),'Shub'] = mae(Shub_obs,Shub[isim],'False')
    Metrics.loc[(simID[isim]),'WDhub'] = mae(WDhub_obs,WDhub[isim],'False')
    Metrics.loc[(simID[isim]),'alpha'] = mae(alpha_obs,alpha[isim],'False')
    Metrics.loc[(simID[isim]),'veer'] = mae(veer_obs,veer[isim],'False')

Table 2a. $MAE_{obs}$ with respect to observations

<table>
<thead>
<tr>
<th></th>
<th>REWS</th>
<th>Shub</th>
<th>WDhub</th>
<th>alpha</th>
<th>veer</th>
</tr>
</thead>
<tbody>
<tr>
<td>WRF-YSU (ref)</td>
<td>1.26</td>
<td>1.35</td>
<td>10.49</td>
<td>0.13</td>
<td>0.07</td>
</tr>
<tr>
<td>WRF-YSU_LES</td>
<td>1.51</td>
<td>1.60</td>
<td>10.67</td>
<td>0.15</td>
<td>0.06</td>
</tr>
<tr>
<td>VentosM</td>
<td>1.56</td>
<td>1.59</td>
<td>10.74</td>
<td>0.12</td>
<td>0.06</td>
</tr>
<tr>
<td>CFDWindID_ke</td>
<td>1.56</td>
<td>1.62</td>
<td>11.49</td>
<td>0.15</td>
<td>0.06</td>
</tr>
<tr>
<td>Alya-WindID_ke</td>
<td>1.48</td>
<td>1.42</td>
<td>11.30</td>
<td>0.14</td>
<td>0.07</td>
</tr>
<tr>
<td>Ellipsys1D_ke</td>
<td>1.37</td>
<td>1.50</td>
<td>11.51</td>
<td>0.16</td>
<td>0.06</td>
</tr>
<tr>
<td>Ellipsys3D_ke</td>
<td>1.36</td>
<td>1.52</td>
<td>10.61</td>
<td>0.16</td>
<td>0.06</td>
</tr>
<tr>
<td>Ellipsys3D_LES</td>
<td>1.38</td>
<td>1.37</td>
<td>11.90</td>
<td>0.18</td>
<td>0.09</td>
</tr>
<tr>
<td>SP-Wind_LES</td>
<td>1.47</td>
<td>1.38</td>
<td>8.79</td>
<td>0.13</td>
<td>0.07</td>
</tr>
<tr>
<td>WRF-YSU_GFS</td>
<td>1.33</td>
<td>1.30</td>
<td>15.69</td>
<td>0.14</td>
<td>0.07</td>
</tr>
<tr>
<td>WRF-YSU_ERA</td>
<td>1.51</td>
<td>1.51</td>
<td>14.45</td>
<td>0.14</td>
<td>0.08</td>
</tr>
<tr>
<td>WRF-TEMF_GFS</td>
<td>1.23</td>
<td>1.36</td>
<td>11.62</td>
<td>0.11</td>
<td>0.10</td>
</tr>
<tr>
<td>WRF-TEMF_ERA</td>
<td>1.50</td>
<td>1.57</td>
<td>16.69</td>
<td>0.10</td>
<td>0.07</td>
</tr>
<tr>
<td>WRF-QNSE_GFS</td>
<td>1.35</td>
<td>1.32</td>
<td>13.21</td>
<td>0.10</td>
<td>0.06</td>
</tr>
<tr>
<td>WRF-QNSE_ERA</td>
<td>1.63</td>
<td>1.62</td>
<td>14.15</td>
<td>0.10</td>
<td>0.07</td>
</tr>
<tr>
<td>WRF-MYNN_GFS</td>
<td>1.21</td>
<td>1.16</td>
<td>18.27</td>
<td>0.15</td>
<td>0.09</td>
</tr>
<tr>
<td>WRF-MYNN_ERA</td>
<td>1.25</td>
<td>1.19</td>
<td>17.62</td>
<td>0.16</td>
<td>0.09</td>
</tr>
<tr>
<td>WRF-MJY_GFS</td>
<td>1.25</td>
<td>1.26</td>
<td>17.00</td>
<td>0.13</td>
<td>0.06</td>
</tr>
<tr>
<td>WRF-MJY_ERA</td>
<td>1.29</td>
<td>1.22</td>
<td>16.46</td>
<td>0.14</td>
<td>0.06</td>
</tr>
<tr>
<td>WRF-Ensemble</td>
<td>1.16</td>
<td>1.12</td>
<td>14.24</td>
<td>0.11</td>
<td>0.07</td>
</tr>
</tbody>
</table>

units m s-1 m s-1 deg - -

Table 1. $MAE_{obs}$ with respect to observations

3.1. Wind Conditions
Metrics2.loc[(simID[isim]),'WDhub'] = mae(WDhub[refsim],WDhub[isim],'False')
Metrics2.loc[(simID[isim]),'alpha'] = mae(alpha[refsim],alpha[isim],'False')
Metrics2.loc[(simID[isim]),'veer'] = mae(veer[refsim],veer[isim],'False')

Tablecaption = ("**Table 2b: $MAE_{ref}$ with respect to **" +simID[refsim])
display(Markdown(tablecaption))
display(Metrics2)

**Table 2b: $MAE_{ref}$ with respect to **WRF-YSU (ref)

<table>
<thead>
<tr>
<th></th>
<th>REWS</th>
<th>Shub</th>
<th>WDhub</th>
<th>alpha</th>
<th>veer</th>
</tr>
</thead>
<tbody>
<tr>
<td>WRF-YSU (ref)</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>WRF-YSU_LES</td>
<td>0.45</td>
<td>0.54</td>
<td>4.09</td>
<td>0.05</td>
<td>0.03</td>
</tr>
<tr>
<td>VentosM</td>
<td>0.69</td>
<td>0.72</td>
<td>6.34</td>
<td>0.05</td>
<td>0.03</td>
</tr>
<tr>
<td>CFDWindID_ke</td>
<td>0.63</td>
<td>0.68</td>
<td>6.22</td>
<td>0.09</td>
<td>0.05</td>
</tr>
<tr>
<td>Alya-WindID_ke</td>
<td>0.73</td>
<td>0.70</td>
<td>2.90</td>
<td>0.07</td>
<td>0.02</td>
</tr>
<tr>
<td>EllipsysID_ke</td>
<td>0.55</td>
<td>0.61</td>
<td>5.22</td>
<td>0.10</td>
<td>0.04</td>
</tr>
<tr>
<td>Ellipsys3D_ke</td>
<td>0.66</td>
<td>0.74</td>
<td>5.17</td>
<td>0.09</td>
<td>0.04</td>
</tr>
<tr>
<td>Ellipsys3D_LES</td>
<td>1.13</td>
<td>1.27</td>
<td>11.99</td>
<td>0.09</td>
<td>0.06</td>
</tr>
<tr>
<td>SP-Wind_LES</td>
<td>0.63</td>
<td>0.69</td>
<td>3.25</td>
<td>0.04</td>
<td>0.02</td>
</tr>
<tr>
<td>WRF-YSU_GFS</td>
<td>0.59</td>
<td>0.59</td>
<td>7.54</td>
<td>0.07</td>
<td>0.04</td>
</tr>
<tr>
<td>WRF-YSUeresa</td>
<td>0.49</td>
<td>0.50</td>
<td>5.66</td>
<td>0.04</td>
<td>0.03</td>
</tr>
<tr>
<td>WRF-TEMF_GFS</td>
<td>1.09</td>
<td>1.25</td>
<td>10.08</td>
<td>0.09</td>
<td>0.06</td>
</tr>
<tr>
<td>WRF-TEMF Era</td>
<td>0.97</td>
<td>1.10</td>
<td>10.01</td>
<td>0.07</td>
<td>0.03</td>
</tr>
<tr>
<td>WRF-ONSE_GFS</td>
<td>0.90</td>
<td>1.00</td>
<td>5.17</td>
<td>0.05</td>
<td>0.03</td>
</tr>
<tr>
<td>WRF-ONSE Era</td>
<td>1.03</td>
<td>1.15</td>
<td>6.06</td>
<td>0.04</td>
<td>0.02</td>
</tr>
<tr>
<td>WRF-MYNN_GFS</td>
<td>0.65</td>
<td>0.77</td>
<td>9.69</td>
<td>0.10</td>
<td>0.05</td>
</tr>
<tr>
<td>WRF-MYNN Era</td>
<td>0.67</td>
<td>0.77</td>
<td>9.51</td>
<td>0.11</td>
<td>0.04</td>
</tr>
<tr>
<td>WRF-MJY_GFS</td>
<td>0.79</td>
<td>0.89</td>
<td>8.56</td>
<td>0.08</td>
<td>0.04</td>
</tr>
<tr>
<td>WRF-MJY Era</td>
<td>0.72</td>
<td>0.84</td>
<td>8.43</td>
<td>0.07</td>
<td>0.04</td>
</tr>
<tr>
<td>WRF-Ensemble</td>
<td>0.63</td>
<td>0.76</td>
<td>5.14</td>
<td>0.05</td>
<td>0.02</td>
</tr>
</tbody>
</table>

units m s-1 m s-1 deg - -

# Metrics vs stability conditions
zLbins = [-20, -0.2, 0.2, 20]  # bins
zLbins_label = ['u','n','s']  # stability classes
iz = 1  # 10 m (fluxes at 3m are NaN)
NzL = len(zLbins_label)

zL = zh_obs[iz]/L_obs[iz]
idf = []
for izL in range(0,NzL): idf.append('REWS_'+zLbins_label[izL])
for izL in range(0,NzL): idf.append('Shub_'+zLbins_label[izL])
for izL in range(0,NzL): idf.append('WDhub_'+zLbins_label[izL])
for izL in range(0,NzL): idf.append('alpha_'+zLbins_label[izL])
for izL in range(0,NzL): idf.append('veer_'+zLbins_label[izL])

Metrics3 = pd.DataFrame(np.zeros((Nsim,15)),
        index = simID,
        columns = idf)
Metrics3.loc['units',:] = np.concatenate((np.repeat('m s-1',NzL),
                           np.repeat('m s-1',NzL),np.repeat('deg',NzL),
                           np.repeat('-',NzL),np.repeat('-',NzL)),axis = 0)

for isim in range(0,Nsim):
    for izL in range(0,NzL):
        i0 = np.where(np.logical_and(zL >= zLbins[izL], zL < zLbins[izL+1]))[0]
Metrics3.loc[(simID[isim]),'REWS_' + zLbins_label[izL]] = mae(REWS[refsim][i0],
                        REWS[isim][i0], 'False')
Metrics3.loc[(simID[isim]),'Shub_' + zLbins_label[izL]] = mae(Shub[refsim][i0],
                        Shub[isim][i0], 'False')
Metrics3.loc[(simID[isim]),'WDhub_' + zLbins_label[izL]] = mae(WDhub[refsim][i0],
                        WDhub[isim][i0], 'False')
Metrics3.loc[(simID[isim]),'alpha_' + zLbins_label[izL]] = mae(alpha[refsim][i0],
                        alpha[isim][i0], 'False')
Metrics3.loc[(simID[isim]),'veer_' + zLbins_label[izL]] = mae(veer[refsim][i0],
                        veer[isim][i0], 'False')

tablecaption = ("**Table 3: $MAE_{ref}$ for unstable ($u$), neutral ($n$) and stable ($s$) conditions.**")
display(Markdown(tablecaption))
display(Metrics3)

Table 3: $MAE_{ref}$ for unstable ($u$), neutral ($n$) and stable ($s$) conditions.

