Welcome to Dynamiks's documentation!

Dynamic wind system simulator

Source code repository and issue tracker:

https://gitlab.windenergy.dtu.dk/DYNAMIKS/dynamiks

Contents

Main components

Wind farm flow models

- DWM flow simulation
  - ParticleDeficitGenerator
  - Particle motion models
  - Wind direction

Common

- Visualization
  - Visualization methods
- Views
  - Points
  - 1D Views
  - 2D Views
  - 3D Views
  - View
  - MultiView
  - View arguments
- Interaction
  - Custom time-stepping loop
  - Step handlers
- Dynamic inflow
  - Dynamic wind speed and direction in the DWMFlowSimulation setup
  - Add changes to the turbulence field
  - Add changes via the mean wind speed function
  - Change the transport speed and direction
  - Modeling a time series of wind direction and speed
- Wind farm yaw control, simple
  - Find optimial yaw settings
  - Dynamiks wind farm control
  - Issues and simplifications
- Implementation details and choises
  - Mapping time to position in turbulence field
  - Deficit interpolation
  - Overtaken particles

- Linear or pchip particle path

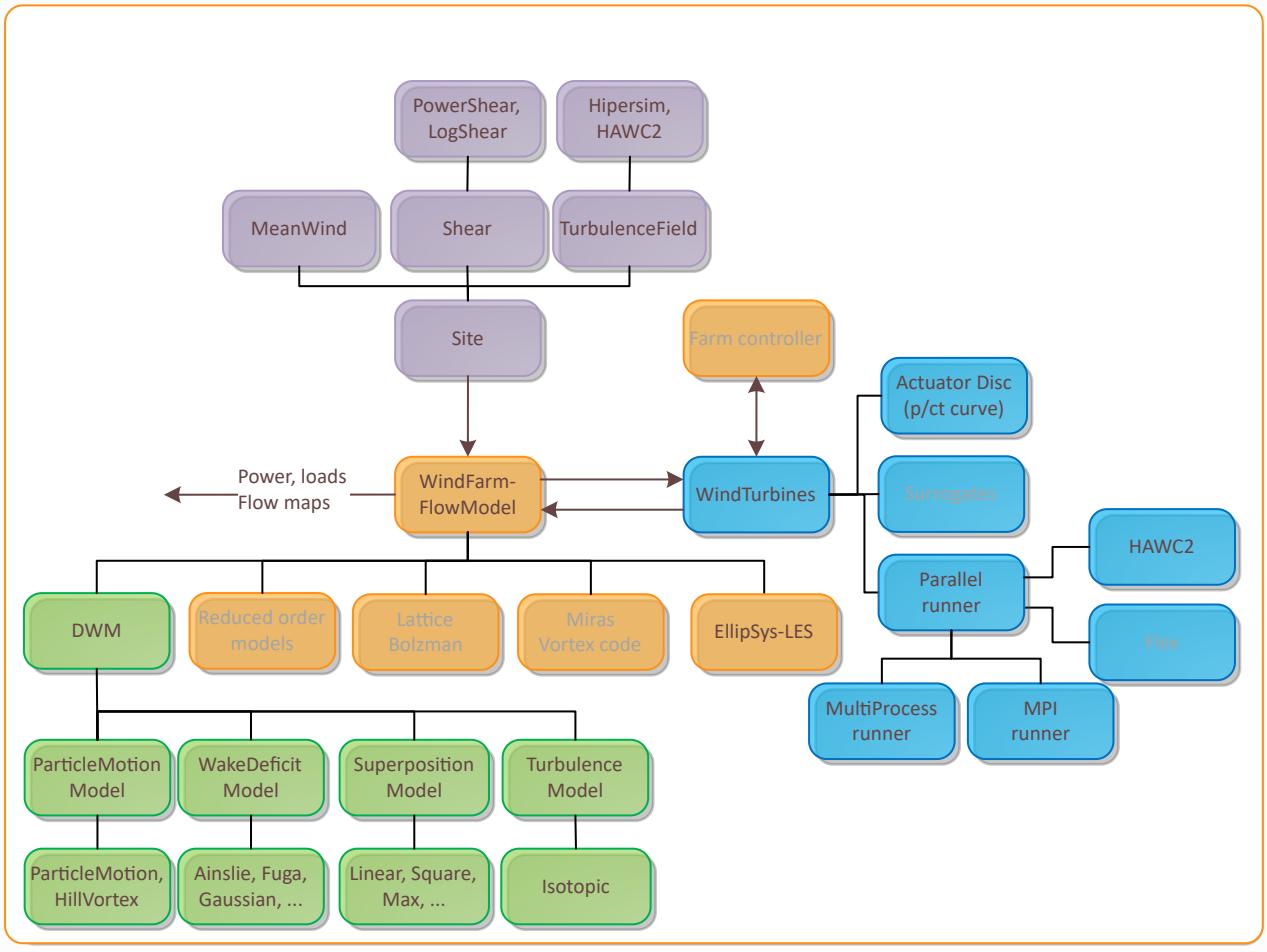
Next

Edit on Gitlab launch binder

Overview

Dynamiks is DTU's Dynamic Wind System Simulator.

It is intended to be a multi-fidelity, modular and flexible framework that makes it easy to couple some of the tools tools of different fidelity, e.g. couple HAWC2 or a simple Actuator disc wind turbine with Dynamic Wake Meandering (DWM), a vortex code (Miras) or CFD-LES (Ellipsys3D).



Couplings to boxes with grey text are not implemented yet

The three main components are:

- Site
- WindTurbines
- WindFarmFlowModel

[ ]:

Previous Next

Installation

```
pip install dynamiks
```

Newest development version can be installed by:

```
pip install git+https://gitlab.windenergy.dtu.dk/DYNAMIKS/dynamiks.git
```

Optional dependencies can be installed by:

```
pip install dynamiks[hawc2]
```

Developer installation

Clone the **dynamiks** repository:

```
git clone https://gitlab.windenergy.dtu.dk/DYNAMIKS/dynamiks.git
cd dynamiks
pip install -e .
```

Publications

HAWC2Farm (predecessor of Dynamiks)

The predecessor of Dynamiks was called HAWC2Farm. It was implemented by Jaime Liew during his PhD where also several publications were authored.

Dynamiks is a new framework, but some elements from HAWC2Farm has been reused. HAWC2Farm combines HAWC2 wind turbines with the DWM model from jDWM

- Liew, J.: HAWC2Farm, Zenodo [code], https://doi.org/10.5281/zenodo.8028485, 2023
- Liew, J.: jDWM, Zenodo [code], https://doi.org/10.5281/zenodo.8028555, 2023.
- Liew, J., Göçmen, T., Lio, A. W. H., and Larsen, G. Chr.: Extending the dynamic wake meandering model in HAWC2Farm: a comparison with field measurements at the Lillgrund wind farm, Wind Energ. Sci., 8, 1387–1402, https://doi.org/10.5194/wes-8-1387-2023, 2023.
- Liew J, Göçmen T, Lio WH, Larsen GC. Model-free closed-loop wind farm control using reinforcement learning with recursive least squares. Wind Energy. 2023; 1-15. https://doi.org/10.1002/we.2852
- Liew J, Andersen SJ, Troldborg N, Gmen T. LES verification of HAWC2Farm aeroelastic wind farm simulations with wake steering and load analysis.In: Journal of Physics: Conference Series. IOP Publishing; 2022; https://doi.org/10.1088/1742-6596/2265/2/022069
- Liew J, Larsen GC. How does the quantity, resolution, and scaling of turbulence boxes affect aeroelastic simulation convergence? In: Journal ofPhysics: Conference Series, Vol. 2265. IOP Publishing; 2022:32049; https://doi.org/10.1088/1742-6596/2265/3/032049

Edit on Gitlab launch binder

QuickStart

Quick start tutorial

[1]:
```
import matplotlib.pyplot as plt
import numpy as np
from dynamiks.utils import doc_utils # use cached animations in sphinx documentation
from dynamiks.utils.test_utils import tfp
from dynamiks.views import Points, XView, XYView
from py_wake.utils.plotting import setup_plot

ws = 10 # mean wind speed
ti = 0.05 # turbulence intensity
```

To setup a flow simulation we need a `Site` some `WindTurbines`

Site

The `Site` object is provides the wind speed and turbulence intensity at given position and time (x,y,z,time)

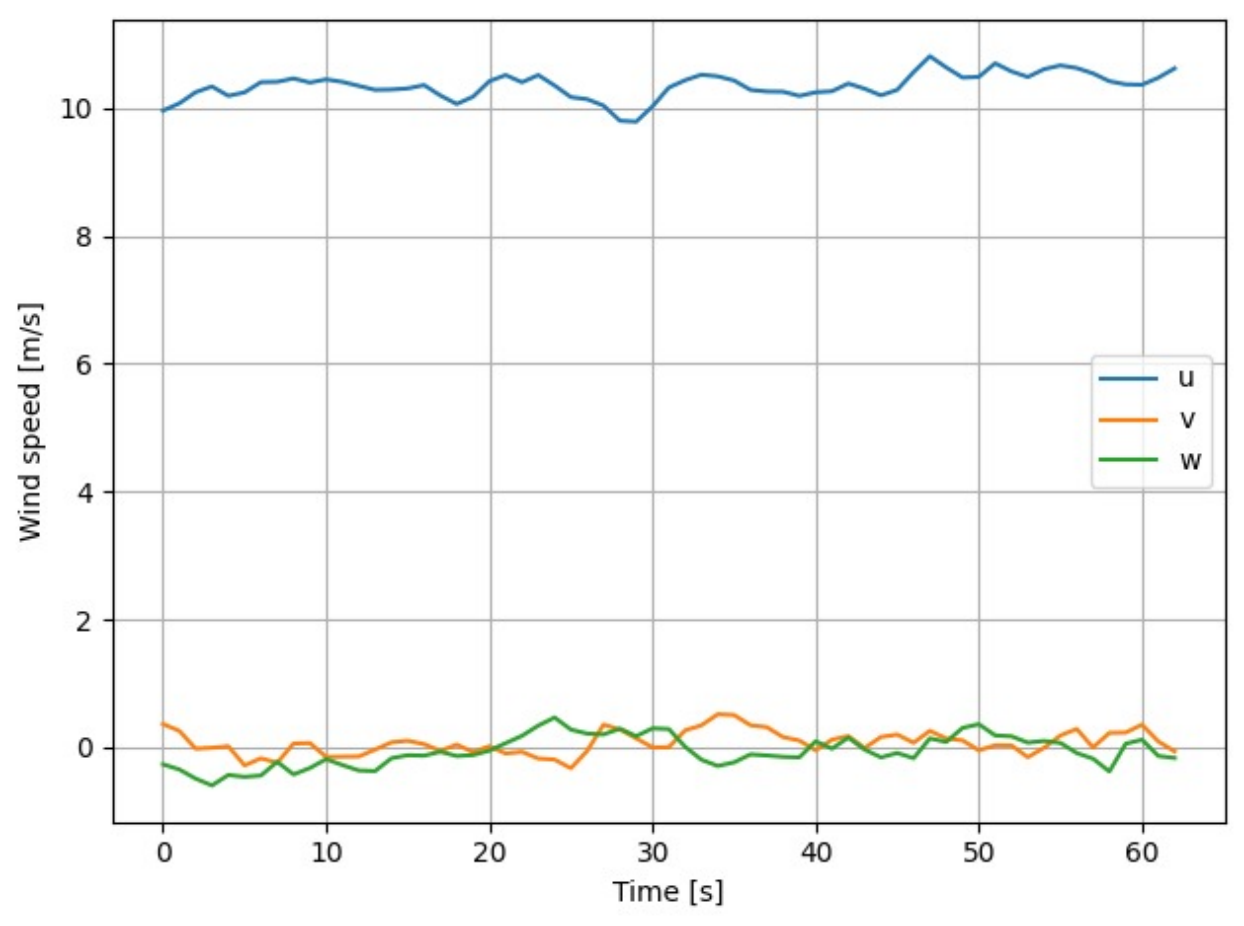
For DWM we can use a the `TurbulenceFieldSite`, which takes a turbulenceField object or function as input. In this example we will use a `MannTurbulenceField`

[2]:
```
from dynamiks.sites import TurbulenceFieldSite
from dynamiks.sites.turbulence_fields import MannTurbulenceField

tf = MannTurbulenceField.from_netcdf(filename = tfp + "mann_turb/hipersim_mann_l29.4_ae1.0000_g
tf.scale_TI(TI=ti, U=ws)
```

[3]:
```
site = TurbulenceFieldSite(ws=ws, turbulenceField=tf, turbulence_offset=(-2500,-200,20))
```

[4]:
```
time = range(200)
view = XView(x=range(200), y=0, z=70)
uvw = site.get_windspeed(view)
for i, n in enumerate('uvw'):
```

```
    plt.plot(uvw[i], label=n)
setup_plot(ylabel='Wind speed [m/s]', xlabel='Time [s]')
```



WindTurbines

The simplest wind turbine model provides power and a ct based on rotor center wind speed.