<table>
<thead>
<tr>
<th></th>
<th>REWS_u</th>
<th>REWS_n</th>
<th>REWS_s</th>
<th>Shub_u</th>
<th>Shub_n</th>
<th>Shub_s</th>
<th>WDhub_u</th>
<th>WDhub_n</th>
</tr>
</thead>
<tbody>
<tr>
<td>WRF-YSU (ref)</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>WRF-YSU_LES</td>
<td>0.48</td>
<td>0.44</td>
<td>0.42</td>
<td>0.58</td>
<td>0.72</td>
<td>0.48</td>
<td>4.59</td>
<td>4.15</td>
</tr>
<tr>
<td>VentosM</td>
<td>0.64</td>
<td>0.75</td>
<td>0.74</td>
<td>0.68</td>
<td>0.80</td>
<td>0.73</td>
<td>9.74</td>
<td>7.02</td>
</tr>
<tr>
<td>CFDwind1D_ke</td>
<td>0.74</td>
<td>0.45</td>
<td>0.56</td>
<td>0.73</td>
<td>0.53</td>
<td>0.65</td>
<td>5.62</td>
<td>7.87</td>
</tr>
<tr>
<td>Alya-Wind1D_ke</td>
<td>0.79</td>
<td>0.39</td>
<td>0.72</td>
<td>0.78</td>
<td>0.51</td>
<td>0.66</td>
<td>4.66</td>
<td>3.16</td>
</tr>
<tr>
<td>Ellipsys1D_ke</td>
<td>0.58</td>
<td>0.40</td>
<td>0.55</td>
<td>0.57</td>
<td>0.40</td>
<td>0.69</td>
<td>4.30</td>
<td>7.47</td>
</tr>
<tr>
<td>Ellipsys3D_ke</td>
<td>0.67</td>
<td>0.47</td>
<td>0.68</td>
<td>0.67</td>
<td>0.60</td>
<td>0.83</td>
<td>4.13</td>
<td>8.03</td>
</tr>
<tr>
<td>Ellipsys3D_LES</td>
<td>0.56</td>
<td>1.27</td>
<td>1.64</td>
<td>0.59</td>
<td>1.06</td>
<td>1.93</td>
<td>11.69</td>
<td>8.36</td>
</tr>
<tr>
<td>SP-Wind_LES</td>
<td>0.48</td>
<td>1.36</td>
<td>0.66</td>
<td>0.50</td>
<td>1.38</td>
<td>0.76</td>
<td>3.00</td>
<td>1.72</td>
</tr>
<tr>
<td>WRF-YSU_GFS</td>
<td>0.25</td>
<td>0.32</td>
<td>0.93</td>
<td>0.30</td>
<td>0.36</td>
<td>0.90</td>
<td>11.02</td>
<td>7.93</td>
</tr>
<tr>
<td>WRF-YSUERA</td>
<td>0.25</td>
<td>0.70</td>
<td>0.68</td>
<td>0.26</td>
<td>0.68</td>
<td>0.69</td>
<td>5.15</td>
<td>5.16</td>
</tr>
<tr>
<td>WRF-TEMP_GFS</td>
<td>0.60</td>
<td>0.96</td>
<td>1.56</td>
<td>0.64</td>
<td>1.12</td>
<td>1.84</td>
<td>10.71</td>
<td>13.78</td>
</tr>
<tr>
<td>WRF-TEMPERA</td>
<td>0.80</td>
<td>0.97</td>
<td>1.12</td>
<td>0.87</td>
<td>1.08</td>
<td>1.32</td>
<td>11.29</td>
<td>16.15</td>
</tr>
<tr>
<td>WRF-QNSE_GFS</td>
<td>0.83</td>
<td>0.60</td>
<td>1.00</td>
<td>0.89</td>
<td>0.71</td>
<td>1.14</td>
<td>6.03</td>
<td>5.99</td>
</tr>
<tr>
<td>WRF-QNSEERA</td>
<td>0.97</td>
<td>1.32</td>
<td>1.04</td>
<td>1.04</td>
<td>1.51</td>
<td>1.20</td>
<td>8.49</td>
<td>7.03</td>
</tr>
<tr>
<td>WRF-MYNN_GFS</td>
<td>0.41</td>
<td>0.24</td>
<td>0.94</td>
<td>0.47</td>
<td>0.35</td>
<td>1.12</td>
<td>11.89</td>
<td>10.45</td>
</tr>
<tr>
<td>WRF-MYNNERA</td>
<td>0.46</td>
<td>0.54</td>
<td>0.88</td>
<td>0.53</td>
<td>0.62</td>
<td>1.01</td>
<td>11.36</td>
<td>9.90</td>
</tr>
<tr>
<td>WRF-MJY_GFS</td>
<td>0.61</td>
<td>0.53</td>
<td>0.99</td>
<td>0.69</td>
<td>0.66</td>
<td>1.11</td>
<td>13.46</td>
<td>8.29</td>
</tr>
<tr>
<td>WRF-MJYERA</td>
<td>0.65</td>
<td>0.69</td>
<td>0.78</td>
<td>0.70</td>
<td>0.82</td>
<td>0.97</td>
<td>10.45</td>
<td>9.53</td>
</tr>
<tr>
<td>WRF-Ensemble</td>
<td>0.45</td>
<td>0.51</td>
<td>0.82</td>
<td>0.51</td>
<td>0.60</td>
<td>1.02</td>
<td>5.79</td>
<td>8.98</td>
</tr>
<tr>
<td>units</td>
<td>m s-1</td>
<td>m s-1</td>
<td>m s-1</td>
<td>m s-1</td>
<td>m s-1</td>
<td>m s-1</td>
<td>m s-1</td>
<td>deg</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>WDhub_s</th>
<th>alpha_u</th>
<th>alpha_n</th>
<th>alpha_s</th>
<th>veer_u</th>
<th>veer_n</th>
<th>veer_s</th>
</tr>
</thead>
<tbody>
<tr>
<td>WRF-YSU (ref)</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>WRF-YSU_LES</td>
<td>3.62</td>
<td>0.03</td>
<td>0.05</td>
<td>0.08</td>
<td>0.02</td>
<td>0.02</td>
<td>0.03</td>
</tr>
<tr>
<td>VentosM</td>
<td>3.10</td>
<td>0.04</td>
<td>0.08</td>
<td>0.05</td>
<td>0.04</td>
<td>0.02</td>
<td>0.03</td>
</tr>
<tr>
<td>CFDwind1D_ke</td>
<td>6.53</td>
<td>0.07</td>
<td>0.10</td>
<td>0.11</td>
<td>0.05</td>
<td>0.09</td>
<td>0.05</td>
</tr>
<tr>
<td>Alya-Wind1D_ke</td>
<td>1.24</td>
<td>0.09</td>
<td>0.03</td>
<td>0.06</td>
<td>0.02</td>
<td>0.02</td>
<td>0.03</td>
</tr>
<tr>
<td>Ellipsys1D_ke</td>
<td>5.75</td>
<td>0.06</td>
<td>0.12</td>
<td>0.13</td>
<td>0.03</td>
<td>0.10</td>
<td>0.05</td>
</tr>
<tr>
<td>Ellipsys3D_ke</td>
<td>5.70</td>
<td>0.04</td>
<td>0.14</td>
<td>0.13</td>
<td>0.02</td>
<td>0.10</td>
<td>0.04</td>
</tr>
<tr>
<td>Ellipsys3D_LES</td>
<td>12.81</td>
<td>0.07</td>
<td>0.10</td>
<td>0.11</td>
<td>0.02</td>
<td>0.05</td>
<td>0.11</td>
</tr>
<tr>
<td>SP-Wind_LES</td>
<td>3.71</td>
<td>0.02</td>
<td>0.02</td>
<td>0.06</td>
<td>0.01</td>
<td>0.02</td>
<td>0.03</td>
</tr>
<tr>
<td>WRF-YSU_GFS</td>
<td>4.26</td>
<td>0.04</td>
<td>0.08</td>
<td>0.09</td>
<td>0.02</td>
<td>0.05</td>
<td>0.06</td>
</tr>
<tr>
<td>WRF-YSUERA</td>
<td>6.22</td>
<td>0.03</td>
<td>0.03</td>
<td>0.06</td>
<td>0.01</td>
<td>0.04</td>
<td>0.04</td>
</tr>
<tr>
<td>WRF-TEMP_GFS</td>
<td>8.95</td>
<td>0.07</td>
<td>0.13</td>
<td>0.10</td>
<td>0.07</td>
<td>0.10</td>
<td>0.04</td>
</tr>
<tr>
<td>WRF-TEMPERA</td>
<td>7.90</td>
<td>0.03</td>
<td>0.11</td>
<td>0.09</td>
<td>0.01</td>
<td>0.01</td>
<td>0.04</td>
</tr>
<tr>
<td>WRF-QNSE_GFS</td>
<td>4.26</td>
<td>0.02</td>
<td>0.06</td>
<td>0.07</td>
<td>0.02</td>
<td>0.04</td>
<td>0.03</td>
</tr>
<tr>
<td>WRF-QNSEERA</td>
<td>3.68</td>
<td>0.01</td>
<td>0.11</td>
<td>0.06</td>
<td>0.01</td>
<td>0.02</td>
<td>0.02</td>
</tr>
<tr>
<td>WRF-MYNN_GFS</td>
<td>7.55</td>
<td>0.06</td>
<td>0.18</td>
<td>0.13</td>
<td>0.03</td>
<td>0.03</td>
<td>0.06</td>
</tr>
</tbody>
</table>

(continues on next page)
### Discussion

[As per Sanz Rodrigo et al. (2017)]

All of the microscale models produce similar patterns of the diurnal cycle, demonstrating the effectiveness of the offline coupling methodology. Fig. 5 shows contour plots of the evolution of the vertical profile of mean flow quantities computed by the participating models compared to the observations on the top row. The second row corresponds to the reference mesoscale model that is used to derive input forcings and boundary conditions to drive the microscale simulations. Since no additional information is added at microscale, we can assume that this mesoscale simulation is already a good reference for microscale models to verify a correct implementation of the meso-micro methodology. The differences in the mean flow arise from the different ways each model represents turbulence, which is noticed in the $TKE$ contour plots.

The ensemble mean of the WRF simulations used in the sensitivity analysis is plotted in the third row. By ensemble averaging, we obtain a better match with the observations than by using any single WRF simulation – this result was also seen for the LLJ cases discussed in [34]. The ensemble here provides a better prediction of the LLJ with a distinct velocity maximum around midnight instead of a broader double-peak as in the reference WRF simulation.

The microscale models diverge in their estimates of rotor-based quantities of interest. Time series of rotor-based quantities of interest are shown in Fig. 3. The spread of the models for REWS is around $2\,m/s$ and around $15^\circ$ in terms of $WD_{hub}$. The spread in terms of wind shear and wind veer is also large specially during nighttime stable conditions. Vertical profiles of wind speed and direction at midnight (Fig. 7) show how the models capture the characteristics of the LLJ. The phase error in the input data dominates the bias in the simulations; this input error cannot be corrected at the microscale by simply changing the turbulence model.