For this purpose PyWake wind turbines are sufficient. They can be used via the wrapper, `PyWakeWindTurbine`.

[5]:
```
from dynamiks.wind_turbines import PyWakeWindTurbines
from py_wake.examples.data.hornsrev1 import V80

wts = PyWakeWindTurbines(x=[0,500], y=[0,0],   # x and y position of two wind turbines
                         windTurbine=V80())

# print diameter and hub height
wts[0].diameter(), wts[0].hub_height()
```

[5]:
```
(array([80.]), array([70.]))
```

FlowSimulation

[6]:
```
from dynamiks.dwm import DWMFlowSimulation
from dynamiks.dwm.particle_deficit_profiles.ainslie import jDWMAinslieGenerator

fs = DWMFlowSimulation(site=site,
```

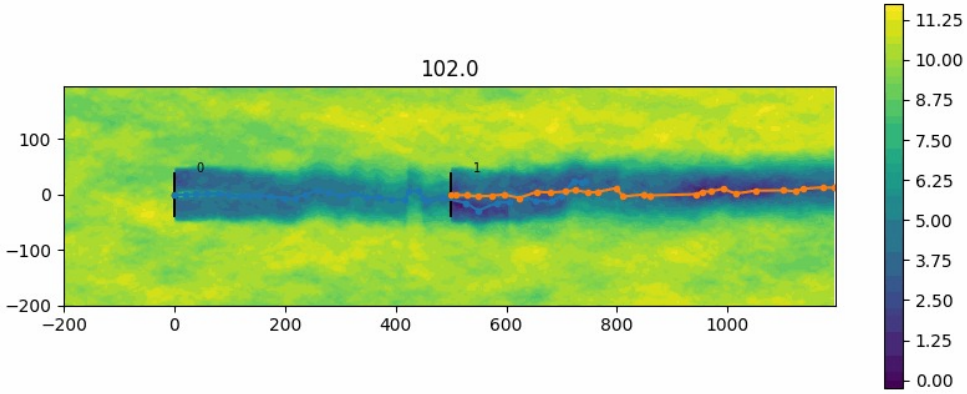
```
                        windTurbines=wts,
                        particleDeficitGenerator=jDWMAinslieGenerator(),
                        dt=1, # time step [s]
                        d_particle=.2, # distance between particles, normalized with wind turbin
                       )
```

[7]:
```
fs.run(100) # run 100s
```

Visualize flow

[8]:
```
fig = plt.figure(figsize=(10,4))
view = XYView(z=70, x= np.arange(-200,1200,5), y=np.arange(-200,200,5), ax=fig.gca())
fs.visualize(fs.time+10, view=view, id='QuickStart')
```

[8]:

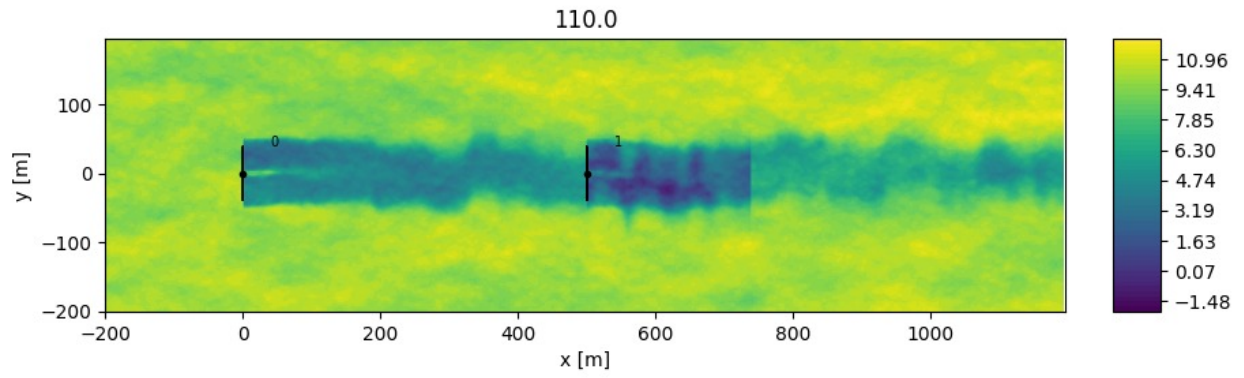


Animate flow

Make and save a gif animation with the next 10s

[9]:
```
fig = plt.figure(figsize=(10,4))
view = XYView(z=70, x= np.arange(-200,1200,5), y=np.arange(-200,200,5), ax=fig.gca())
ani = fs.animate(fs.time+10, filename='visualization.gif', view=view);
```

100% 10/10 [00:05<00:00, 1.64it/s]



Show the gif animation

[10]:
```
from IPython.display import HTML
```

```
HTML('<img src="visualization.gif">')
```

[10]: 

Previous Next

Wind turbines

[1]:
```
# setup site used in this tutorial
import matplotlib.pyplot as plt
import numpy as np
from dynamiks.utils import doc_utils # use cached animations in sphinx documentation
from dynamiks.sites.turbulence_fields import MannTurbulenceField
from dynamiks.utils.test_utils import tfp, DemoSite
from dynamiks.sites._site import TurbulenceFieldSite
from py_wake.utils.plotting import setup_plot

ws = 10
ti = 0.06
site = DemoSite(ws,ti)
```

Actuator disk wind turbines

Simple actuator disk wind turbines are defined in terms of

- Position
- Name
- Diameter
- Hubheight
- Power curve
- Ct curve

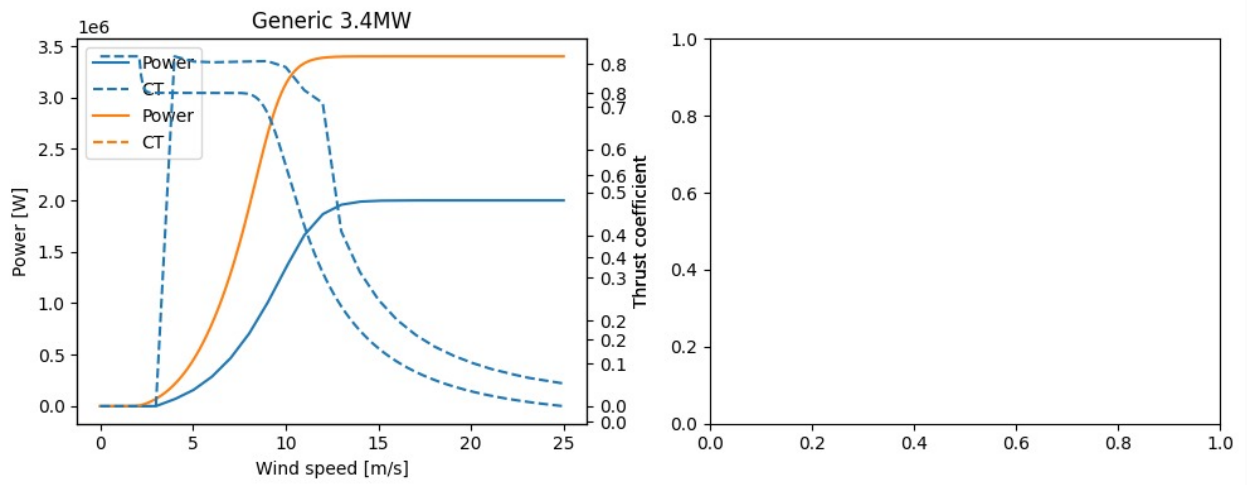
This turbine type is really fast but does not model the blades or structural loads

Actuator disk wind turbines can e.g. be generated from PyWake wind turbines (predefined, generic or custom)

[2]:
```
# instantiate a V80 and a generic 3.4MW PyWake wind turbines
from py_wake.wind_turbines import WindTurbines
from py_wake.examples.data.hornsrev1 import V80
from py_wake.wind_turbines.generic_wind_turbines import GenericWindTurbine
v80 = V80()
```

```
generic = GenericWindTurbine("Generic 3.4MW", diameter=130, hub_height=110, power_norm=3400, tu

# collect the two turbines in one WindTurbines object
pywake_windTurbines = WindTurbines.from_WindTurbine_lst([v80,generic])
```

[3]:
```
# plot power and ct curves
axes = plt.subplots(1,2, figsize=(10,4))[1]
for t, ax in zip(pywake_windTurbines.types(), axes):
    plt.sca(ax)
    pywake_windTurbines.plot_power_ct(type=t)
    plt.tight_layout()
```



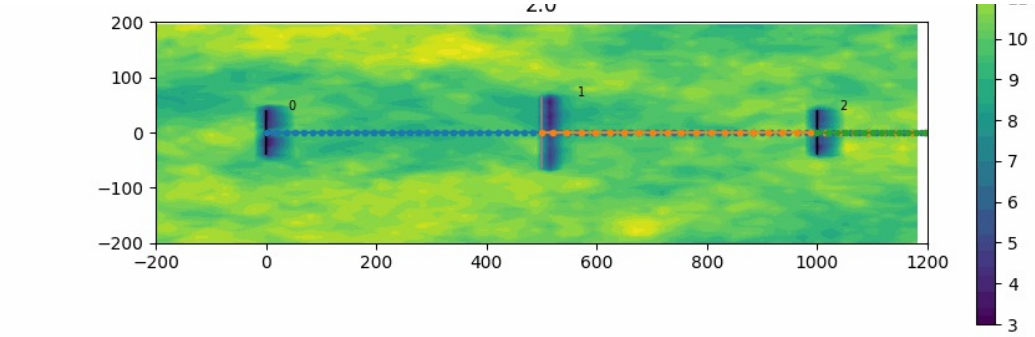
[4]:
```
# instantiate a dynamiks WindTurbines object
from dynamiks.wind_turbines import PyWakeWindTurbines

wts = PyWakeWindTurbines(x=[0,500,1000], y=[0,0,0],   # x and y position of two wind turbines
                         windTurbine=pywake_windTurbines,
                         types=[0,1,0])

# print diameter and hub height
for i,(xyz) in enumerate(wts.positions_east_north.T):
    print(f'WT{i}, diameter: {wts[i].diameter()}, hub height: {wts[i].hub_height()}, position:
```

```
WT0, diameter: [80.], hub height: [70.], position: (np.float64(0.0), np.float64(0.0), np.float6
WT1, diameter: [130.], hub height: [110.], position: (np.float64(500.0), np.float64(0.0), np.fl
WT2, diameter: [80.], hub height: [70.], position: (np.float64(1000.0), np.float64(0.0), np.flo
```

[5]:
```
from dynamiks.dwm.dwm_flow_simulation import DWMFlowSimulation
from dynamiks.dwm.particle_deficit_profiles.ainslie import jDWMAinslieGenerator
from dynamiks.views import XYView

fs = DWMFlowSimulation(site=site, windTurbines=wts, particleDeficitGenerator=jDWMAinslieGenerat
fs.run(10)
```

[6]:
```
ax = plt.figure(figsize=(10,4)).gca()
view = XYView(z=70, x=np.linspace(-200,1200), y=np.linspace(-200,200), ax=ax)
fs.visualize(fs.time+100, view=view, id='WindTurbines_AD')
```

[6]:



Output from wind turbines

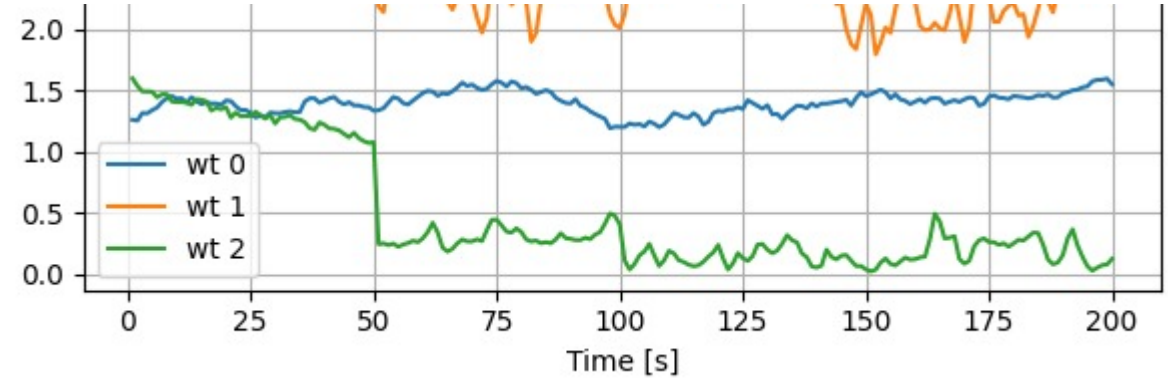
Wind turbines has a `sensor` object with a set of predefined sensors. Additional sensors can be added by the method `windTurbines.add_sensor`

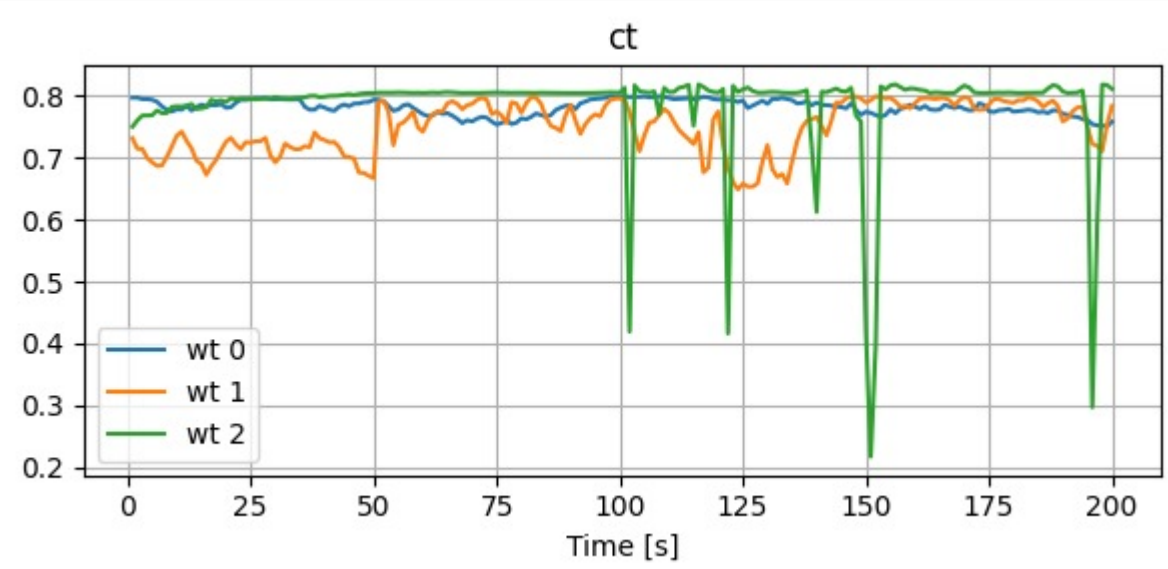
[7]:
```
help(wts.add_sensor)
```

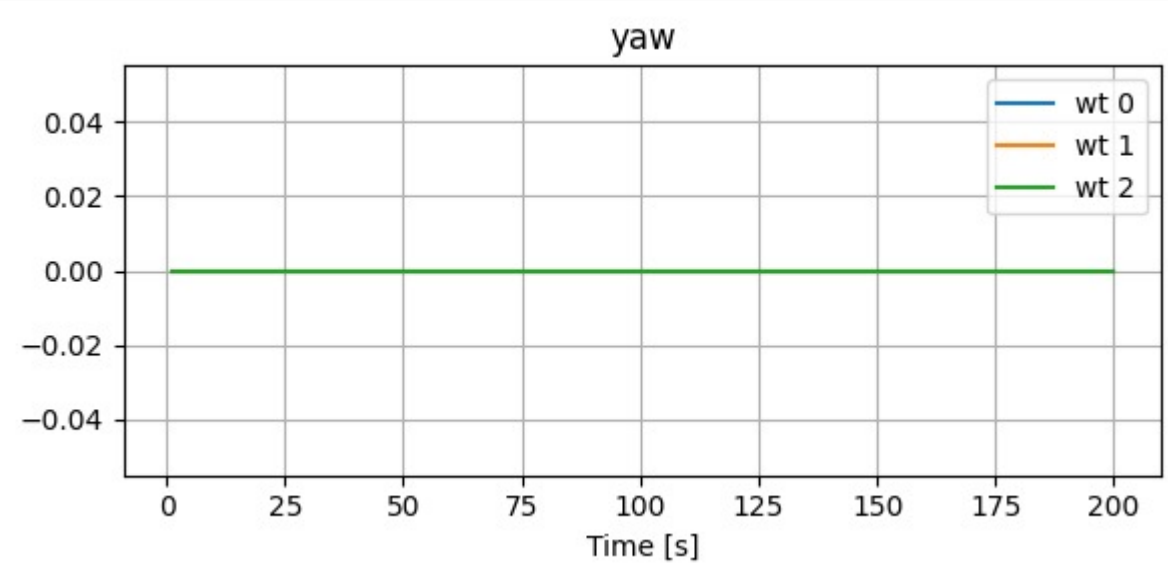
```
Help on method add_sensor in module dynamiks.wind_turbines._windTurbines:

add_sensor(name, getter=None, setter=None, expose=False, ext_lst=None) method of dynamiks.wind_
    add a wind turbine sensor

    Sensor values available using: windTurbines.sensors.<name>

    Parameters
    ----------
    name : str
        Name of sensor. Cannot contain spaces
    getter : function
        function, f(wt) -> sensor_value
    setter : function, optional
        function, f(wt, value) -> None,
    expose : boolean, optional
        if True, the getter/setter is exposed to the wind turbine enabling e.g. wt.yaw as a sho
    ext_lst : list or None
        List of sensor name extensions in case the sensor returns more than one value,
        e.g. the u,v and w components of the wind, see examples
    Examples
    --------
    windTurbines.add_sensor('power', lambda wt: wt.power())
    windTurbines.add_sensor('ws', ext_lst=['u','v','w'])" # adds the three sensors: ws_u, ws_v,
```

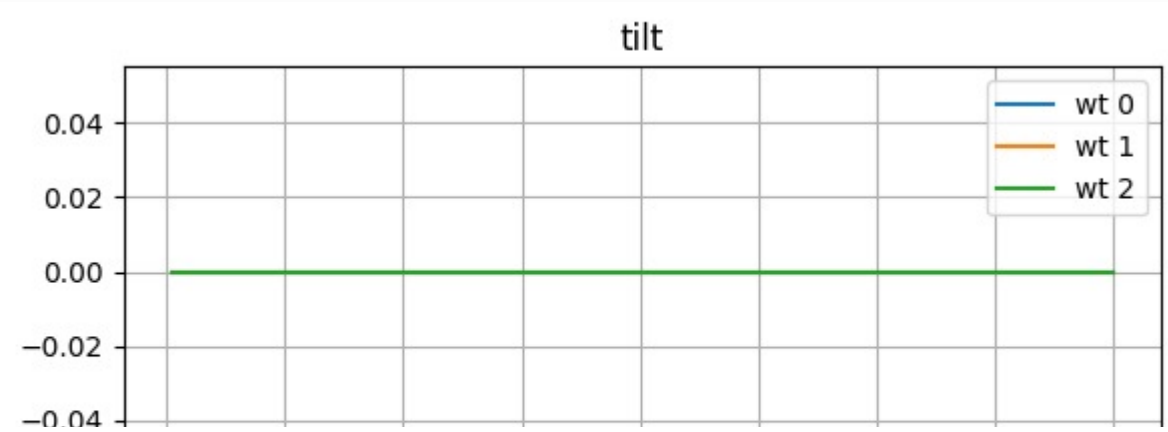
An xarray data array with all sensor data can be obtained as follows

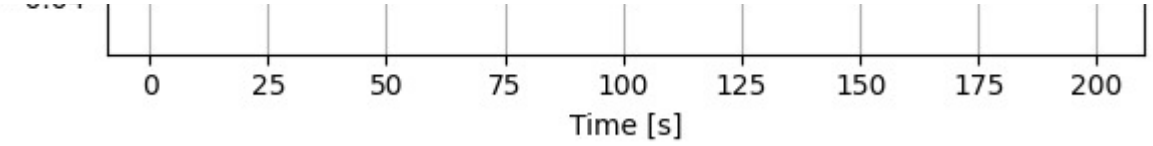
[8]:
```
fs.run(200)
da = wts.sensors.to_xarray()
for sensor in da.sensor:
    plt.figure(figsize=(6,3))
    for wt in da.wt:
        da.sel(sensor=sensor, wt=wt).plot(label=f'wt {wt.item()}')
    setup_plot(title=sensor.item(), xlabel='Time [s]')
```

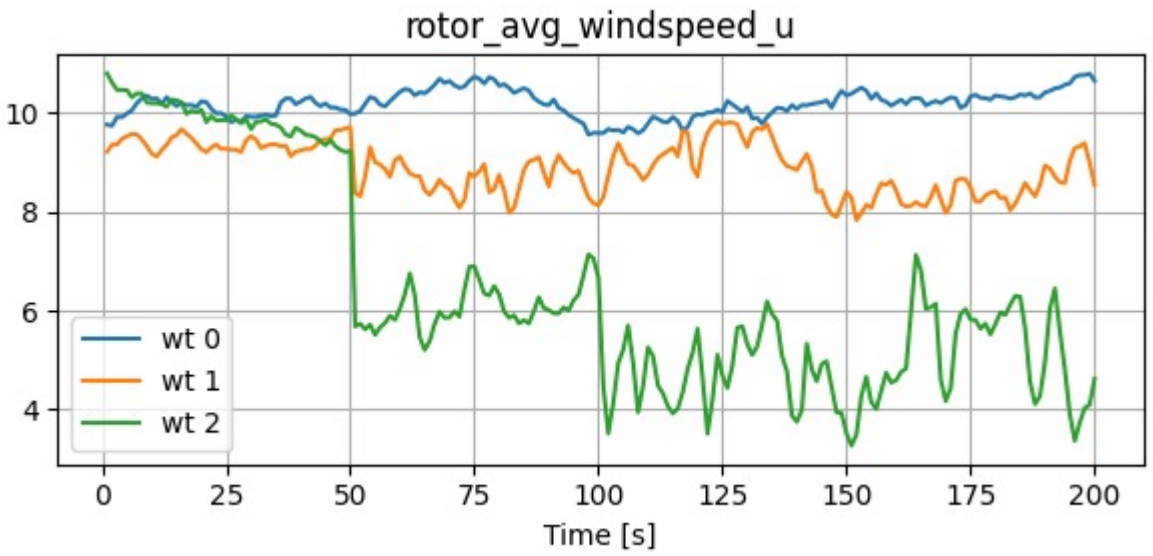

2.0
1.5
1.0
0.5
0.0
wt 0
wt 1
wt 2
0
25
50
75
100
125
150
175
200
Time [s]

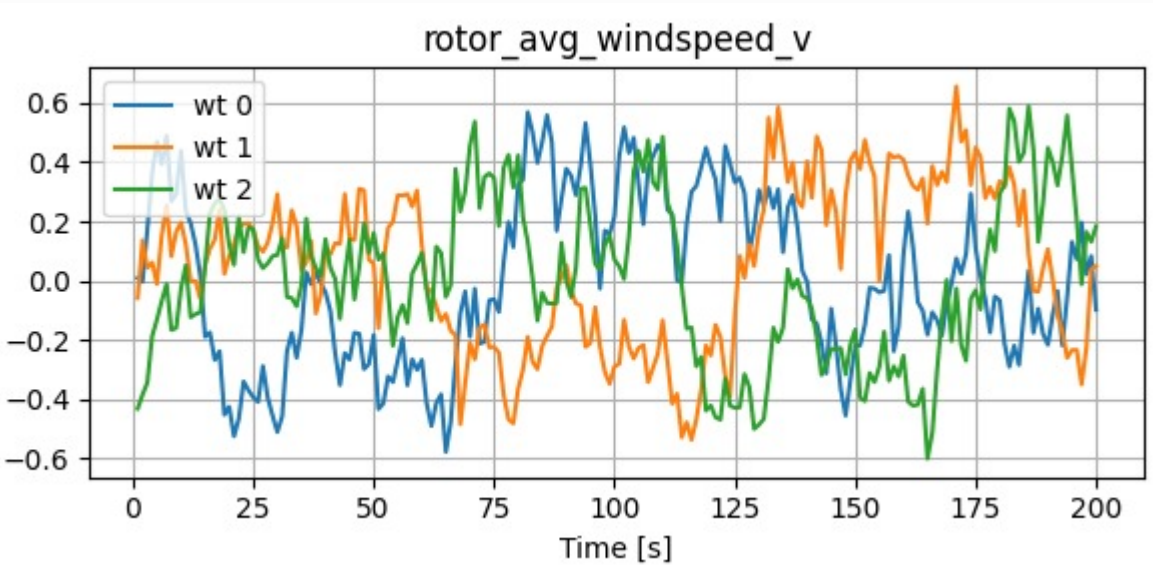

ct
0.8
0.7
0.6
0.5
0.4
0.3
0.2
wt 0
wt 1
wt 2
0
25
50
75
100
125
150
175
200
Time [s]