Table 2 summarize the differences between the simulations and observations in terms of $MAE$ with respect to observations (Table 2a, $MAE_{obs}$) and with respect to the reference WRF simulation (Table 2b, $MAE_{ref}$). These are differences integrated over the whole diurnal cycle; therefore mixing all kinds of surface stability and large-scale conditions into a single quantity. We shall focus on the $MAE_{ref}$ to quantify the impact of choosing a different turbulence model at microscale.

WRF-YSU_LES is the closest to the reference, which is to be expected since they are results from different nests of the same simulation. Still, differences are significant of around $0.5\,m/s$ of wind speed and $4^\circ$ of wind direction at hub-height. Microscale models increase the error by $0.2-0.7\,m/s$ of wind speed and up to $7^\circ$ of wind direction at hub-height with respect to the WRF-YSU_LES results. With respect to observations, all simulations show similar results with a $MAE$ of $1.1-1.6\,m/s$ of wind speed and up to $14^\circ$ of wind direction at hub-height. Considering vertical wind speed shear and wind direction veer, SP-Wind and the WRF ensemble produce the closest results to the reference WRF simulation.

Different sets of $k-\epsilon$ constants have been tested (not shown) leading to changes that are within the spread shown in Figure 6, which is also comparable to that observed in the WRF sensitivity analysis when changing the planetary boundary-layer scheme.

Recently, capabilities of EllipSys3D have been extended to cover wall modeled LES of stratified flows and at the same time, a “striped” down 1D version of the code have been made operational [28]. Figures 5 and 6 and Tables 2 and 3 show a general good agreement between URANS based EllipSys1D and EllipSys3D computations. Some minor differences exist thought; a possible cause of them might be related to the fact that the vertical velocity ($W$) and the advection terms are implicitly assumed to be zero in the EllipSys1D. Further investigations are necessary to confirm this.
Initial LES computations based on coarse grid resolution and very basic Smagorinsky SGS model show that EllipSys3D LES is capable of reproducing the basic LLJ features observed at the Cabauw site, but Tables 2 and 3 MAE comparisons and Figures 2 and 3 indicate that significantly higher resolution and a more advanced approach to SGS turbulence modeling are needed in order to capture all main details relevant for its application in a wind energy context.

Regarding VENTOS®/M wind speed results, both Figures 5 and 7 show that the LLJ magnitude was reasonably well predicted. The diurnal cycle of the simulation wind speed shows over-predictions around 1.4 m s⁻¹, affecting the REWS and Shub error values in Table 2 which are higher than the reference simulation. Good agreement was obtained for the wind direction, shear and veer regarding their integrated error. Analysis of Figure 6 indicates a generalized under-prediction of $\alpha$ for several nocturnal periods, analogous to a higher turbulent shape factor of the boundary-layer. These mismatches happen also with WRF and, despite the VENTOS®/M limitations of its heat-flux boundary condition, the microscale simulation predicts higher values of $\alpha$ and closer to the observations. The results further show a ~2.5 K temperature bias that occurs in both WRF and VENTOS®/M results, as well as in the other microscale simulations, which originates from the ERA-Interim input data.

Finally, Table 3 shows the $MAE_{ref}$ is computed for different stability classes filtering with the observed stability parameter $z/L$, where $L$ is the Obukhov length and $z = 10$ m: unstable ($'u'$: $z/L < -0.2$), neutral ($'n'$: $-0.2 < z/L < 0.2$) and stable ($'s'$: $z/L > 0.2$). Not all the models behave similarly depending on stability. The WRF ensemble REWS is more sensitive in stable conditions. This is also the case for Ellipsys3D_LES probably due to the coarser resolution of the simulation compared to the other LES simulations that do not show this high sensitivity in stable conditions.

In general, it is difficult to extract more meaningful conclusions from Table 2 and Table 3 due to the limited statistical significance of the samples. The overall assessment would be richer if several diurnal cycles from uncorrelated synoptic conditions would have been tested. The ensemble WRF simulations for the same cycle already show significant improvement on mean flow quantities.

Conclusions

[As per Sanz Rodrigo et al. (2017)]

Results of the GABLS3 diurnal cycle benchmark with an emphasis on rotor-relevant values are presented. The main challenge for microscale models was to produce consistent flow fields with respect to the mesoscale model that was used to derive their input forcings. This consistency has been achieved by both LES and URANS models. The spread of the models is significant but of similar magnitude as that shown by WRF using different boundary-layer parameterizations. The input uncertainty coming from the mesoscale, even in relatively ideal conditions, is large and results in $MAE$ of wind speed at hub-height of the order of 1.1-1.6 m s⁻¹ over the whole cycle and and hourly errors of up to 3 m s⁻¹. This is partly mitigated when using an ensemble average of several simulations which also lead to better results in terms of wind shear and wind veer. By ensuring consistency of the microscale models at introducing input forcings we can proceed with further analysis on how RANS and LES models interpret the structure of turbulence in different stability conditions.

Acknowledgements

This article was produced with funding from the “MesoWake” Marie Curie International Outgoing Fellowship (FP7-PEOPLE-2013-IOF, European Commission’s grant agreement number 624562). It also counts with contributions supported by the New European Wind Atlas (NEWA, FP7-ENERGY.2013.10.1.2 European Commission’s grant agreement number 618122). WRF-LES simulations have been produced using high-performance computing resources from the PRACE-MesoWake project in the MareNostrum system, based in Barcelona, Spain. The SP-Wind simulations were performed with computational resources and services provided by the VSC (Flemish Supercomputer Center), funded by the Research Foundation – Flanders (FWO) and the Flemish Government – department EWI. JMT’s mesoscale ensemble simulations were conducted using the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation grant number ACI-1053575. JKL’s efforts were supported by the National Renewable Energy Laboratory under APUP UGA-0-41026-22. NREL is a national laboratory.

3.1. Wind Conditions

References


3.1. Wind Conditions 33
Assessment of meso-micro offline coupling methodology based on driving CFDWind single-column-model with WRF tendencies: the GABLS3 diurnal cycle case

Javier Sanz Rodrigo, National Renewable Energy Laboratory (CENER), Sarriguren, Spain, jsrodrigo@cener.com

April 2017

Introduction

This notebook provides the model evaluation process developed for the assessment of a methodology for testing atmospheric boundary layer models under realistic mesoscale forcing. Tendencies from WRF are used as inputs for CFDWind single-column-model (SCM) to reproduce the GABLS3 diurnal cycle at the Cabauw site. A sensitivity test is conducted to test different SCM settings as well as profile nudging to dynamically correct for model bias.

This work has been used to set up a benchmark for other wind energy models dealing with meso-micro coupling. The benchmark was launched at the Torque 2016 conference [1] and the results of this study with CFDWind1D have been published in [2].

Benchmark Set-Up

Background information and benchmark set-up can be found in: http://windbench.net/gabls-3

CFDWind1D Simulations

The following simulations were conducted in the sensitivity analysis to target the following model evaluation objectives:

1. Demonstrate consistency of online (WRF) vs asynchronous meso-micro coupling
2. Evaluate the choice of turbulent closure with realistic forcing
3. Quantify the impact of the choice of surface boundary conditions on fluxes and quantities of interest
4. Quantify the relative importance of mesoscale tendencies on quantities of interest
5. Assess bias-correction nudging method using typical wind energy mast configurations
6. Assess bias-correction nudging method using typical wind energy lidar configurations
<table>
<thead>
<tr>
<th>SimID</th>
<th>Turbulence</th>
<th>Surface B.C.</th>
<th>Forcing</th>
<th>Objectives</th>
</tr>
</thead>
<tbody>
<tr>
<td>WRF-YSU</td>
<td>YSU</td>
<td>Noah</td>
<td>ERA Interim</td>
<td>1</td>
</tr>
<tr>
<td>ke_T2 (ref)</td>
<td>k − ε</td>
<td>WRF T$_2$</td>
<td>WRF tendencies</td>
<td>1</td>
</tr>
<tr>
<td>SI_T2</td>
<td>S − l</td>
<td>WRF T$_2$</td>
<td>WRF tendencies</td>
<td>2</td>
</tr>
<tr>
<td>kl_T2</td>
<td>k − l</td>
<td>WRF T$_2$</td>
<td>WRF tendencies</td>
<td>2</td>
</tr>
<tr>
<td>ke_T2wt</td>
<td>k − ε</td>
<td>WRF T$_2$ and $w\theta_0$</td>
<td>WRF tendencies</td>
<td>3</td>
</tr>
<tr>
<td>ke_Tsk</td>
<td>k − ε</td>
<td>WRF Θ$_0$</td>
<td>WRF tendencies</td>
<td>3</td>
</tr>
<tr>
<td>ke_T2obs</td>
<td>k − ε</td>
<td>Observed T$_2$</td>
<td>WRF tendencies</td>
<td>3</td>
</tr>
<tr>
<td>noTadv</td>
<td>k − ε</td>
<td>WRF T$_2$</td>
<td>without Θ$_{adv}$ tendency</td>
<td>4</td>
</tr>
<tr>
<td>noTadvUadv</td>
<td>k − ε</td>
<td>WRF T$_2$</td>
<td>without advection tendencies</td>
<td>4</td>
</tr>
<tr>
<td>noTadvUadv_Sg0</td>
<td>k − ε</td>
<td>WRF T$_2$</td>
<td>only surface pressure gradient</td>
<td>4</td>
</tr>
<tr>
<td>UVnud80</td>
<td>k − ε</td>
<td>Observed T$_2$</td>
<td>$U, V$: 10-80 m; Θ: 2-80 m; $\tau_{nud} = 60$ min</td>
<td>5</td>
</tr>
<tr>
<td>UVnud120</td>
<td>k − ε</td>
<td>Observed T$_2$</td>
<td>$U, V$: 10-120 m; Θ: 2-120 m; $\tau_{nud} = 60$ min</td>
<td>5</td>
</tr>
<tr>
<td>UVnud200</td>
<td>k − ε</td>
<td>Observed T$_2$</td>
<td>$U, V$: 10-200 m; Θ: 2-200 m; $\tau_{nud} = 60$ min</td>
<td>5</td>
</tr>
<tr>
<td>UVnud200_tau10</td>
<td>k − ε</td>
<td>Observed T$_2$</td>
<td>$U, V$: 10-200 m; Θ: 2-200 m; $\tau_{nud} = 10$ min</td>
<td>5</td>
</tr>
<tr>
<td>UVnud400</td>
<td>k − ε</td>
<td>WRF T$_2$</td>
<td>$U, V$: 40-400 m; $\tau_{nud} = 60$ min</td>
<td>6</td>
</tr>
<tr>
<td>UVnud200</td>
<td>k − ε</td>
<td>WRF T$_2$</td>
<td>$U, V$: 40-200 m; $\tau_{nud} = 60$ min</td>
<td>6</td>
</tr>
<tr>
<td>UVnud200_tau30</td>
<td>k − ε</td>
<td>WRF T$_2$</td>
<td>$U, V$: 40-200 m; $\tau_{nud} = 30$ min</td>
<td>6</td>
</tr>
<tr>
<td>UVnud200_tau10</td>
<td>k − ε</td>
<td>WRF T$_2$</td>
<td>$U, V$: 40-200 m; $\tau_{nud} = 10$ min</td>
<td>6</td>
</tr>
</tbody>
</table>