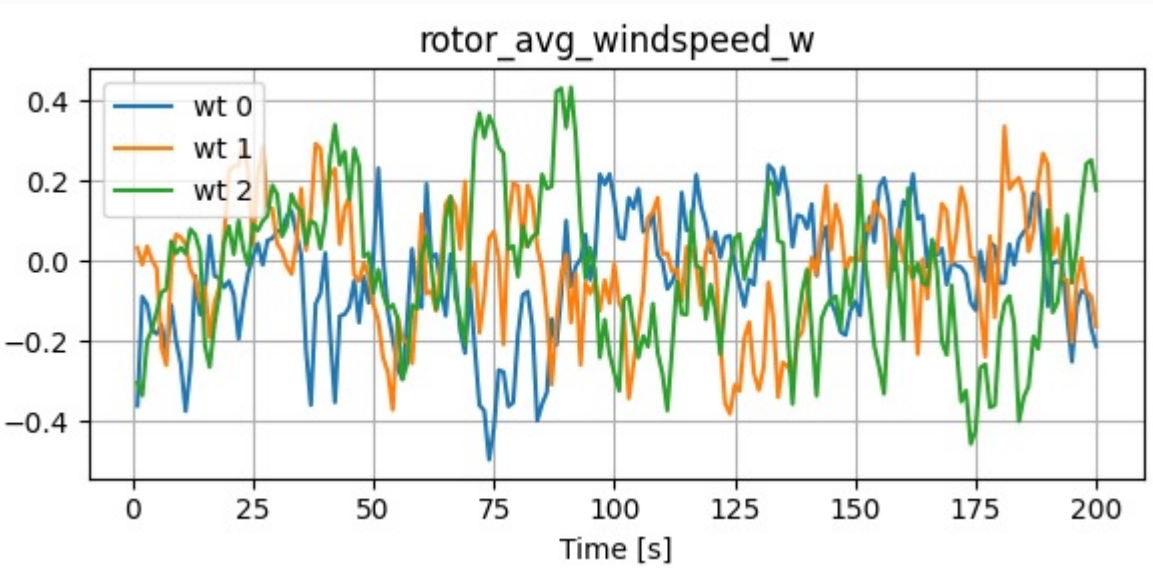

yaw
0.04
0.02
0.00
−0.02
−0.04
wt 0
wt 1
wt 2
0
25
50
75
100
125
150
175
200
Time [s]


tilt
0.04
0.02
0.00
−0.02
wt 0
wt 1
wt 2

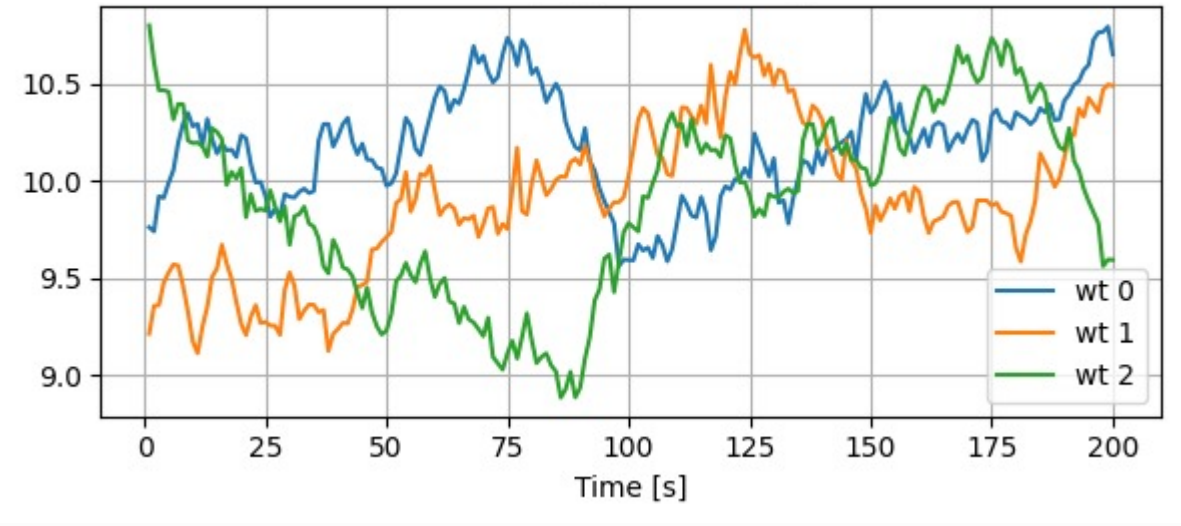


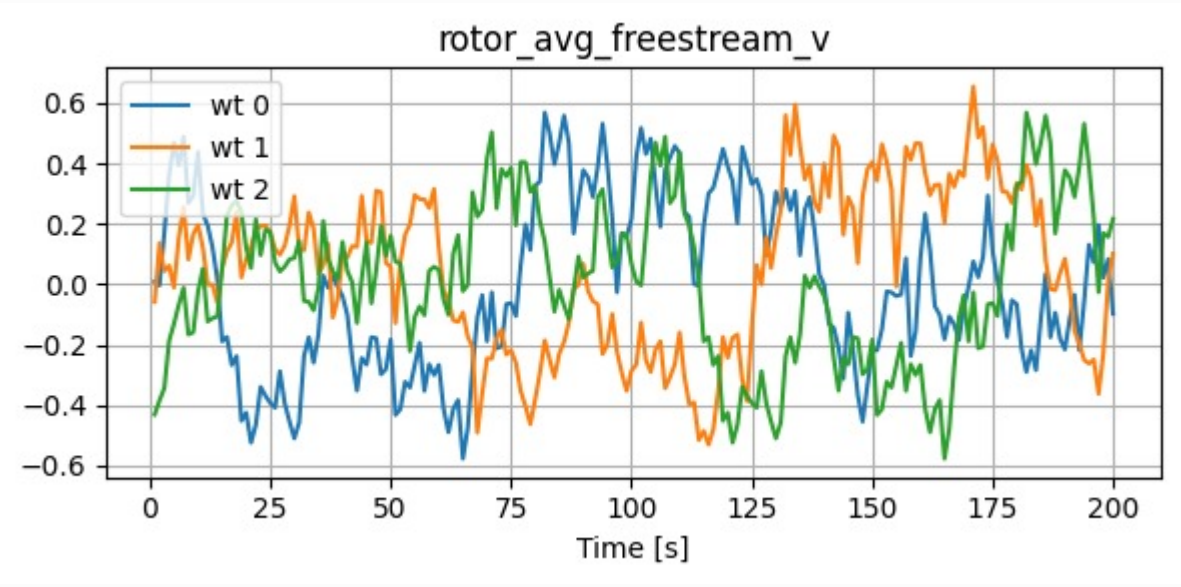


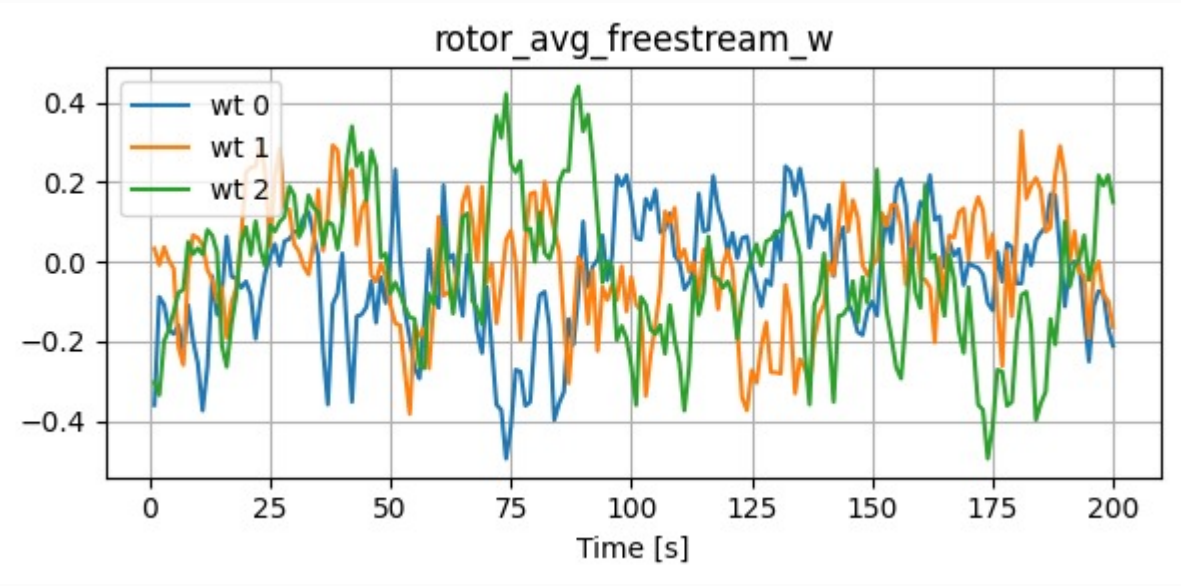




rotor_avg_freestream_u







HAWC2 wind turbine

The HAWC2 wind turbine provides a higher fidelity model that combines a multibody structural formulation with aerodynamic forces acting on the rotating blades. This model captures structural loads and handles wind speed variations over the rotor, but the computational costs are higher than the actuator disk models.

To use the HAWC2 wind turbines you need:

- `h2lib`: A library version of HAWC2. It can be installed by `pip install dynamiks[hawc2]`, see installation.
- A valid license
- HAWC2 models of the wind turbines

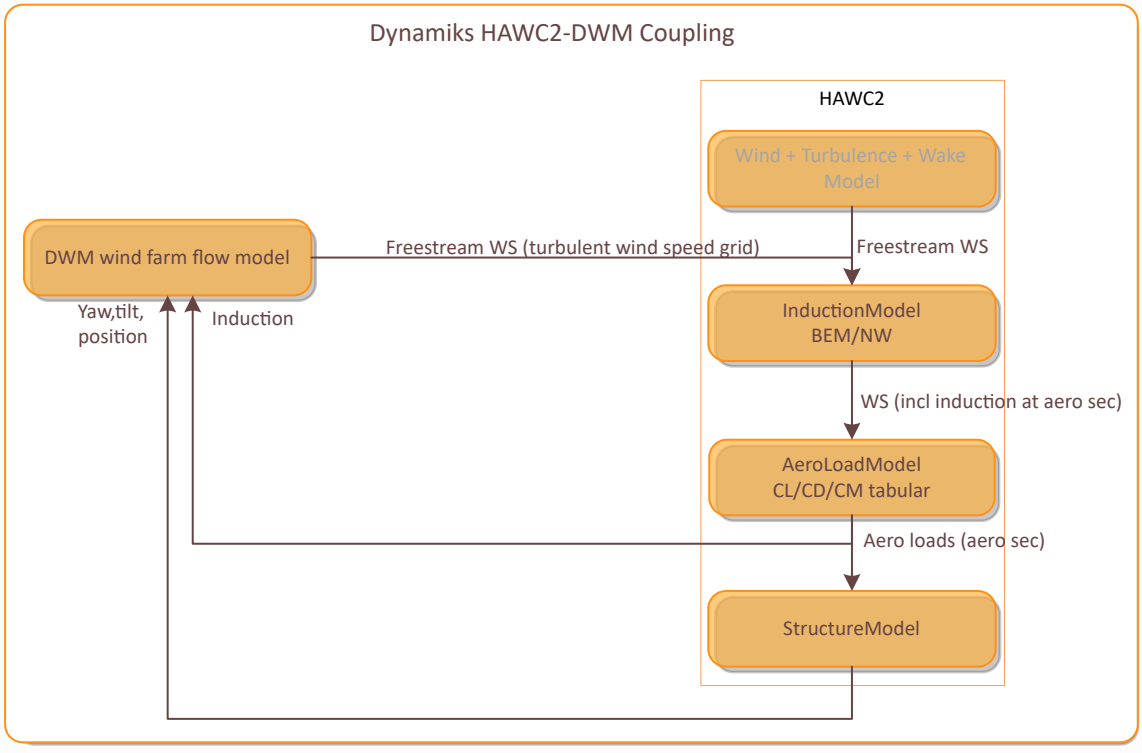
HAWC2-DWM (HAWC2Farm)

HAWC2 is capable of simulating one wind turbine and therefore multiple HAWC2 instances are needed to simulate a wind farm in Dynamiks.

When coupling HAWC2 with the DWM wind farm flow model, the wind speed module of HAWC2 is bypassed. Instead the turbulent wind speeds including mean wind, shear, gusts, wakes etc. are calculated by Dynamiks on a 3D grid covering an aera around each wind turbine and passed to the corresponding HAWC2 instance.

Each HAWC2 instance will use the provided wind speeds to calculate the aerodynamic forces on the blades, the induction and the structural response.

Finally Dynamiks receives the induction and structural states which is used by the DWM model to calculate the wakes.




```
[9]: from dynamiks.wind_turbines.hawc2_windturbine import HAWC2WindTurbines
```

```python
from h2lib_tests.test_files import tfp as h2lib_tfp
htc_filename_lst = [h2lib_tfp + "DTU_10_MW/htc/DTU_10MW_RWT.htc",
                    h2lib_tfp + "IEA-15-240-RWT-Onshore/htc/IEA_15MW_RWT_Onshore_simple.htc"]
wts = HAWC2WindTurbines(x=[0,500,1000], y=[0,0,0],
                        htc_filename_lst=htc_filename_lst,
                        types=[0, 1, 0],
                        case_name='MyTestCase', # subfolder name in the htc, res and log folder
                        suppress_output=False # don't show hawc2 output in console
                       )
```

```
License verified - OK
 Opening main command file:
 /usr/local/lib/python3.11/site-packages/h2lib_tests/test_files/DTU_10_MW/htc/My
 TestCase/DTU_10MW_RWT_wt2.htc
 Current directory is
 /usr/local/lib/python3.11/site-packages/h2lib_tests/test_files/DTU_10_MW
 Continue on no convergence = true
 Newmark commands read with succes
 Simulation commands read with succes
 Reading data of main body : tower
License verified - OK
 Opening main command file:
 /usr/local/lib/python3.11/site-packages/h2lib_tests/test_files/IEA-15-240-RWT-O
 nshore/htc/MyTestCase/IEA_15MW_RWT_Onshore_simple_wt1.htc
 Current directory is
 /usr/local/lib/python3.11/site-packages/h2lib_tests/test_files/IEA-15-240-RWT-O
 nshore
 Succes opening ./data/DTU_10MW_RWT_Tower_st.dat
 Continue on no convergence = true
License verified - OK
 Opening main command file:
 /usr/local/lib/python3.11/site-packages/h2lib_tests/test_files/DTU_10_MW/htc/My
 TestCase/DTU_10MW_RWT_wt0.htc
 Current directory is
 /usr/local/lib/python3.11/site-packages/h2lib_tests/test_files/DTU_10_MW
 Newmark commands read with succes
 Simulation commands read with succes
 Continue on no convergence = true
 Newmark commands read with succes
 Simulation commands read with succes
 Reading data of main body : tower
 Reading data of main body : tower
 Succes opening ./data/IEA_15MW_RWT_Tower_st.dat
 timoschenko input commands read with succes
 Succes opening ./data/DTU_10MW_RWT_Tower_st.dat
 topologi_c2def_inputs read with succes
 Topologi main body tower commands read with succes
 Reading data of main body : towertop
 Succes opening ./data/DTU_10MW_RWT_Towertop_st.dat
 timoschenko input commands read with succes
 timoschenko input commands read with succes
 topologi_c2def_inputs read with succes
 Topologi main body tower commands read with succes
 Reading data of main body : towertop
 topologi_c2def_inputs read with succes
 Topologi main body tower commands read with succes
 Reading data of main body : towertop
 Succes opening ../IEA-15-240-RWT/IEA_15MW_RWT_Dummy_st.dat
 Succes opening ./data/DTU_10MW_RWT_Towertop_st.dat
 timoschenko input commands read with succes
 topologi_c2def_inputs read with succes
 Topologi main body towertop commands read with succes
 Reading data of main body : shaft
 timoschenko input commands read with succes
 topologi_c2def_inputs read with succes
 Topologi main body towertop commands read with succes
 Succes opening ./data/DTU_10MW_RWT_Shaft_st.dat
 Reading data of main body : connector
 Succes opening ../IEA-15-240-RWT/IEA_15MW_RWT_Dummy_st.dat
 timoschenko input commands read with succes
```

```
timoschenko input commands read with succes
topologi_c2def_inputs read with succes
Topologi main body towertop commands read with succes
Reading data of main body : shaft
topologi_c2def_inputs read with succes
timoschenko input commands read with succes
Topologi main body shaft commands read with succes
Reading data of main body : hub1
topologi_c2def_inputs read with succes
Topologi main body connector commands read with succes
Reading data of main body : shaft
Succes opening ./data/DTU_10MW_RWT_Hub_st.dat
Succes opening ./data/DTU_10MW_RWT_Shaft_st.dat
Succes opening ../IEA-15-240-RWT/IEA_15MW_RWT_Shaft_st.dat
timoschenko input commands read with succes
topologi_c2def_inputs read with succes
Topologi main body shaft commands read with succes
timoschenko input commands read with succes
Reading data of main body : hub1
topologi_c2def_inputs read with succes
Topologi main body shaft commands read with succes
Reading data of main body : hub1
Succes opening ./data/DTU_10MW_RWT_Hub_st.dat
Succes opening ../IEA-15-240-RWT/IEA_15MW_RWT_Dummy_st.dat
timoschenko input commands read with succes
topologi_c2def_inputs read with succes
Topologi main body hub1 commands read with succes
Reading data of main body : hub2
Topologi main body hub2 commands read with succes
Reading data of main body : hub3
Topologi main body hub3 commands read with succes
Reading data of main body : blade1
timoschenko input commands read with succes
topologi_c2def_inputs read with succes
Topologi main body hub1 commands read with succes
Succes opening ./data/DTU_10MW_RWT_Blade_st.dat
Reading data of main body : hub2
Topologi main body hub2 commands read with succes
Reading data of main body : hub3
Topologi main body hub3 commands read with succes
Reading data of main body : blade1
Succes opening ../IEA-15-240-RWT/IEA_15MW_RWT_Blade_st_noFPM.st
timoschenko input commands read with succes
topologi_c2def_inputs read with succes
Topologi main body hub1 commands read with succes
Reading data of main body : hub2
Topologi main body hub2 commands read with succes
Reading data of main body : hub3
Topologi main body hub3 commands read with succes
Reading data of main body : blade1
Succes opening ./data/DTU_10MW_RWT_Blade_st.dat
timoschenko input commands read with succes
timoschenko input commands read with succes
topologi_c2def_inputs read with succes
Topologi main body blade1 commands read with succes
Reading data of main body : blade2
Topologi main body blade2 commands read with succes
topologi_c2def_inputs read with succes
Reading data of main body : blade3
Topologi main body blade1 commands read with succes
Topologi main body blade3 commands read with succes
timoschenko input commands read with succes
Reading data of main body : blade2
Topologi main body blade2 commands read with succes
Reading data of main body : blade3
Topologi main body blade3 commands read with succes
Base orientation input commands read with succes
Base orientation input commands read with succes
relative orientation input commands read with succes
relative orientation input commands read with succes
relative orientation input commands read with succes
topologi_c2def_inputs read with succes
```

```
 Topologi main body blade1 commands read with succes
 Reading data of main body : blade2
 Topologi main body blade2 commands read with succes
 Reading data of main body : blade3
 Topologi main body blade3 commands read with succes
 relative orientation input commands read with succes
 Base orientation input commands read with succes
 relative orientation input commands read with succes
 relative orientation input commands read with succes
 relative orientation input commands read with succes
 relative orientation input commands read with succes
 relative orientation input commands read with succes
 relative orientation input commands read with succes
 relative orientation input commands read with succes
 relative orientation input commands read with succes
 relative orientation input commands read with succes
 relative orientation input commands read with succes
 Orientation input commands read with succes
 relative orientation input commands read with succes
 Fix0 constraint input commands read with succes
 relative orientation input commands read with succes
 Fix1 constraint input commands read with succes
 Bearing1 constraint input commands read with succes
 Fix1 constraint input commands read with succes
 Fix1 constraint input commands read with succes
 Fix1 constraint input commands read with succes
 bearing2 constraint input commands read with succes
 relative orientation input commands read with succes
 bearing2 constraint input commands read with succes
 relative orientation input commands read with succes
 bearing2 constraint input commands read with succes
 constraint input commands read with succes
 Topologi commands read with succes
 relative orientation input commands read with succes
 relative orientation input commands read with succes
In mann module: Dont scale applied - turbulence scalings are bypassed
 relative orientation input commands read with succes
 relative orientation input commands read with succes
*** WARNING *** The default advection direction of turbulence has been reversed in HAWC2 versio
 relative orientation input commands read with succes
The default advection direction complies with the Mann turbulence model, and turbulence boxes g
See the release notes for HAWC2 version 13.1 and the manual for more details.
Mann commands read with succes
 relative orientation input commands read with succes
 Orientation input commands read with succes
 Tower shadow (potential2 flow) commands read with succes
 Wind commands read with succes
 Fix0 constraint input commands read with succes
 aerodrag element commands read with succes
 Fix1 constraint input commands read with succes
 Bearing1 constraint input commands read with succes
 aerodrag element commands read with succes
 Fix1 constraint input commands read with succes
 Aerodrag commands read with succes
 Fix1 constraint input commands read with succes
 Fix1 constraint input commands read with succes
 relative orientation input commands read with succes
 bearing2 constraint input commands read with succes
 Aerodynamic commands read with succes
 Orientation input commands read with succes
 bearing2 constraint input commands read with succes
 bearing2 constraint input commands read with succes
 constraint input commands read with succes
 Topologi commands read with succes
 Fix0 constraint input commands read with succes
 Fix1 constraint input commands read with succes
 Fix1 constraint input commands read with succes
 Bearing1 constraint input commands read with succes
In mann module: Dont scale applied - turbulence scalings are bypassed
*** WARNING *** The default advection direction of turbulence has been reversed in HAWC2 versio
 Fix1 constraint input commands read with succes
The default advection direction complies with the Mann turbulence model, and turbulence boxes g
```

```
See the release notes for HAWC2 version 13.1 and the manual for more details.
Mann commands read with succes
 Fix1 constraint input commands read with succes
 Tower shadow (potential2 flow) commands read with succes
 Fix1 constraint input commands read with succes
 Wind commands read with succes
 aerodrag element commands read with succes
 aerodrag element commands read with succes
 Aerodrag commands read with succes
 bearing2 constraint input commands read with succes
 Aerodynamic commands read with succes
 bearing2 constraint input commands read with succes
 bearing2 constraint input commands read with succes
 constraint input commands read with succes
 Topologi commands read with succes
 output commands read with succes
 Output commands read
 Dll type2 input commands read with succes
 Wind commands read with succes
 output commands read with succes
 Output commands read
 Actions commands read
 Dll type2 input commands read with succes
 output commands read with succes
 Output commands read
 aerodrag element commands read with succes
 Actions commands read
 Dll type2 input commands read with succes
 aerodrag element commands read with succes
 Aerodrag commands read with succes
 output commands read with succes
 Output commands read
 Actions commands read
 Dll type2 input commands read with succes
 output commands read with succes
 Output commands read
 Dll type2 input commands read with succes
 output commands read with succes