Remarks: - All surface boundary conditions use Monin-Obukhov similarity theory (MOST) - All use the same WRF tendencies, adding nudging or modified tendencies as indicated - Details about the CFDWind1D model settings and nudging methodology can be found in [2] - All the simulations are run on a 4-km long log-linear grid with 301 levels and a time step of 1 s - The $k − \epsilon$ model of Sogachev et al. (2012) [3] is used as a reference, with the following set of constants: $\kappa = 0.4$, $C_{\epsilon 1} = 1.52$, $C_{\epsilon 2} = 1.833$, $\sigma_k = 2.95$, $\sigma_\epsilon = 2.95$ and $C_{\mu} = 0.03$

### 3.1. Wind Conditions

```python
# Load libraries and define input data

"[3]: %matplotlib inline"

```
PO = 100000. # Reference pressure [Pa]

# Site ID
siteID = 'GABLS3'
lats = 51.971 # degrees N
lons = 4.927 # degrees E

# Evaluation Period and reference Rotor characteristics
datefrom = datetime.datetime(2006, 7, 1, 12, 0, 0) # Origin of evaluation period
dateto = datetime.datetime(2006, 7, 2, 12, 0, 0) # End of evaluation period
Hhub = 120. # hub-height
Drot = 160. # rotor diameter
ts = 10 # sampling frequency to evaluate [min]

# Mesoscale tendencies averaging settings
tav = 60.0 # Time averaging time used in simulations [min]
Lav = 9000.0 # Spatial averaging [m]

Load Simulation data

[20]: dirsim = '.
filesim = [dirsim + '/GABLS3_tendencies_d02_YSU_w60_L9000.nc',
dirsim + '/GABLS3_CFDWindSCM_turb5_T2_9.nc',
dirsim + '/GABLS3_CFDWindSCM_turb1_T2_10.nc',
dirsim + '/GABLS3_CFDWindSCM_turb5_T2wt_6.nc',
dirsim + '/GABLS3_CFDWindSCM_turb5_Tsk_7.nc',
dirsim + '/GABLS3_CFDWindSCM_turb5_T2obs_11.nc',
dirsim + '/GABLS3_CFDWindSCM_turb5_T2obs_noTadv_3.nc',
dirsim + '/GABLS3_CFDWindSCM_turb5_T2obs_noTadvUadv_4.nc',
dirsim + '/GABLS3_CFDWindSCM_turb5_T2obs_noTadvUadvSg0_5.nc',
dirsim + '/GABLS3_CFDWindSCM_turb5_T2obs_UVTnud2-80_12.nc',
dirsim + '/GABLS3_CFDWindSCM_turb5_T2obs_UVTnud2-120_13.nc',
dirsim + '/GABLS3_CFDWindSCM_turb5_T2obs_UVTnud2-200_20.nc',
dirsim + '/GABLS3_CFDWindSCM_turb5_T2obs_UVTnud2-200tau10_21.nc',
dirsim + '/GABLS3_CFDWindSCM_turb5_T2_UVnud40-400_14.nc',
dirsim + '/GABLS3_CFDWindSCM_turb5_T2_UVnud40-200_15.nc',
dirsim + '/GABLS3_CFDWindSCM_turb5_T2_UVnud40-200tau30_16.nc',
dirsim + '/GABLS3_CFDWindSCM_turb5_T2_UVnud40-200tau10_17.nc',
]

simID = ['WRF-YSU','ke_T2',
'k1_T2','ke_T2wt','ke_Tsk','ke_T2obs',
'noTadv','noTadvUadv','noTadvUadv_Sg0',
'UVnud80','UVnud120','UVnud200','UVnud200_tau10',
'UVnud400','UVnud200','UVnud200_tau30','UVnud200_tau10']

3.1. Wind Conditions
Load Observations

```
[21]: # Note that the file 'gabls3_scm_cabauw_obs_v33.nc' can be obtained from the KNMI GABLS3 website
# http://projects.knmi.nl/gabls/gabls3_scm_cabauw_obs_v33.nc
# Alternatively, it is also provided in the Windbench/GABLS3 input dataset
dirobs = '.'
fileobs = dirobs + '/gabls3_scm_cabauw_obs_v33.nc'
nodata = -9999.0 # missing data flag
date0 = datetime.datetime(2006,7,1,0,0,0) # origin of time_obs
f1 = netCDF4.Dataset(fileobs, 'r')
dates_obs = f1.variables['date'][:]
time_obs = f1.variables['time'][:] # 'hours since 2006-07-01 00:00:00 0:00'
date_obs = []
for i in range (0,len(time_obs)):
    date_obs.append(date0 + datetime.timedelta(seconds = np.int(time_obs[i]*3600.0)))
ifrom_obs=0
for j in range(0,len(date_obs)):
    if date_obs[j] <= datefrom:
        ifrom_obs = j
ito_obs=0
for j in range(0,len(date_obs)):
    if date_obs[j] <= dateto:
        ito_obs = j+1
date_obs = mdates.date2num(date_obs[ifrom_obs:ito_obs])
time_obs = 24.0*(date_obs - date_obs[0])
# Surface variables
ustar_obs = pd.DataFrame(np.ma.array(f1.variables['ustar'][ifrom_obs:ito_obs], mask=(f1.variables['ustar'][ifrom_obs:ito_obs] == nodata)), index = mdates.num2date(date_obs))
U10_obs = pd.DataFrame(np.ma.array(f1.variables['u10m'][ifrom_obs:ito_obs], mask=(f1.variables['u10m'][ifrom_obs:ito_obs] == nodata)), index = mdates.num2date(date_obs))
V10_obs = pd.DataFrame(np.ma.array(f1.variables['v10m'][ifrom_obs:ito_obs], mask=(f1.variables['v10m'][ifrom_obs:ito_obs] == nodata)), index = mdates.num2date(date_obs))
S10_obs = pd.DataFrame(np.ma.array(f1.variables['f10m'][ifrom_obs:ito_obs], mask=(f1.variables['f10m'][ifrom_obs:ito_obs] == nodata)), index = mdates.num2date(date_obs))
WD10_obs = pd.DataFrame(np.ma.array(f1.variables['d10m'][ifrom_obs:ito_obs], mask=(f1.variables['d10m'][ifrom_obs:ito_obs] == nodata)), index = mdates.num2date(date_obs))
T2_obs = pd.DataFrame(np.ma.array(f1.variables['t2m'][ifrom_obs:ito_obs], mask=(f1.variables['t2m'][ifrom_obs:ito_obs] == nodata)), index = mdates.num2date(date_obs))
HFX_obs = pd.DataFrame(np.ma.array(f1.variables['shf'][ifrom_obs:ito_obs], mask=(f1.variables['shf'][ifrom_obs:ito_obs] == nodata)), index = mdates.num2date(date_obs))
Ts_obs = pd.DataFrame(np.ma.array(f1.variables['ts'][ifrom_obs:ito_obs], mask=(f1.variables['ts'][ifrom_obs:ito_obs] == nodata)), index = mdates.num2date(date_obs))
```

(continues on next page)
zs_obs = f1.variables['zs'][ifrom_obs:ito_obs][0,:]
# soil temperature heights [m]
Tsk_obs = pd.DataFrame(np.ma.array(f1.variables['tsk'][ifrom_obs:ito_obs], mask=(f1.
variable['tsk'][ifrom_obs:ito_obs] == nodata)), index = mdates.num2date(date_obs))
Tsk1_obs = pd.DataFrame(np.ma.array(f1.variables['tsk1'][ifrom_obs:ito_obs], mask=(f1.
variable['tsk1'][ifrom_obs:ito_obs] == nodata)), index = mdates.num2date(date_obs))

# Vertical profiles
zf_obs = f1.variables['zf'][ifrom_obs:ito_obs][0,:]
# velocity profile heights [m]
zt_obs = f1.variables['zt'][ifrom_obs:ito_obs][0,:]
# temperature profile heights
U_obs = pd.DataFrame(np.ma.array(f1.variables['u'][ifrom_obs:ito_obs], mask=(f1.
variable['u'][ifrom_obs:ito_obs] == nodata)), index = mdates.num2date(date_obs))
V_obs = pd.DataFrame(np.ma.array(f1.variables['v'][ifrom_obs:ito_obs], mask=(f1.
variable['v'][ifrom_obs:ito_obs] == nodata)), index = mdates.num2date(date_obs))
S_obs = pd.DataFrame(np.ma.array(f1.variables['f'][ifrom_obs:ito_obs], mask=(f1.
variable['f'][ifrom_obs:ito_obs] == nodata)), index = mdates.num2date(date_obs))

# horizontal wind speed [deg]
WD_obs = pd.DataFrame(np.ma.array(f1.variables['d'][ifrom_obs:ito_obs], mask=(f1.
variable['d'][ifrom_obs:ito_obs] == nodata)), index = mdates.num2date(date_obs))
# wind direction [deg]
Th_obs = pd.DataFrame(np.ma.array(f1.variables['th'][ifrom_obs:ito_obs], mask=(f1.
variable['th'][ifrom_obs:ito_obs] == nodata)), index = mdates.num2date(date_obs))
# potential temperature [K]
q_obs = pd.DataFrame(np.ma.array(f1.variables['q'][ifrom_obs:ito_obs], mask=(f1.
variable['q'][ifrom_obs:ito_obs] == nodata)), index = mdates.num2date(date_obs))

# fluxes
zh_obs = f1.variables['zh'][ifrom_obs:ito_obs][0,:]
# fluxes heights
wt_obs = pd.DataFrame(np.ma.array(f1.variables['wt'][ifrom_obs:ito_obs], mask=(f1.
variable['wt'][ifrom_obs:ito_obs] == nodata)), index = mdates.num2date(date_obs))
uw_obs = pd.DataFrame(np.ma.array(f1.variables['uw'][ifrom_obs:ito_obs], mask=(f1.
variable['uw'][ifrom_obs:ito_obs] == nodata)), index = mdates.num2date(date_obs))
vw_obs = pd.DataFrame(np.ma.array(f1.variables['vw'][ifrom_obs:ito_obs], mask=(f1.
variable['vw'][ifrom_obs:ito_obs] == nodata)), index = mdates.num2date(date_obs))
us_obs = (uw_obs**2 + vw_obs**2)**0.25