 Output commands read
 Dll type2 input commands read with succes
 DLL commands read with succes
 Aerodynamic commands read with succes
 output commands read with succes
 Output commands read
 Actions commands read
 Dll type2 input commands read with succes
 output commands read with succes
 Output commands read
 Actions commands read
 Dll type2 input commands read with succes
 output commands read with succes
 Output commands read
 Actions commands read
 Dll type2 input commands read with succes
 output commands read with succes
 Output commands read
 Dll type2 input commands read with succes
 DLL commands read with succes
 output commands read with succes
 Output commands read
 Dll type2 input commands read with succes
 output commands read with succes
 Output commands read
 output commands read with succes
 Output commands read
 Actions commands read
 Dll type2 input commands read with succes
 output commands read with succes
 Output commands read
 Actions commands read
 Dll type2 input commands read with succes
 output commands read with succes
```

```
 Output commands read
 Actions commands read
 Dll type2 input commands read with succes
 output commands read with succes
 Output commands read
 output commands read with succes
 Output commands read
 Dll type2 input commands read with succes
 DLL commands read with succes
 output commands read with succes
 Output commands read
 Initialization of structure
 Initialization of structure
 Initialization of structure
```

```
*** WARNING *** The default advection direction of turbulence has been reversed in HAWC2 versio
*** WARNING *** The default advection direction of turbulence has been reversed in HAWC2 versio
```

```
 Initializing of aero rotor...
 Initialization of rotor aerodynamics
 Succes opening ./data/DTU_10MW_RWT_ae.dat
 Succes opening ./data/DTU_10MW_RWT_pc.dat
 Initializing of aero rotor...
 Initialization of rotor aerodynamics
 Succes opening ./data/DTU_10MW_RWT_ae.dat
 Succes opening ./data/DTU_10MW_RWT_pc.dat
 Initializing of aero rotor...
 Initialization of rotor aerodynamics
 Succes opening ../IEA-15-240-RWT/IEA_15MW_RWT_ae.dat
 Succes opening ../IEA-15-240-RWT/IEA_15MW_RWT_pc_OpenFASTpolars_3dcorr.dat
 Initialization of rotor induction
 Initialization of wind
 Initialization of external type2 DLL
 External DLL ./control/dtu_we_controller.dll is attempted to open
DLL loaded with success /usr/local/lib/python3.11/site-packages/h2lib_tests/test_files/DTU_10_M
Using /usr/local/lib/python3.11/site-packages/h2lib_tests/test_files/DTU_10_MW/./control/dtu_we
 DLL subroutine init init_regulation_advanced is called
 In initialization call of ./control/dtu_we_controller.dll Output is
  0.000000000000000E+000
*** WARNING *** Symbol not found in dll: message
 *** INFO *** The DLL subroutine message could not be loaded - bypassed!
 Initialization of external type2 DLL
 External DLL ./control/generator_servo.dll is attempted to open
DLL loaded with success /usr/local/lib/python3.11/site-packages/h2lib_tests/test_files/DTU_10_M
Using /usr/local/lib/python3.11/site-packages/h2lib_tests/test_files/DTU_10_MW/./control/genera
 DLL subroutine init init_generator_servo is called
 In initialization call of ./control/generator_servo.dll Output is
  0.000000000000000E+000
*** WARNING *** Symbol not found in dll: message
 *** INFO *** The DLL subroutine message could not be loaded - bypassed!
 Initialization of external type2 DLL
 External DLL ./control/mech_brake.dll is attempted to open
DLL loaded with success /usr/local/lib/python3.11/site-packages/h2lib_tests/test_files/DTU_10_M
Using /usr/local/lib/python3.11/site-packages/h2lib_tests/test_files/DTU_10_MW/./control/mech_b
 DLL subroutine init init_mech_brake is called
 In initialization call of ./control/mech_brake.dll Output is
  0.000000000000000E+000
*** WARNING *** Symbol not found in dll: message
 *** INFO *** The DLL subroutine message could not be loaded - bypassed!
 Initialization of external type2 DLL
 External DLL ./control/servo_with_limits.dll is attempted to open
DLL loaded with success /usr/local/lib/python3.11/site-packages/h2lib_tests/test_files/DTU_10_M
Using /usr/local/lib/python3.11/site-packages/h2lib_tests/test_files/DTU_10_MW/./control/servo_
 DLL subroutine init init_servo_with_limits is called
 In initialization call of ./control/servo_with_limits.dll Output is
  0.000000000000000E+000
*** WARNING *** Symbol not found in dll: message
 *** INFO *** The DLL subroutine message could not be loaded - bypassed!
 Initialization of external type2 DLL
 External DLL ./control/towerclearance_mblade.dll is attempted to open
DLL loaded with success /usr/local/lib/python3.11/site-packages/h2lib_tests/test_files/DTU_10_M
```

```
Using /usr/local/lib/python3.11/site-packages/h2lib_tests/test_files/DTU_10_MW/./control/towerc
 DLL subroutine init initialize is called
 In initialization call of ./control/towerclearance_mblade.dll Output is
  0.000000000000000E+000
*** WARNING *** Symbol not found in dll: message
 *** INFO *** The DLL subroutine message could not be loaded - bypassed!
 Creating link between structure and aerodynamics
 Initialization of rotor induction
 Initialization of wind
 Initialization of external type2 DLL
 External DLL ./control/dtu_we_controller.dll is attempted to open
DLL loaded with success /usr/local/lib/python3.11/site-packages/h2lib_tests/test_files/DTU_10_M
Using /usr/local/lib/python3.11/site-packages/h2lib_tests/test_files/DTU_10_MW/./control/dtu_we
 DLL subroutine init init_regulation_advanced is called
 In initialization call of ./control/dtu_we_controller.dll Output is
  0.000000000000000E+000
*** WARNING *** Symbol not found in dll: message
 *** INFO *** The DLL subroutine message could not be loaded - bypassed!
 Initialization of external type2 DLL
 External DLL ./control/generator_servo.dll is attempted to open
DLL loaded with success /usr/local/lib/python3.11/site-packages/h2lib_tests/test_files/DTU_10_M
Using /usr/local/lib/python3.11/site-packages/h2lib_tests/test_files/DTU_10_MW/./control/genera
 DLL subroutine init init_generator_servo is called
 In initialization call of ./control/generator_servo.dll Output is
  0.000000000000000E+000
*** WARNING *** Symbol not found in dll: message
 *** INFO *** The DLL subroutine message could not be loaded - bypassed!
 Initialization of external type2 DLL
 External DLL ./control/mech_brake.dll is attempted to open
 Creating link between structure and aerodrag
 Initialization of Aerodrag
DLL loaded with success /usr/local/lib/python3.11/site-packages/h2lib_tests/test_files/DTU_10_M
Using /usr/local/lib/python3.11/site-packages/h2lib_tests/test_files/DTU_10_MW/./control/mech_b
 DLL subroutine init init_mech_brake is called
 In initialization call of ./control/mech_brake.dll Output is
  0.000000000000000E+000
*** WARNING *** Symbol not found in dll: message
 *** INFO *** The DLL subroutine message could not be loaded - bypassed!
 Initialization of external type2 DLL
 External DLL ./control/servo_with_limits.dll is attempted to open
DLL loaded with success /usr/local/lib/python3.11/site-packages/h2lib_tests/test_files/DTU_10_M
Using /usr/local/lib/python3.11/site-packages/h2lib_tests/test_files/DTU_10_MW/./control/servo_
 DLL subroutine init init_servo_with_limits is called
 In initialization call of ./control/servo_with_limits.dll Output is
  0.000000000000000E+000
*** WARNING *** Symbol not found in dll: message
 *** INFO *** The DLL subroutine message could not be loaded - bypassed!
 Initialization of external type2 DLL
 External DLL ./control/towerclearance_mblade.dll is attempted to open
DLL loaded with success /usr/local/lib/python3.11/site-packages/h2lib_tests/test_files/DTU_10_M
Using /usr/local/lib/python3.11/site-packages/h2lib_tests/test_files/DTU_10_MW/./control/towerc
 DLL subroutine init initialize is called
 In initialization call of ./control/towerclearance_mblade.dll Output is
  0.000000000000000E+000
*** WARNING *** Symbol not found in dll: message
 *** INFO *** The DLL subroutine message could not be loaded - bypassed!
 Creating link between structure and aerodynamics
 Creating link between structure and aerodrag
 Initialization of Aerodrag
```

```
 ***********************************************************************
 *  Build information for dtu_we_controller.dll
 *  DTU Wind Energy Controller - main control dll
 *  Intel, version         2021 ,     20201112
 *  Linux
 ***********************************************************************
 *  GIT-TAG        = 0.5-2-g885d71c
 *  GIT-BRANCH     =
 *  BUILD_TYPE     = linux release
 *  BUILDER        = gitlab-runner
 *  COMPUTER_NAME  = runner-6s-xzdse-project-966-concurrent-0
```

```
 *  BUILD_DATE     = 2021-10-01
 **********************************************************************
 Controller dll initialization is successed!!
*** WARNING *** Symbol not found in dll: message
*** WARNING *** Symbol not found in dll: message
*** WARNING *** Symbol not found in dll: message
 Pitch Servo 1.2 loaded...
*** WARNING *** Symbol not found in dll: message
 Tower clearance DLL (mblade, ver. 1.0) loaded...
*** WARNING *** Symbol not found in dll: message
 **********************************************************************
 *  Build information for dtu_we_controller.dll
 *  DTU Wind Energy Controller - main control dll
 *  Intel, version         2021 ,     20201112
 *  Linux
 **********************************************************************
 *  GIT-TAG        = 0.5-2-g885d71c
 *  GIT-BRANCH     =
 *  BUILD_TYPE     = linux release
 *  BUILDER        = gitlab-runner
 *  COMPUTER_NAME  = runner-6s-xzdse-project-966-concurrent-0
 *  BUILD_DATE     = 2021-10-01
 **********************************************************************
 Controller dll initialization is successed!!
*** WARNING *** Symbol not found in dll: message
*** WARNING *** Symbol not found in dll: message
*** WARNING *** Symbol not found in dll: message
 Pitch Servo 1.2 loaded...
*** WARNING *** Symbol not found in dll: message
 Tower clearance DLL (mblade, ver. 1.0) loaded...
*** WARNING *** Symbol not found in dll: message
 **********************************************************************
 *  Build information for dtu_we_controller.dll
 *  DTU Wind Energy Controller - main control dll
 *  Intel, version         2021 ,     20201112
 *  Linux
 **********************************************************************
 *  GIT-TAG        = 0.5-2-g885d71c
 *  GIT-BRANCH     =
 *  BUILD_TYPE     = linux release
 *  BUILDER        = gitlab-runner
 *  COMPUTER_NAME  = runner-6s-xzdse-project-966-concurrent-0
 *  BUILD_DATE     = 2021-10-01
 **********************************************************************
 Controller dll initialization is successed!!
*** WARNING *** Symbol not found in dll: message
*** WARNING *** Symbol not found in dll: message
*** WARNING *** Symbol not found in dll: message
 Pitch Servo 1.2 loaded...
*** WARNING *** Symbol not found in dll: message
 Tower clearance DLL (mblade, ver. 1.0) loaded...
*** WARNING *** Symbol not found in dll: message
 Initialization of rotor induction
 Initialization of wind
 Initialization of external type2 DLL
 External DLL ./control/dtu_we_controller.dll is attempted to open
DLL loaded with success /usr/local/lib/python3.11/site-packages/h2lib_tests/test_files/IEA-15-2
Using /usr/local/lib/python3.11/site-packages/h2lib_tests/test_files/IEA-15-240-RWT-Onshore/./c
 DLL subroutine init init_regulation_advanced is called
 In initialization call of ./control/dtu_we_controller.dll Output is
  0.000000000000000E+000
*** WARNING *** Symbol not found in dll: message
 *** INFO *** The DLL subroutine message could not be loaded - bypassed!
 Initialization of external type2 DLL
 External DLL ./control/generator_servo.dll is attempted to open
DLL loaded with success /usr/local/lib/python3.11/site-packages/h2lib_tests/test_files/IEA-15-2
Using /usr/local/lib/python3.11/site-packages/h2lib_tests/test_files/IEA-15-240-RWT-Onshore/./c
 DLL subroutine init init_generator_servo is called
 In initialization call of ./control/generator_servo.dll Output is
  0.000000000000000E+000
*** WARNING *** Symbol not found in dll: message
 *** INFO *** The DLL subroutine message could not be loaded - bypassed!
```