# Obukhov length
T0_obs = np.tile(Th_obs[36].values,(5,1)).T
L_obs = -us_obs**3/(K*(g/T0_obs)*wt_obs)

# Resample and apply tav to observations
ws = int(tav/ts) # rolling window size
ustar_obs_w = ustar_obs.resample(str(ts)+'Min').mean().bfill().rolling(window = ws).
mean()
HFX_obs_w = HFX_obs.resample(str(ts)+'Min').mean().bfill().rolling(window = ws).mean()
U10_obs_w = U10_obs.resample(str(ts)+'Min').mean().bfill().rolling(window = ws).mean()
V10_obs_w = V10_obs.resample(str(ts)+'Min').mean().bfill().rolling(window = ws).mean()
S10_obs_w = S10_obs.resample(str(ts)+'Min').mean().bfill().rolling(window = ws).mean()
WD10_obs_w = WD10_obs.resample(str(ts)+'Min').mean().bfill().rolling(window = ws).
mean()
T2_obs_w = T2_obs.resample(str(ts)+'Min').mean().bfill().rolling(window = ws).mean()
Ts_obs_w = Ts_obs.resample(str(ts)+'Min').mean().bfill().rolling(window = ws).mean()
U_obs_w = U_obs.resample(str(ts)+'Min').mean().bfill().rolling(window = ws).mean()
V_obs_w = V_obs.resample(str(ts)+'Min').mean().bfill().rolling(window = ws).mean()
S_obs_w = S_obs.resample(str(ts)+'Min').mean().bfill().rolling(window = ws).mean()
WD_obs_w = WD_obs.resample(str(ts)+'Min').mean().bfill().rolling(window = ws).mean()
Th_obs_w = Th_obs.resample(str(ts)+'Min').mean().bfill().rolling(window = ws).mean()
q_obs_w = q_obs.resample(str(ts)+'Min').mean().bfill().rolling(window = ws).mean()
wt_obs_w = wt_obs.resample(str(ts)+'Min').mean().bfill().rolling(window = ws).mean()
uw_obs_w = uw_obs.resample(str(ts)+'Min').mean().bfill().rolling(window = ws).mean()
vw_obs_w = vw_obs.resample(str(ts)+'Min').mean().bfill().rolling(window = ws).mean()
us_obs_w = us_obs.resample(str(ts)+'Min').mean().bfill().rolling(window = ws).mean()
L_obs_w = L_obs.resample(str(ts)+'Min').mean().bfill().rolling(window = ws).mean()
TKE_obs_w = pd.DataFrame(data=None, columns=U_obs_w.columns,index=U_obs_w.index)

if zflux == 3.0:
    ust0_obs_w = np.ravel(ustar_obs_w.values)
    Psfc_obs = np.interp(time_obs,np.ravel(t[0]),np.ravel(Psfc[0].values))
    T0_obs_w = np.ravel(Th_obs_w[36].values)
    rho_obs_w = Psfc_obs/(R_air*T0_obs_w)
    wt0_obs_w = np.ravel(HFX_obs_w.values)/(rho_obs_w*Cp_air)
    L0_obs_w = -ust0_obs_w**3/(K*(g/T0_obs_w)*wt0_obs_w)
else:
    izh = np.where(zh_obs == zflux)[0][0]
    ust0_obs_w = us_obs_w[izh].values
    wt0_obs_w = wt_obs_w[izh].values
    L0_obs_w = L_obs_w[izh].values

Plot tz contours

[19]: # List simulations for index reference
for isim in range(0,len(simID)):
    print isim, simID[isim]
0 WRF-YSU
1 ke_T2
2 Sl_T2
3 kl_T2
4 ke_T2wt
5 ke_Tsk
6 ke_T2obs
7 noTadv
8 noTadvUadv
9 noTadvUadv_Sg0
10 UVTnud80
11 UVTnud120
12 UVTnud200
13 UVTnud200_tau10
14 UVnud400
15 UVnud200
16 UVnud200_tau30
17 UVnud200_tau10

[30]: # Select the simulations you want to plot
plotsim = np.array([0,1,7,8,9,13,17])

# Plot settings
zlim = 2000 # height limit [m]
Zcmap = plt.get_cmap('jet') # colormap
figname = siteID+'_tzfields.png' # output filename
Zlevels = np.array([np.linspace(0,16,17), np.linspace(30,210,13)]) # range for S and WD
\[
\text{np.linspace(285,305,11)}, \quad \# \text{ range for } \Theta
\]
\[
\text{np.linspace(0,5,11)} \quad \# \text{ range for } \text{TKE}
\]

\[
X = []; Y = []; Z = []; \text{taxis} = []; \text{zaxis} = [] ; \text{Ztitle} = []
\]
\[
Z = [S_{\text{obs-w}}.T, \text{WD}_{\text{obs-w}}.T, \Theta_{\text{obs-w}}.T, \text{TKE}_{\text{obs-w}}.T]
\]
\[
[Xf,Yf] = \text{np.meshgrid(date}_{\text{obs}}, zf}_{\text{obs})
\]
\[
[XT,YT] = \text{np.meshgrid(date}_{\text{obs}}, zT}_{\text{obs})
\]
\[
X = [Xf, Xf, XT, Xf]
\]
\[
Y = [Yf, Yf, YT, Yf]
\]
\[
\text{taxis} = [\text{date}_{\text{obs}}, \text{date}_{\text{obs}}, \text{date}_{\text{obs}}, \text{date}_{\text{obs}}]
\]
\[
\text{zaxis} = [zf}_{\text{obs}}, zf}_{\text{obs}}, zT}_{\text{obs}}, zf}_{\text{obs}]
\]
\[
\text{Ztitle} = [\text{Observations}]
\]

for ip in range(0,len(plotsim)):
    isim = plotsim[ip]
    Z.append(S[isim].T)
    taxis.append(t[isim]); zaxis.append(z[isim]); X.append(X0); Y.append(Y0)
    Z.append(WD[isim].T)
    taxis.append(t[isim]); zaxis.append(z[isim]); X.append(X0); Y.append(Y0)
    Z.append(Th[isim].T)
    taxis.append(t[isim]); zaxis.append(z[isim]); X.append(X0); Y.append(Y0)
    Z.append(TKE[isim].T)
    taxis.append(t[isim]); zaxis.append(z[isim]); X.append(X0); Y.append(Y0)
Ztitle.append(simID[isim])

Zlabel = ('$S$ [m s$^{-1}$]', '$WD$ ['+u'$\degree$'], '$\Theta$ [K]', '$TKE$ [m$^{2}$ s$^{-2}$])
taxis_label = 'UTC time in hours since ' + datefrom.strftime('%Y-%m-%d %H:%M') + ', $L_{avg} = %.0f$ km, $t_{avg} = %.0f$ min'
zaxis_label = '$z$ [m]

hoursFmt = mdates.DateFormatter('%H')
Zlevels = np.tile(Zlevels,len(Ztitle))

hoursFmt = mdates.DateFormatter('%H')
tticks = [datefrom + datetime.timedelta(hours = 3*x) for x in range(0, 9)]

fig, ax = plt.subplots(len(Ztitle), 4, sharex='col', sharey='row', figsize=(11,17), dpi=300)
xrotor = np.array([mdates.date2num(datefrom), mdates.date2num(dateto)])
yrotor1 = np.array([Hhub - 0.5*Drot, Hhub - 0.5*Drot])
yrotor2 = np.array([Hhub + 0.5*Drot, Hhub + 0.5*Drot])
cbleft = np.array([0.125,0.33,0.53,0.73])

for iax in range (0,len(Ztitle)*4):
    ix,iy = np.unravel_index(iax,(len(Ztitle),4))
    CF0 = ax[ix,iy].contourf(X[ix,iy],Y[ix,iy],Z[ix,iy], Zlevels[ix,iy], cmap=Zcmap)
    ax[ix,iy].plot(xrotor,yrotor1,'--k')
    ax[ix,iy].plot(xrotor,yrotor2,'--k')
    ax[ix,iy].set_ylim([10, zlim])
    ax[ix,iy].set_yscale('log')
    ax[ix,iy].xaxis.set_major_formatter(hoursFmt)
    ax[ix,iy].set_xticks(tticks)

    if iy == 0:
        ax[ix,iy].set_ylabel(zaxis_label)
    if iy == 3:

(continues on next page)
```python
ax[ix,iy].yaxis.set_label_position("right")
ax[ix,iy].set_ylabel(Ztitle[ix], fontsize = 9, fontweight='bold')
if ix == len(Ztitle)-1:
    fig.subplots_adjust(top=0.87)
    cbar_ax = fig.add_axes([cbleft[iy],0.885, 0.165, 0.012])
    cbar = fig.colorbar(CF0, cax=cbar_ax, orientation='horizontal')
    cbar.ax.set_xlabel(Zlabel[iy], labelpad = -42, x = 0.5)

ax[len(Ztitle)-1,1].set_xlabel(taxis_label,x=1.2);
plt.setp([a.get_xticklabels() for a in ax[0,:]], visible=False)
plt.setp([a.get_yticklabels() for a in ax[:, 1]], visible=False)
#fig.savefig(figname, bbox_inches='tight', dpi = 300)

plt.show()
```

```python
from IPython.display import Markdown, display
figcaption = ("**Fig 1. Time-height contour of horizontal wind speed $S$, direction $\rightarrow W D$, potential temperature $\Theta$ and turbulent kinetic energy $TKE$. Dotted lines denote a rotor diameter of 160 m at 120 m hub-height.**")
display(Markdown(figcaption))
```
3.1. Wind Conditions
Fig 1. Time-height contour of horizontal wind speed $S$, direction $WD$, potential temperature $\Theta$ and turbulent kinetic energy $TKE$. Dotted lines denote a rotor diameter of 160 m at 120 m hub-height.

Vertical Profiles

```python
[31]: # Specify the datetime at which you want to plot the vertical profiles
t0 = datetime.datetime(2006,7,2,0,0,0)

# Plot settings
linespec = ["k-","b.-","r.-","c.-","m-","g-","y-","c-","b.-","r--"]
lwidth = np.array([2,1,1,1,1,1,1,1,2,2])
Nm = 3  # marker every

Z_obs = (S_obs_w.loc[t0].values, WD_obs_w.loc[t0].values,
         Th_obs_w.loc[t0].values)
z_obs = (zf_obs, zf_obs, zT_obs)

Z_sim = []; z_sim = []
for iplot in range(0,len(plotsim)):
    isim = plotsim[iplot]
    Z_sim.append((S[isim].loc[t0].values, WD[isim].loc[t0].values,
                  Th[isim].loc[t0].values))
    z_sim.append((z[isim], z[isim], z[isim]))

figname = siteID+'_'+t0.strftime('%Y-%m-%d_%H')+'_profiles.png'
zlim = 1000
fig,ax = plt.subplots(1, 3, sharey='row', figsize=(8,6))
for iax in range (0,3):
    ax[iax].plot(Z_obs[iax], z_obs[iax], 'ok', color='grey', label = 'obs ')
    for iplot in range(0,len(plotsim)):
        isim = plotsim[iplot]
        if (isim == 0 and iax == 2):
            aa = []
        else:
            ax[iax].plot(Z_sim[iplot][iax][1:], z_sim[iplot][iax][1:],
                         linespec[iplot], linewidth = lwidth[iplot], label = simID[isim],
                         markevery=Nm)
    #ax[iax].set_yscale('log')
    xlim = ax[iax].get_xlim()
    ax[iax].plot(xlim,yrotor1,'--k')
    ax[iax].plot(xlim,yrotor2,'--k')
    ax[iax].set_xlim([1., xlim])
    ax[iax].set_yticks(np.linspace(0,zlim,5))
    ax[iax].grid(which='major',color='k',linestyle=':')
    #ax[0].set_xlim([0, 16])
    ax[1].set_xlim([60, 180])
    ax[2].set_xlim([285, 300])
    ax[0].set_xlabel('$S$ $[m s^{-1}]$')
    ax[1].set_xlabel('$WD$ ['+u'\N{DEGREE SIGN}'+']')
    ax[2].set_xlabel(r'$\Theta$ $[K]$')
    ax[0].set_ylabel('$z$ $[m]$')
    ax[0].set_title('$t_{0}$ = '+ t0.strftime('%Y-%m-%d %H:%M:%S'))
```

(continues on next page)
**Fig 2. Vertical profiles of wind speed, wind direction and potential temperature at UTC **

2006-07-02 00:00:00

### Surface Fluxes

```
figname = siteID+'_surfaceflux.png'
fig, (ax1,ax2,ax3,ax4) = plt.subplots(4,1, sharex=True, figsize=(6,8), dpi=300)
ax1.plot(date_obs[::3], ust0_obs_w[::3], 'ok', color='grey', label = 'obs ')
for iplot in range(0,len(plotsim)):
isim = plotsim[iplot]
ax1.plot(t[isim], us[isim], linespec[iplot],linewidth = lwidth[iplot], label = simID[isim])
ax1.grid(which='major',color='grey',linestyle=':')
ax1.set_ylabel(r'$u_{*0}$ [$m s^{-1}$]')
figcaption = ('**Fig 2. Vertical profiles of wind speed, wind direction and potential temperature at UTC **

*Fig 2. Vertical profiles of wind speed, wind direction and potential temperature at UTC **

2006-07-02 00:00:00

### Surface Fluxes

```
ax1.legend(prop={'size':8}, bbox_to_anchor=(1.35, 1), ncol=1, loc='upper right')

ax2.plot(date_obs[::3], wt0_obs_w[::3], 'ok', color='grey', label = 'obs ')
for iplot in range(0,len(plotsim)):
    isim = plotsim[iplot]
    ax2.plot(t[isim], wt[isim], linespec[iplot], linewidth = lwidth[iplot],
             label = simID[isim])
ax2.set_ylabel(r'$w\theta_0$ [$K m s^{-1}$']
#ax2.yaxis.set_major_formatter(tick.FormatStrFormatter('%2.1e'))
ax2.grid(which='major',color='grey',linestyle=':)

ax3.plot(date_obs[::3], zflux/L0_obs_w[::3], 'ok', color='grey', label = 'obs ')
for iplot in range(0,len(plotsim)):
    isim = plotsim[iplot]
    ax3.plot(t[isim], zflux/L[isim], linespec[iplot], linewidth = lwidth[iplot],
             label = simID[isim])
ax3.set_ylabel(r'$z/L_0$ [-]'
#ax3.set_ylim([-1.5, 1.5])
ax3.grid(which='major',color='grey',linestyle=':')

ax4.plot(date_obs[::3], Th_obs_w[36][::3], 'ok', color='grey', label = 'obs ')
for iplot in range(0,len(plotsim)):
    isim = plotsim[iplot]
    ax4.plot(t[isim], T2[isim], linespec[iplot], linewidth = lwidth[iplot],
             label = simID[isim])
ax4.set_ylabel(r'$T_2$ [$K$']
ax4.set_xlabel(taxis_label)
ax4.xaxis.set_major_formatter(hoursFmt)
ax4.grid(which='major',color='grey',linestyle=':)
ax4.set_xticks(tticks)
ax4.set_xlim([mdates.date2num(datefrom), mdates.date2num(dateto)])

#fig.savefig(figname, bbox_inches='tight', dpi=300)

plt.show()

figcaption = (**Fig 3. Surface-layer characteristics**)
display(Markdown(figcaption))
Fig 3. Surface-layer characteristics

Quantities of Interest and Metrics

The performance of the models is based on quantities of interest relevant for wind energy applications. These quantities are evaluated across a reference rotor span of 160 m, between 40 and 200 m, characteristic of an 8-MW large wind turbine. Besides hub-height wind speed $S_{hub}$ and direction $WD_{hub}$, it is relevant to consider the rotor-equivalent wind speed $REWS$, the turbulence intensity (not evaluated here), the wind speed shear $\alpha$, and the wind direction shear or veer $\psi$.

The $REWS$ is especially suitable to account for wind shear in wind turbine power performance tests [14]. The $REWS$ is the wind speed corresponding to the kinetic energy flux through the swept rotor area, when accounting for the vertical shear:

$$REWS = \left[ \frac{1}{A} \sum_i (A_i S_i^3 \cos \beta_i) \right]^{1/3}$$

where $A$ is the rotor area and $A_i$ are the horizontal segments that separate vertical measurement points of horizontal wind speed $S_i$ across the rotor plane. The $REWS$ is here weighted by the cosine of the angle $\beta_i$ of the wind direction $WD_i$ with respect to the hub-height wind direction to account for the effect of wind veer [15].

3.1. Wind Conditions
Wind shear is defined by fitting a power-law curve across the rotor wind speed points $S_i$:

$$ S_i = S_{hub} \left( \frac{z_i}{z_{hub}} \right)^\alpha $$

Similarly, wind veer is defined as the slope $\psi$ of the linear fit of the wind direction difference:

$$ \beta_i = \psi(z_i - z_{hub}) $$

To evaluate simulations and measurements consistently, these quantities are obtained after resampling, by linear interpolation, velocity and wind direction vertical profiles at 10 points across the rotor area and then computing the REWS and the shear functional fits. While these fitting functions are commonly used in wind energy, their suitability in LLJ conditions is questionable. The regression coefficient from the fitting can be used to determine this suitability.

A rolling average with a window size of one hour is applied to simulation and observational data to remove the impact of high frequency fluctuations in the analysis.

Validation results are quantified in terms of the mean absolute error ($MAE$):

$$ MAE = \frac{1}{N} \sum_{i=1}^{N} |\chi_{pred} - \chi_{obs}| $$

where $\chi$ is any of the above mentioned quantities of interest, predicted (pred) or observed (obs), and $N$ is the number of samples evaluated in the time series.

[33]: # Define rotor-based quantities of interest
zrot = np.linspace(Hhub - 0.5*Drot, Hhub + 0.5*Drot, 1 + Drot/10, endpoint=True)
def rotor(z, Sz, WDz):
    # Rotor QoIs [m s-1]
    # z: heights [m] where the velocity and wind direction are known spanning the rotor diameter
    # Sz, WDz: Wind speed [m s-1] and direction [deg from N] at z levels [tdim,zdim]
    # Returns:
    # REWS: rotor equivalent wind speed [m s-1]
    # alpha: wind shear, power-law exponent from linear fit lg(U/Uhub) =
    # alpha*R2: R-squared from least squares fit to linear function
    # veer: wind veer, slope of linear function beta = WDz - WDhub = veer*(z - zhub)
    # veer_R2: R-squared from least squares fit to linear function
    tdim = Sz.shape[0]
zdim = Sz.shape[1]
Rrot = 0.5*(z[-1]-z[0])
Hhub = 0.5*(z[-1]+z[0])
ihub = int(0.5*len(zrot))
Arotor = np.pi*(Rrot)**2
Uz = -Sz*np.sin(np.pi*WDz/180.)
Vz = -Sz*np.cos(np.pi*WDz/180.)
Shub = Sz[:,ihub]
WDhub = WDz[:,ihub]
def cz(x,R,H):
    return 2.*(R**2 - (x-H)**2)**0.5
sumA = np.zeros((Sz.shape[0]))
veer = np.zeros((Sz.shape[0]))
for i in range(0,zdim-1):
    Ai = integrate.quad(cz, z[i], z[i+1], args = (Rrot,Hhub))
    Si = 0.5*(Sz[:,i+1]+Sz[:,i])
    Ui = 0.5*(Uz[:,i+1]+Uz[:,i])
    Vi = 0.5*(Vz[:,i+1]+Vz[:,i])
WDi = 180. + np.arctan2(Ui,Vi)*180./np.pi
betai = WDi - WDhub
sumA = sumA + Ai[0]*(Si*np.cos(np.pi*betai/180.))**3

REWS = (sumA/Arotor)**(1./3.)

alpha = np.zeros(tdim); alpha_stderr = np.zeros(tdim); alpha_R2 = np.zeros(tdim)
veer = np.zeros(tdim); veer_stderr = np.zeros(tdim); veer_R2 = np.zeros(tdim)
for it in range(0,tdim):
    regmodel = sm.OLS(np.log(Sz[it,:]/Shub[it]), np.log(z/Hhub))
    results = regmodel.fit()
    alpha[it] = results.params[0]
    alpha_stderr[it] = results.bse[0]
    alpha_R2[it] = results.rsquared
    regmodel = sm.OLS(WDz[it,:]-WDhub[it], z - Hhub)
    results = regmodel.fit()
    veer[it] = results.params[0]
    veer_stderr[it] = results.bse[0]
    veer_R2[it] = results.rsquared

return REWS, Shub, WDhub, alpha, alpha_R2, veer, veer_R2

Srews_obs = interpolate.interp2d(zf_obs[30:34],date_obs,S_obs_w.values[:,30:34])(zrot,
                               date_obs)
Urews_obs = interpolate.interp2d(zf_obs[30:34],date_obs,U_obs_w.values[:,30:34])(zrot,
                               date_obs)
Vrews_obs = interpolate.interp2d(zf_obs[30:34],date_obs,V_obs_w.values[:,30:34])(zrot,
                               date_obs)
WDrews_obs = 180. + np.arctan2(Urews_obs,Vrews_obs)*180./np.pi

REWS_obs, Shub_obs, WDhub_obs, alpha_obs, alpha_R2_obs, veer_obs, veer_R2_obs =
                               rotor(zrot,Srews_obs,WDrews_obs)

REWS = []; Shub = []; WDhub = []
alpha = []; alpha_R2 = []; veer = []; veer_R2 = []
for isim in range(0,Nsim):
    Srews_sim = interpolate.interp2d(z[isim],np.ravel(t[isim].values),S[isim].
                                values)(zrot,np.ravel(t[isim].values))
    Urews_sim = interpolate.interp2d(z[isim],np.ravel(t[isim].values),U[isim].
                                values)(zrot,np.ravel(t[isim].values))
    Vrews_sim = interpolate.interp2d(z[isim],np.ravel(t[isim].values),V[isim].
                                values)(zrot,np.ravel(t[isim].values))
    WDrews_sim = 180. + np.arctan2(Urews_sim,Vrews_sim)*180./np.pi
    REWS0, Shub0, WDhub0, alpha0, alpha_R20, veer0, veer_R20 =
                               rotor(zrot,Srews_sim,WDrews_sim)
    REWS.append(REWS0)
    Shub.append(Shub0)
    WDhub.append(WDhub0)
    alpha.append(alpha0)
    alpha_R2.append(alpha_R20)
    veer.append(veer0)
    veer_R2.append(veer_R20)