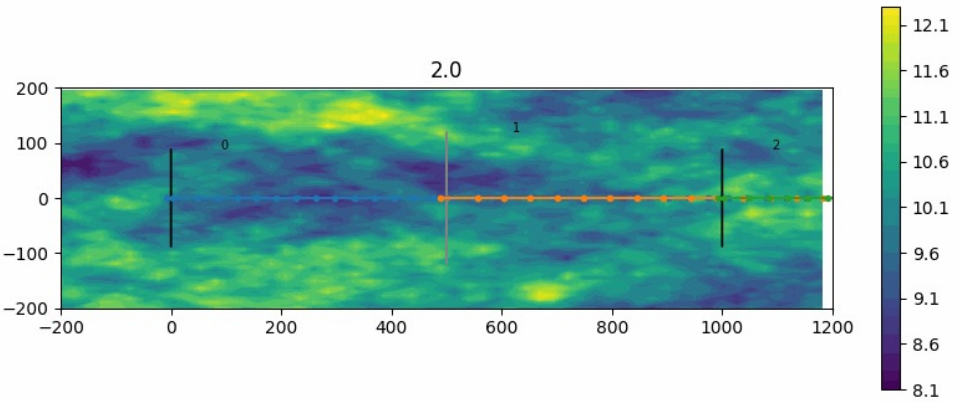
```
 Initialization of external type2 DLL
 External DLL ./control/mech_brake.dll is attempted to open
DLL loaded with success /usr/local/lib/python3.11/site-packages/h2lib_tests/test_files/IEA-15-2
Using /usr/local/lib/python3.11/site-packages/h2lib_tests/test_files/IEA-15-240-RWT-Onshore/./c
 DLL subroutine init init_mech_brake is called
 In initialization call of ./control/mech_brake.dll Output is
  0.000000000000000E+000
*** WARNING *** Symbol not found in dll: message
 *** INFO *** The DLL subroutine message could not be loaded - bypassed!
 Initialization of external type2 DLL
 External DLL ./control/servo_with_limits.dll is attempted to open
DLL loaded with success /usr/local/lib/python3.11/site-packages/h2lib_tests/test_files/IEA-15-2
Using /usr/local/lib/python3.11/site-packages/h2lib_tests/test_files/IEA-15-240-RWT-Onshore/./c
 DLL subroutine init init_servo_with_limits is called
 In initialization call of ./control/servo_with_limits.dll Output is
  0.000000000000000E+000
*** WARNING *** Symbol not found in dll: message
 *** INFO *** The DLL subroutine message could not be loaded - bypassed!
 Initialization of external type2 DLL
 External DLL ./control/towerclearance_mblade.dll is attempted to open
DLL loaded with success /usr/local/lib/python3.11/site-packages/h2lib_tests/test_files/IEA-15-2
Using /usr/local/lib/python3.11/site-packages/h2lib_tests/test_files/IEA-15-240-RWT-Onshore/./c
 DLL subroutine init initialize is called
 In initialization call of ./control/towerclearance_mblade.dll Output is
  0.000000000000000E+000
*** WARNING *** Symbol not found in dll: message
 *** INFO *** The DLL subroutine message could not be loaded - bypassed!
 Creating link between structure and aerodynamics
 Creating link between structure and aerodrag
 Initialization of Aerodrag
```

[10]:
```
from dynamiks.dwm.dwm_flow_simulation import DWMFlowSimulation
from dynamiks.dwm.particle_deficit_profiles.ainslie import jDWMAinslieGenerator
from dynamiks.views import XYView
fs = DWMFlowSimulation(site=site, windTurbines=wts, particleDeficitGenerator=jDWMAinslieGenerat
```

[11]:
```
# Add rotor speed and azimuth sensor
wts.add_sensor("rotor", "constraint bearing1 shaft_rot 2", ext_lst=['azi','speed']);
```

[12]:
```
ax = plt.figure(figsize=(10,4)).gca()
view = XYView(z=70, x=np.linspace(-200,1200), y=np.linspace(-200,200), ax=ax)
fs.visualize(fs.time+100, id='WindTurbines_H2', view=view)
```

[12]:



Output from HAWC2

Output form the HAWC2 instances can be obtained during the simulation by Dynamiks (with

the Dynamiks time resolution) or from the HAWC2 result file (with the HAWC2 time resolution).

During simulation by Dynamiks

Output from HAWC2 can be obtained during simulation by Dynamiks. To use this option, the needed sensors must be added in advance, see code cell above where a rotor speed and azimuth sensor are added.

The HAWC2WindTurbine extends the `add_sensor` method, such that

- the `getter` argument can be a HAWC2 sensor string
- the `setter` argument can be a int, which is interpreted as the index of a `general variable` sensor to set. In this way, e.g. yaw or derating setpoints can be set

Note, that some sensors, e.g. `aero_power`, `aero_thrust` and rotor center wind speed are predefined.

[13]:
```
help(wts.add_sensor)
```

```
Help on method add_sensor in module dynamiks.wind_turbines.hawc2_windturbine:

add_sensor(name, getter=None, setter=None, expose=False, ext_lst=None) method of dynamiks.wind_
    add a wind turbine sensor

    Sensor values available using: windTurbines.sensors.<name>

    Parameters
    ----------
    name : str
        Name of sensor. Cannot contain spaces
    getter : str, function
        if str: HAWC2 sensor string, e.g. "constraint bearing1 shaft_rot 2"
        if function: f(wt) -> sensor_value
    setter : int, function or None, optional
        if int, index of the HAWC2 'general variable' sensor to set
        if function, f(wt, value) -> None
    expose : boolean, optional
        if True, the getter/setter is exposed to the wind turbine enabling e.g. wt.yaw as a sho
    ext_lst : list or None
        List of sensor name extensions in case the sensor returns more than one value,
        e.g. the u,v and w components of the wind. In this case "add_sensor('ws', ext_lst=['u',
        the three sensors: ws_u, ws_v, ws_w
```