[43]: # plot QoIs

(continues on next page)
figname = siteID+'\_QoIs_rotor.png'

taxis_label = 'UTC time in hours since ' + datefrom.strftime('%Y-%m-%d %H:%M') + ', $D$ \rightarrow $ = ' + \%.0f\(Drot\) + ' m, $H$ = ' + \%.0f\(Hhub\) + ' m'

fig, ax = plt.subplots(2, 2, sharex='col', figsize=(9,6), dpi=300)
ax[0][0].plot(date_obs[::3], REWS_obs[::3], 'ok', color='grey', label = 'obs ')
for iplot in range(0,len(plotsim)):
    isim = plotsim[iplot]
    ax[0][0].plot(t[isim], REWS[isim], linespec[iplot],linewidth = lwidth[iplot], label = simID[isim], markevery=Nm)
ax[0][0].set_ylabel(r'$REWS$ \ [$m \ s^{-1}$]')
ax[0][0].grid(which='major',color='grey',linestyle=':)
ax[0][0].set_ylim([2., 14.])
ax[0][0].xaxis.set_major_formatter(hoursFmt)
ax[0][0].set_xticks(tticks)
ax[0][0].set_xlim([mdates.date2num(datefrom), mdates.date2num(dateto)])
ax[0][0].set_xlabel(taxis_label)
ax[0][0].xaxis.set_label_coords(1.2, -0.1)
ax[0][1].plot(date_obs[::3], alpha_obs[::3], 'ok', color='grey', label = 'obs ')
for iplot in range(0,len(plotsim)):
    isim = plotsim[iplot]
    ax[0][1].plot(t[isim], alpha[isim], linespec[iplot],linewidth = lwidth[iplot], label = simID[isim], markevery=Nm)
ax[0][1].set_ylabel(r'Wind Shear $\alpha$')
ax[0][1].grid(which='major',color='grey',linestyle=':)
ax[0][1].set_ylim([-0.2, 1.0])

ax[1][0].plot(date_obs[::3], WDhub_obs[::3], 'ok', color='grey', label = 'obs ')
for iplot in range(0,len(plotsim)):
    isim = plotsim[iplot]
    ax[1][0].plot(t[isim], WDhub[isim], linespec[iplot],linewidth = lwidth[iplot], label = simID[isim], markevery=Nm)
ax[1][0].set_ylabel(r'$WD_{hub}$ \ [\textdegree]')
ax[1][0].grid(which='major',color='grey',linestyle=':)
ax[1][0].set_xlim([60., 180.])
ax[1][0].set_xticks(tticks)
ax[1][0].set_xlabel((mdates.date2num(datefrom), mdates.date2num(dateto))
ax[1][0].set_xlabel(taxis_label)
ax[1][0].xaxis.set_label_coords(1.2, -0.1)

ax[1][0].plot(date_obs[::3], veer_obs[::3], 'ok', color='grey', label = 'obs ')
for iplot in range(0,len(plotsim)):
    isim = plotsim[iplot]
    ax[1][0].plot(t[isim], veer[isim], linespec[iplot],linewidth = lwidth[iplot], label = simID[isim], markevery=Nm)
ax[1][0].set_ylabel(r'Wind Veer $\psi$')
ax[1][0].grid(which='major',color='grey',linestyle=':)
ax[1][0].set_xlim([-0.1, 0.6])
ax[1][0].set_xticks(tticks)
ax[1][0].set_xlabel((mdates.date2num(datefrom), mdates.date2num(dateto))

ax[0][1].legend(prop={'size':8}, bbox_to_anchor=(1.5, 1), ncol=1, loc='upper right')
fig.subplots_adjust(wspace=.25)
plt.show()

#fig.savefig(figname, bbox_inches='tight', dpi = 300)

figcaption = ('**Fig 4. Time series of rotor rotor-based quantities of interest used for validation**')
Fig 4. Time series of rotor rotor-based quantities of interest used for validation

```python
from IPython.display import display
pd.options.display.float_format = '{:,.2f}'.format  # format for output data in tables

# Compute error metrics

def mae(xtrue, xpred, norm):
    # Normalized Mean Absolute Error
    # xtrue: series with true values
    # xpred: series with predicted values
    nonan = ~np.isnan(xtrue)
    if norm == 'True':
        xtrue0 = xtrue[nonan]/np.mean(xtrue[nonan])
        xpred0 = xpred[nonan]/np.mean(xtrue[nonan])
    else:
        xtrue0 = xtrue[nonan]
        xpred0 = xpred[nonan]

    N = len(xtrue0)
    abserr = np.abs(xpred0 - xtrue0)
    MAE = np.sum(abserr)/N
    return MAE

# Errors with respecto to observations
Metrics = pd.DataFrame(np.zeros((Nsim+1, 5)),
    columns = ['REWS', 'Shub', 'WDhub', 'alpha', 'veer'],
    index = ['MAE'] + simID)
Metrics.loc['units',:] = np.array(['m s⁻¹', 'm s⁻¹', 'deg', '-', '-'])
```

3.1. Wind Conditions
for isim in range(0,Nsim):
    Metrics.loc[(simID[isim]),'REWS'] = mae(REWS_obs,REWS[isim],'False')
    Metrics.loc[(simID[isim]),'Shub'] = mae(Shub_obs,Shub[isim],'False')
    Metrics.loc[(simID[isim]),'WDhub'] = mae(WDhub_obs,WDhub[isim],'False')
    Metrics.loc[(simID[isim]),'alpha'] = mae(alpha_obs,alpha[isim],'False')
    Metrics.loc[(simID[isim]),'veer'] = mae(veer_obs,veer[isim],'False')

tablecaption = ("**Table 1: $\text{MAE}_{\text{obs}}$ with respect to observations**")
display(Markdown(tablecaption))
display(Metrics)

| Table 1. $\text{MAE}_{\text{obs}}$ with respect to observations |
|------------------|------------------|------------------|------------------|------------------|
| REWS | Shub | WDhub | alpha | veer |
| MAE  | 0.00 | 0.00  | 0.00  | 0.00  | 0.00  |
| WRF-YSU | 1.37 | 1.48  | 11.59 | 0.13  | 0.08  |
| ke_T2 | 1.42 | 1.54  | 12.72 | 0.14  | 0.08  |
| S1_T2 | 1.87 | 1.85  | 11.40 | 0.19  | 0.07  |
| kl_T2 | 1.84 | 1.81  | 10.88 | 0.19  | 0.07  |
| ke_T2wt | 1.40 | 1.49  | 12.71 | 0.13  | 0.08  |
| ke_Tsk | 1.63 | 1.91  | 16.39 | 0.15  | 0.10  |
| ke_T2obs | 1.75 | 1.77  | 11.66 | 0.12  | 0.09  |
| noTadv | 1.44 | 1.30  | 13.77 | 0.17  | 0.06  |
| noTadvUadv | 1.76 | 1.87  | 11.78 | 0.18  | 0.07  |
| noTadvUadv_Sg0 | 3.21 | 3.20  | 16.17 | 0.29  | 0.12  |
| UVTnud80 | 1.42 | 1.36  | 10.33 | 0.14  | 0.07  |
| UVnud120 | 1.26 | 1.17  | 11.85 | 0.14  | 0.09  |
| UVnud200 | 0.71 | 0.76  | 9.36  | 0.09  | 0.04  |
| UVnud200_tau10 | 0.16 | 0.19  | 3.80  | 0.05  | 0.02  |
| UVnud400 | 0.59 | 0.73  | 10.13 | 0.12  | 0.05  |
| UVnud200 | 0.66 | 0.80  | 10.49 | 0.12  | 0.05  |
| UVnud200_tau30 | 0.45 | 0.49  | 7.21  | 0.10  | 0.05  |
| UVnud200tau10 | 0.26 | 0.34  | 4.39  | 0.08  | 0.05  |
| units | m/s-1 | m/s-1 | deg | - | - |

[39]: # Errors with respect to reference simulation
refsim = 1

Metrics2 = pd.DataFrame(np.zeros((Nsim+1, 5)),
                         columns = ['REWS','Shub','WDhub','alpha','veer'],
                         index = ['MEAE'] + simID)
Metrics2.loc['units',:] = np.array(['m/s-1','m/s-1','deg','-','-'])
for isim in range(0,Nsim):
    Metrics2.loc[(simID[isim]),'REWS'] = mae(REWS[refsim],REWS[isim],'False')
    Metrics2.loc[(simID[isim]),'Shub'] = mae(Shub[refsim],Shub[isim],'False')
    Metrics2.loc[(simID[isim]),'WDhub'] = mae(WDhub[refsim],WDhub[isim],'False')
    Metrics2.loc[(simID[isim]),'alpha'] = mae(alpha[refsim],alpha[isim],'False')
    Metrics2.loc[(simID[isim]),'veer'] = mae(veer[refsim],veer[isim],'False')

tablecaption = ("**Table 2: $\text{MAE}_{\text{ref}}$ with respect to $\text{ke\_T2}$**")
display(Markdown(tablecaption))
display(Metrics)

**Table 2: $\text{MAE}_{\text{ref}}$ with respect to $\text{ke\_T2}$**

| Table 2: $\text{MAE}_{\text{ref}}$ with respect to $\text{ke\_T2}$ |
|------------------|------------------|------------------|------------------|------------------|
| REWS | Shub | WDhub | alpha | veer |
| MAE  | 0.00 | 0.00  | 0.00  | 0.00  | 0.00  |
Discussion

See discussion about results in [2].

Conclusions

The GABLS3 diurnal cycle case has been revisited and evaluated in terms of wind energy specific metrics. Instead of using the adjusted mesoscale tendencies of the original GABLS3 set-up, the mesoscale tendencies computed by WRF are directly used to force the SCM. Momentum budget analysis shows the relative importance of the different forcing terms in the momentum equations. By spatial and temporal averaging, the high-frequency fluctuations due to microscale effects are filtered out. Using mesoscale tendencies to drive the SCM results in consistent flow fields compared to the WRF simulation, even though the more simplified physics of the ABL.

By sensitivity analysis on the mesoscale tendencies, it is shown that the main driver of the ABL is the time and height dependent horizontal pressure gradient. Advection terms come with high uncertainties and hour-to-hour they can lead to large errors. Nevertheless, their impact in terms of quantities of interest’s aggregated errors is positive.

The $k - \varepsilon$ model of Sogachev et al. (2012) presents better performance than the lower-order turbulence closure models. Considering surface boundary conditions for the potential temperature equation, prescribing the surface temperature by indirectly introducing the WRF 2-m temperature with MOST is more adequate than using the skin temperature or the observed 2-m temperature.

Instead of adjusting at mesoscale, corrections are introduced at microscale through observational profile nudging, to make use of the routine measurements collected in wind energy campaigns. Mast-based and lidar-based profiler set-ups are compared to show the added value of measuring at greater heights than the hub-height, main advantage of lidar systems. Sensitivity to the nudging time-scale is large, specially to compensate errors introduced by the mesoscale advection forcing.

References

Acknowledgements

This work was produced with funding from the ‘MesoWake’ Marie Curie International Outgoing Fellowship (FP7-PEOPLE-2013-IOF, European Commission’s grant agreement number 624562).

Scope

The GABLS3 case has been selected in the NEWA project as a baseline exercise for the design of mesoscale-to-microscale methodologies for wind resource assessment. The case is suitable for the development of microscale wind farm models that incorporate realistic forcing, derived from a mesoscale model, along a typical diurnal case that leads to the development of a nocturnal low-level jet. Challenges of this case include: incorporating time- and height-dependent mesoscale forcing in microscale models, turbulence modeling at varying atmospheric stability conditions, defining suitable surface boundary conditions for momentum and heat and characterization of the wind profile in (non-logarithmic) LLJ conditions.

Data Accessibility

Data is provided open-access for registered participants. Observational data is available at the GABLS3-SCM KNMI website.

Objectives

Wind-energy specific objectives of the benchmark include:

- Demonstrate the capability of wind energy ABL models to incorporate realistic mesoscale forcing
- Implement surface boundary conditions suitable for wind assessment studies using mesoscale simulation data and/or observations (typical of wind energy campaigns)
- Develop suitable model calibration strategies for wind energy applications or, in other words, how to best use available measurements (typical of wind energy campaigns) to correct meso-micro predictions
- Define suitable metrics for validation of ABL models based on wind energy quantities of interest

By “typical wind energy campaigns” we would like to encourage modellers to prioritize observations that are common place in wind resource assessment campaigns (80 masts with velocity and temperature measurements, lidar profilers measuring up to 400 m).
**Site Description**

The GABLS3 set-up is described in Bosveld et al. [8]. The case analyzes the period from 12:00 UTC 1 July to 12:00 UTC 2 July 2006, at the KNMI-Cabauw Experimental Site for Atmospheric Research (CESAR), located in the Netherlands (51.971°N, 4.927°E), with a distance of 50 km to the North Sea at the WNW direction [12]. The elevation of the site is approximately -0.7 m, surrounded by relatively flat terrain characterized by grassland, fields and some scattered tree lines and villages (Figure 1). The mesoscale roughness length for the sector of interest (60º - 120º) is 15 cm.

![Figure 1: Land-use map of a 30x30 km area around the Cabauw site (figure from KNMI's HYDRA project website)](image)

**Measurement Campaign and Case Selection**

The CESAR measurements are carried out at a 200-m tower, free of obstacles up to a few hundred meters in all directions. The measurements include 10-min averaged vertical profiles of wind speed, wind direction, temperature and humidity at heights 10, 20, 40, 80, 140 and 200 m, as well as surface radiation and energy budgets. Turbulence fluxes are also monitored at four heights: 3, 60, 100 and 180 m. A RASS profiler measures wind speed, wind direction and virtual temperature above 200 m.

The selection criteria for GABLS3 consisted on the following filters applied to a database of 6 years (2001 - 2006): stationary synoptic conditions, clear skies (net longwave cooling > 30 W m² at night), no fog, moderate geostrophic winds (5 to 19 m s⁻¹, with less than 3 m s⁻¹ variation at night) and small thermal advective tendencies. Out of the 9 diurnal cycles resulting from this filtering process, the one that seemed more suitable was finally selected: 12:00 UTC 1 July to 12:00 UTC 2 July 2006.

More information about the case background and set-up can be found in the official GABLS3 website.
Input data

The case set-up and input data of the original GABLS3 case can be found in the KNMI website. This is useful if you want to compare with published results of the original SCM model intercomparison. In the original GABLS3 set-up, the simulated mesoscale tendencies are adjusted to produce a better match with the surface geostrophic wind obtained from a network of synoptic stations and the wind speed at 200-m measured at the Cabauw tower. Initial profiles are based on soundings measured near the Cabauw mast.

Alternatively, you can use inputs generated entirely from a WRF simulation, as described in [1][2]. Here, instead of using observed initial profiles and adjusted mesoscale forcings you can use initial profiles and forcing produced directly from a mesoscale simulation. This is more representative of a wind energy model-chain set-up, where the inputs of a microscale model are generated by a “wind atlas” methodology that doesn’t normally include corrections with local measurements. Instead, local adjustments are allowed at the microscale level by incorporating onsite measurements as if these measurements were part of a typical wind resource assessment campaign.

The WRF simulation is based on a one-way nesting configuration of three concentric square domains centered at the Cabauw site, of the same size 181x181, and at 9, 3 and 1 km horizontal resolution. The vertical grid, approximately 13 km high, is based on 46 terrain-following (eta) levels with 24 levels in the first 1000 m, the first level at approximately 13 m, a uniform spacing of 25 m over the first 300 m and then stretched to a uniform resolution of 600 m in the upper part. The U.S. Geological Survey (USGS) land-use surface data, that comes by default with the WRF model, is used together with the unified Noah land-surface model to define the boundary conditions at the surface. Other physical parameterizations used are: the rapid radiative transfer model (RRTM), the Dudhia radiation scheme and the Yonsei University (YSU) first-order PBL scheme. The simulation uses input data from ERA-Interim with a spin-up time of 24 hours. The WRF set-up follows the reference configuration of Kleczek et al [3], who run a sensitivity analysis of WRF showing reasonably good results at reproducing the nocturnal LLJ.

A NetCDF file is provided with the following information:

- Site coordinates and Coriolis parameter
- Time-height 2D arrays of velocity components (U,V,W) and potential temperature (Th)
- Time-height 2D arrays of mesoscale forcings (tendencies): geostrophic wind (Ug, Vg), advective wind (Uadv, Vadv) and advective potential temperature (Thadv)
- Time array of surface-layer quantities: friction velocity (уст), kinematic heat flux (wt), 2-m temperature (T2), skin temperature (TSK), surface pressure (Psfc)

Units, dimensions and variables description are all provided in the NetCDF file. Momentum tendencies (Figure 2) are provided in [m s-1] and should be multiplied by the Coriolis parameter to obtain appropriate forces in [m s-2]. For convenience, we have omitted information about humidity since the assumption of dry-atmosphere is typically adopted by wind energy flow models.
Validation data

The following quantities of interest (QoI) will be evaluated as described in [1][2], using a reference rotor size of 160 m diameter at a hub-height of 120 m (~ 7 MW turbine):

- Rotor equivalent wind speed (REWS)
- Hub-height wind direction (WDhub)
- Turbulence intensity at hub-height (TIhub)
- Wind shear (power-law exponent $\alpha$) and wind veer (slope of linear fit to wind direction differences $\psi$) across the rotor plane
- Surface-layer quantitites: $T_2$, $ust$, $wt$ and $z/L$

The evaluation consists on time-series plots of these QoIs along the diurnal cycle and mean-absolute error (MAE) integrated over the whole cycle.

Model runs

The benchmark is mainly developed for microscale models that make use of the input data described above. However, mesoscale or multi-scale (online meso-micro) simulations are also welcome. The following suggestions are provided to guide the model runs:

- We shall use the 2-m temperature ($T_2$) as our most practical reference to deduce the potential temperature surface boundary conditions using Monin Obukhov similarity theory, since this variable is routinely measured in measurement campaigns and is part of the standard output of meteorological models.
• Simulations may be based entirely on the mesoscale input data or incorporate measurements from the Cabauw mast. Priority should be given to measurements that can be found in “typical wind energy campaigns” (80 masts with velocity and temperature measurements, lidar profilers measuring up to 400 m).

• Sensitivity analysis of mesoscale models can be used to quantify the input uncertainty derived from the spread of the ensemble of simulations.

• Online multi-scale simulations models can be used as a reference for microscale models that are coupled to mesoscale asynchronously through the input data. To allow this comparison multi-scale simulations should be also run with ERA-Interim input data.

• Microscale models using Sogachev et al. (2012) k-ε turbulence model shall use this set of constants: \( \kappa = 0.4 \), \( C_\varepsilon 1 = 1.52 \), \( C_\varepsilon 2 = 1.833 \), \( \sigma_k = 2.95 \), \( \sigma_\varepsilon = 2.95 \) and \( C_\mu = 0.03 \)

If resources allow, please use a spin-up time of 24 hours as in the input data.

Output data

Data should be provided in a single NetCDF file as described in the python template. The following output variables are requested:

• Time-height 2D arrays of: velocity components, potential temperature and turbulent kinetic energy

• Time 1D array of surface-layer quantities: friction velocity (\( u_{st} \), at 3 m), kinematic heat flux (\( w_t \), at 3 m) and 2-m temperature (\( T_2 \))

• Time in hours since 2006-07-01 12:00 UTC

• Heights in meters (please provide model levels at least up to 4000 m)

References


Acknowledgements

This benchmark has been produced within a Marie Curie International Outgoing Fellowship, MesoWake project, with funding from the European Union’s Seventh Framework Programme for research, technological development and demonstration under grant agreement no 624562.

References

NEWA Meso-Micro Challenge for Wind Resource Assessment

Background

Scope and Objectives

References

3.1.3 Benchmarks

Monin Obukhov Similarity Theory (MOST)
Background

Monin Obukhov (M-O) similarity theory (Monin and Obukhov, 1954) sets the point of departure of modern micrometeorology (Foken, 2006). It is valid in the surface layer, i.e. approximately in the first 10% of the ABL, where Coriolis effects are negligible compared to friction, and under stationary and horizontally homogeneous conditions with no radiation. In these ideal conditions the vertical variations of wind direction, shear stress, heat and moisture fluxes are constant. M-O theory states that any dimensionless turbulence characteristic will only depend on a reduced set of scales. In addition to friction velocity ($u^*$) and the height above the ground ($z$), as basic scales in neutral conditions, the surface (virtual) potential temperature $\Theta_0$ and kinematic heat flux ($w_\Theta$) are also required in thermally stratified conditions. The Obukhov length scale $L$ is made of a combination of these parameters,

$$L = \frac{\kappa g}{\nu^2} \left( \frac{z}{L} \right)$$

where $g$ is the gravity and $\kappa$ is the von Karman constant. A dimensionless height $z/L$ is used as stability parameter ($z/L < 0$ for unstable, $z/L > 0$ for stable and $z/L = 0$ for neutral conditions). Any dimensionless turbulent characteristic will depend solely on this parameter. By integration of the velocity and potential temperature gradients, well-known logarithmic profiles are obtained:

$$U(z) = \frac{1}{\kappa} \ln \left( \frac{z}{z_0} \right)$$

where $U$ is the mean velocity at height $z$, $z_0$ and $z_0t$ are the roughness length for momentum and heat, $\Theta_0^*$ is a temperature scale, $\bar{\Theta}$ is the mean (virtual) potential temperature at height $z$, $C_\mu$ is a constant, and $\Phi_x$ and $\Psi_x$ are stability functions obtained from flux-profile experiments in flat terrain (see for instance, Panofsky and Dutton, 1984). $k$ is the turbulent kinetic energy and $\epsilon$ is the turbulent dissipation rate, also a function of the stability parameter.

M-O theory is used to design wind engineering surface layer models. When an empty domain is simulated in steady-state with homogeneous surface conditions the flow should produce the fully-developed log-profiles predicted by the theory. These are the conditions that will be simulated in this test case.

For instance, Richards and Hoxey (1983) calibrated the RANS $k-\epsilon$ turbulence model by enforcing consistency with M-O theory in the surface layer in neutral conditions. Alinot and Masson (2005) followed the same approach to derive consistency conditions for a $k-\epsilon$ model in stratified conditions.

Scope and Objectives

This test case should be mandatory for anyone using surface layer models since it describes the fundamental physics in flat terrain conditions. Simulations of homogeneous profiles in an empty domain also allow to check the equilibrium of the wall functions with the turbulent flow model (Blocken et al., 2007). In a building-block approach, it shall be used as a precursor validation to any other benchmark.

References


Leipzig

Background

Scope and Objectives

References

Askervein

Background

Scope and Objectives

References

Bolund

Background

Scope and Objectives

References

3.1.4 Open Source Models

- CFDWind
Chapter 3. Modeling Communities
CHAPTER 4

Indices and tables

- genindex
- modindex
- search