[14]:
```
from dynamiks.utils.data_dumper import DataDumper

fs.run(fs.time+20, verbose=1) # run another 20s
```

```
100% |████████████████████| 20/20 [00:32<00:00,  1.72s/it]
```

```
Dynamic stall method:  2  used for entire rotor
Dynamic stall method:  2  used for entire rotor
   ... initialization of unified dynamic stall model.
   ... Unified dynamic stall model time constants
   ... Attached flow A1 =   0.165000, A2 =   0.335000, A3 =   0.000000
```

```
   ... Attached flow B1 =   0.045500, B2 =   0.300000, B3 =   0.300000
   ... Stalled flow taupre =   1.500000,  taubly =   6.000000
   ... initialization of unified dynamic stall model.
Dynamic stall method:  2  used for entire rotor
   ... initialization of unified dynamic stall model.
DS model deactivated: blade  1 at 0mm. Max f_sep=  0.00, dclda=  0.00, thickness=100.00. See ma
DS model deactivated: blade  1 at 120mm. Max f_sep=  0.00, dclda=  0.00, thickness= 99.95. See
   ... Unified dynamic stall model time constants
   ... Attached flow A1 =   0.165000, A2 =   0.335000, A3 =   0.000000
   ... Unified dynamic stall model time constants
   ... Attached flow B1 =   0.045500, B2 =   0.300000, B3 =   0.300000
   ... Stalled flow taupre =   1.500000,  taubly =   6.000000
DS model deactivated: blade  1 at 481mm. Max f_sep=  0.00, dclda=  0.00, thickness= 99.81. See
   ... Attached flow A1 =   0.165000, A2 =   0.335000, A3 =   0.000000
   ... Attached flow B1 =   0.045500, B2 =   0.300000, B3 =   0.300000
   ... Stalled flow taupre =   1.500000,  taubly =   6.000000
DS model deactivated: blade  1 at 1080mm. Max f_sep=  0.00, dclda=  0.00, thickness= 99.57. See
DS model deactivated: blade  1 at 0mm. Max f_sep=  0.00, dclda=  0.00, thickness=100.00. See ma
DS model deactivated: blade  1 at 1916mm. Max f_sep=  0.00, dclda=  0.00, thickness= 99.23. See
DS model deactivated: blade  1 at 0mm. Max f_sep=  0.00, dclda=  0.00, thickness=100.00. See ma
DS model deactivated: blade  1 at 88mm. Max f_sep=  0.00, dclda=  0.00, thickness=100.00. See m
DS model deactivated: blade  1 at 2984mm. Max f_sep=  1.00, dclda=  0.05, thickness= 98.80. See
DS model deactivated: blade  1 at 88mm. Max f_sep=  0.00, dclda=  0.00, thickness=100.00. See m
DS model deactivated: blade  1 at 355mm. Max f_sep=  0.00, dclda=  0.00, thickness=100.00. See
DS model deactivated: blade  1 at 4281mm. Max f_sep=  1.00, dclda=  0.24, thickness= 97.85. See
DS model deactivated: blade  1 at 355mm. Max f_sep=  0.00, dclda=  0.00, thickness=100.00. See
DS model deactivated: blade  1 at 797mm. Max f_sep=  0.00, dclda=  0.00, thickness=100.00. See
DS model deactivated: blade  1 at 5802mm. Max f_sep=  1.00, dclda=  1.14, thickness= 93.02. See
DS model deactivated: blade  1 at 797mm. Max f_sep=  0.00, dclda=  0.00, thickness=100.00. See
DS model deactivated: blade  1 at 1414mm. Max f_sep=  0.00, dclda=  0.00, thickness=100.00. See
DS model deactivated: blade  1 at 7539mm. Max f_sep=  1.00, dclda=  2.15, thickness= 87.50. See
DS model deactivated: blade  1 at 1414mm. Max f_sep=  0.00, dclda=  0.00, thickness=100.00. See
DS model deactivated: blade  1 at 2203mm. Max f_sep=  4.24, dclda=  0.00, thickness= 99.77. See
DS model deactivated: blade  1 at 2203mm. Max f_sep=  4.24, dclda=  0.00, thickness= 99.77. See
DS model deactivated: blade  1 at 3160mm. Max f_sep=  4.24, dclda=  0.01, thickness= 98.69. See
DS model deactivated: blade  1 at 3160mm. Max f_sep=  4.24, dclda=  0.01, thickness= 98.69. See
DS model deactivated: blade  1 at 4282mm. Max f_sep=  4.24, dclda=  0.01, thickness= 97.41. See
DS model deactivated: blade  1 at 4282mm. Max f_sep=  4.24, dclda=  0.01, thickness= 97.41. See
DS model deactivated: blade  1 at 5565mm. Max f_sep=  4.24, dclda=  0.02, thickness= 94.83. See
DS model deactivated: blade  1 at 5565mm. Max f_sep=  4.24, dclda=  0.02, thickness= 94.83. See
DS model deactivated: blade  1 at 7002mm. Max f_sep=  4.24, dclda=  0.04, thickness= 90.40. See
DS model deactivated: blade  1 at 7002mm. Max f_sep=  4.24, dclda=  0.04, thickness= 90.40. See
DS model deactivated: blade  1 at 8588mm. Max f_sep=  4.24, dclda=  0.07, thickness= 84.54. See
DS model deactivated: blade  1 at 8588mm. Max f_sep=  4.24, dclda=  0.07, thickness= 84.54. See
DS model deactivated: blade  1 at 10317mm. Max f_sep=  4.24, dclda=  0.10, thickness= 77.35. Se
DS model deactivated: blade  1 at 10317mm. Max f_sep=  4.24, dclda=  0.10, thickness= 77.35. Se
DS model deactivated: blade  1 at 12181mm. Max f_sep=  4.24, dclda=  0.14, thickness= 69.22. Se
DS model deactivated: blade  1 at 12181mm. Max f_sep=  4.24, dclda=  0.14, thickness= 69.22. Se
DS model deactivated: blade  1 at 14172mm. Max f_sep=  4.24, dclda=  0.18, thickness= 60.80. Se
DS model deactivated: blade  1 at 14172mm. Max f_sep=  4.24, dclda=  0.18, thickness= 60.80. Se
DS model deactivated: blade  1 at 16283mm. Max f_sep=  1.12, dclda=  2.62, thickness= 53.05. Se
DS model deactivated: blade  1 at 16283mm. Max f_sep=  1.12, dclda=  2.62, thickness= 53.05. Se
DS model deactivated: blade  2 at 0mm. Max f_sep=  0.00, dclda=  0.00, thickness=100.00. See ma
DS model deactivated: blade  2 at 120mm. Max f_sep=  0.00, dclda=  0.00, thickness= 99.95. See
DS model deactivated: blade  2 at 481mm. Max f_sep=  0.00, dclda=  0.00, thickness= 99.81. See
DS model deactivated: blade  2 at 1080mm. Max f_sep=  0.00, dclda=  0.00, thickness= 99.57. See
DS model deactivated: blade  2 at 1916mm. Max f_sep=  0.00, dclda=  0.00, thickness= 99.23. See
DS model deactivated: blade  2 at 2984mm. Max f_sep=  1.00, dclda=  0.05, thickness= 98.80. See
DS model deactivated: blade  2 at 4281mm. Max f_sep=  1.00, dclda=  0.24, thickness= 97.85. See
DS model deactivated: blade  2 at 5802mm. Max f_sep=  1.00, dclda=  1.14, thickness= 93.02. See
DS model deactivated: blade  2 at 7539mm. Max f_sep=  1.00, dclda=  2.15, thickness= 87.50. See
DS model deactivated: blade  2 at 0mm. Max f_sep=  0.00, dclda=  0.00, thickness=100.00. See ma
DS model deactivated: blade  2 at 88mm. Max f_sep=  0.00, dclda=  0.00, thickness=100.00. See m
DS model deactivated: blade  2 at 0mm. Max f_sep=  0.00, dclda=  0.00, thickness=100.00. See ma
DS model deactivated: blade  2 at 355mm. Max f_sep=  0.00, dclda=  0.00, thickness=100.00. See
DS model deactivated: blade  2 at 88mm. Max f_sep=  0.00, dclda=  0.00, thickness=100.00. See m
DS model deactivated: blade  2 at 797mm. Max f_sep=  0.00, dclda=  0.00, thickness=100.00. See
DS model deactivated: blade  2 at 355mm. Max f_sep=  0.00, dclda=  0.00, thickness=100.00. See
DS model deactivated: blade  2 at 1414mm. Max f_sep=  0.00, dclda=  0.00, thickness=100.00. See
DS model deactivated: blade  2 at 797mm. Max f_sep=  0.00, dclda=  0.00, thickness=100.00. See
DS model deactivated: blade  2 at 1414mm. Max f_sep=  0.00, dclda=  0.00, thickness=100.00. See
DS model deactivated: blade  2 at 2203mm. Max f_sep=  4.24, dclda=  0.00, thickness= 99.77. See
```

```
DS model deactivated: blade  2 at 2203mm. Max f_sep=  4.24, dclda=  0.00, thickness= 99.77. See
DS model deactivated: blade  2 at 3160mm. Max f_sep=  4.24, dclda=  0.01, thickness= 98.69. See
DS model deactivated: blade  2 at 3160mm. Max f_sep=  4.24, dclda=  0.01, thickness= 98.69. See
DS model deactivated: blade  2 at 4282mm. Max f_sep=  4.24, dclda=  0.01, thickness= 97.41. See
DS model deactivated: blade  2 at 4282mm. Max f_sep=  4.24, dclda=  0.01, thickness= 97.41. See
DS model deactivated: blade  2 at 5565mm. Max f_sep=  4.24, dclda=  0.02, thickness= 94.83. See
DS model deactivated: blade  2 at 5565mm. Max f_sep=  4.24, dclda=  0.02, thickness= 94.83. See
DS model deactivated: blade  2 at 7002mm. Max f_sep=  4.24, dclda=  0.04, thickness= 90.40. See
DS model deactivated: blade  2 at 7002mm. Max f_sep=  4.24, dclda=  0.04, thickness= 90.40. See
DS model deactivated: blade  2 at 8588mm. Max f_sep=  4.24, dclda=  0.07, thickness= 84.54. See
DS model deactivated: blade  2 at 8588mm. Max f_sep=  4.24, dclda=  0.07, thickness= 84.54. See
DS model deactivated: blade  2 at 10317mm. Max f_sep=  4.24, dclda=  0.10, thickness= 77.35. Se
DS model deactivated: blade  2 at 10317mm. Max f_sep=  4.24, dclda=  0.10, thickness= 77.35. Se
DS model deactivated: blade  2 at 12181mm. Max f_sep=  4.24, dclda=  0.14, thickness= 69.22. Se
DS model deactivated: blade  2 at 12181mm. Max f_sep=  4.24, dclda=  0.14, thickness= 69.22. Se
DS model deactivated: blade  2 at 14172mm. Max f_sep=  4.24, dclda=  0.18, thickness= 60.80. Se
DS model deactivated: blade  2 at 14172mm. Max f_sep=  4.24, dclda=  0.18, thickness= 60.80. Se
DS model deactivated: blade  2 at 16283mm. Max f_sep=  1.12, dclda=  2.62, thickness= 53.05. Se
DS model deactivated: blade  2 at 16283mm. Max f_sep=  1.12, dclda=  2.62, thickness= 53.05. Se
DS model deactivated: blade  3 at 0mm. Max f_sep=  0.00, dclda=  0.00, thickness=100.00. See ma
DS model deactivated: blade  3 at 120mm. Max f_sep=  0.00, dclda=  0.00, thickness= 99.95. See
DS model deactivated: blade  3 at 481mm. Max f_sep=  0.00, dclda=  0.00, thickness= 99.81. See
DS model deactivated: blade  3 at 1080mm. Max f_sep=  0.00, dclda=  0.00, thickness= 99.57. See
DS model deactivated: blade  3 at 1916mm. Max f_sep=  0.00, dclda=  0.00, thickness= 99.23. See
DS model deactivated: blade  3 at 2984mm. Max f_sep=  1.00, dclda=  0.05, thickness= 98.80. See
DS model deactivated: blade  3 at 4281mm. Max f_sep=  1.00, dclda=  0.24, thickness= 97.85. See
DS model deactivated: blade  3 at 5802mm. Max f_sep=  1.00, dclda=  1.14, thickness= 93.02. See
DS model deactivated: blade  3 at 7539mm. Max f_sep=  1.00, dclda=  2.15, thickness= 87.50. See
DS model deactivated: blade  3 at 0mm. Max f_sep=  0.00, dclda=  0.00, thickness=100.00. See ma
DS model deactivated: blade  3 at 88mm. Max f_sep=  0.00, dclda=  0.00, thickness=100.00. See m
DS model deactivated: blade  3 at 355mm. Max f_sep=  0.00, dclda=  0.00, thickness=100.00. See
DS model deactivated: blade  3 at 0mm. Max f_sep=  0.00, dclda=  0.00, thickness=100.00. See ma
DS model deactivated: blade  3 at 797mm. Max f_sep=  0.00, dclda=  0.00, thickness=100.00. See
DS model deactivated: blade  3 at 88mm. Max f_sep=  0.00, dclda=  0.00, thickness=100.00. See m
DS model deactivated: blade  3 at 1414mm. Max f_sep=  0.00, dclda=  0.00, thickness=100.00. See
DS model deactivated: blade  3 at 355mm. Max f_sep=  0.00, dclda=  0.00, thickness=100.00. See
DS model deactivated: blade  3 at 797mm. Max f_sep=  0.00, dclda=  0.00, thickness=100.00. See
DS model deactivated: blade  3 at 2203mm. Max f_sep=  4.24, dclda=  0.00, thickness= 99.77. See
DS model deactivated: blade  3 at 1414mm. Max f_sep=  0.00, dclda=  0.00, thickness=100.00. See
DS model deactivated: blade  3 at 3160mm. Max f_sep=  4.24, dclda=  0.01, thickness= 98.69. See
DS model deactivated: blade  3 at 2203mm. Max f_sep=  4.24, dclda=  0.00, thickness= 99.77. See
DS model deactivated: blade  3 at 4282mm. Max f_sep=  4.24, dclda=  0.01, thickness= 97.41. See
DS model deactivated: blade  3 at 3160mm. Max f_sep=  4.24, dclda=  0.01, thickness= 98.69. See
DS model deactivated: blade  3 at 5565mm. Max f_sep=  4.24, dclda=  0.02, thickness= 94.83. See
DS model deactivated: blade  3 at 4282mm. Max f_sep=  4.24, dclda=  0.01, thickness= 97.41. See
DS model deactivated: blade  3 at 7002mm. Max f_sep=  4.24, dclda=  0.04, thickness= 90.40. See
DS model deactivated: blade  3 at 5565mm. Max f_sep=  4.24, dclda=  0.02, thickness= 94.83. See
DS model deactivated: blade  3 at 8588mm. Max f_sep=  4.24, dclda=  0.07, thickness= 84.54. See
DS model deactivated: blade  3 at 7002mm. Max f_sep=  4.24, dclda=  0.04, thickness= 90.40. See
DS model deactivated: blade  3 at 10317mm. Max f_sep=  4.24, dclda=  0.10, thickness= 77.35. Se
DS model deactivated: blade  3 at 8588mm. Max f_sep=  4.24, dclda=  0.07, thickness= 84.54. See
DS model deactivated: blade  3 at 12181mm. Max f_sep=  4.24, dclda=  0.14, thickness= 69.22. Se
DS model deactivated: blade  3 at 10317mm. Max f_sep=  4.24, dclda=  0.10, thickness= 77.35. Se
DS model deactivated: blade  3 at 14172mm. Max f_sep=  4.24, dclda=  0.18, thickness= 60.80. Se
DS model deactivated: blade  3 at 12181mm. Max f_sep=  4.24, dclda=  0.14, thickness= 69.22. Se
DS model deactivated: blade  3 at 16283mm. Max f_sep=  1.12, dclda=  2.62, thickness= 53.05. Se
DS model deactivated: blade  3 at 14172mm. Max f_sep=  4.24, dclda=  0.18, thickness= 60.80. Se
DS model deactivated: blade  3 at 16283mm. Max f_sep=  1.12, dclda=  2.62, thickness= 53.05. Se
```

An xarray dataarray with the sensor data can be obtained by:

[15]:
```
da = wts.sensors.to_xarray()
da.head(time=1)
```

[15]: xarray.DataArray (**time**: 1, **wt**: 3, **sensor**: 13)

```
array([[[ 0.        ,  0.        , 10.19769323,  0.34039133,
         -0.12008897,  9.8198888 ,  0.08939756, -0.1043076 ,
          9.8198888 ,  0.08939756, -0.1043076 , 11.45097921,
          1.90896757],
        [ 0.        ,  0.        ,  8.87425122, -0.50322405,
          0.0887508 ,  9.99069778, -0.03992162, -0.21001935,
          9.99069778, -0.03992162, -0.21001935, 11.4391023 ,
          1.90643692],
        [ 0.        ,  0.        , 10.16976009, -0.08874703,
         -0.24468704, 10.3989547 , -0.12491694, -0.35079413,
         10.3989547 , -0.12491694, -0.35079413, 11.45097921,
          1.90896757]]])
```

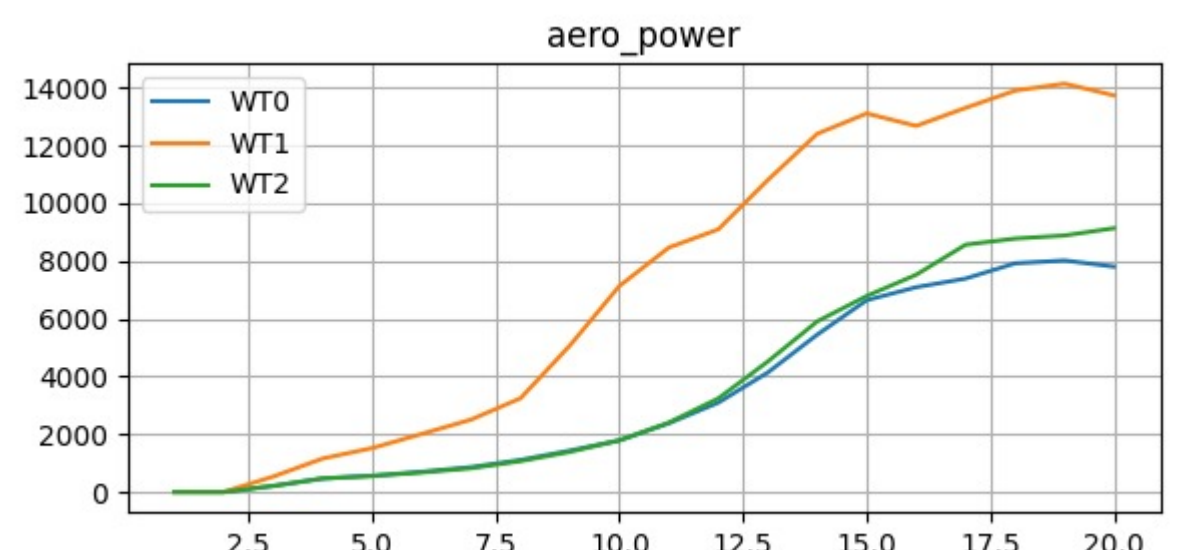
▼ Coordinates:

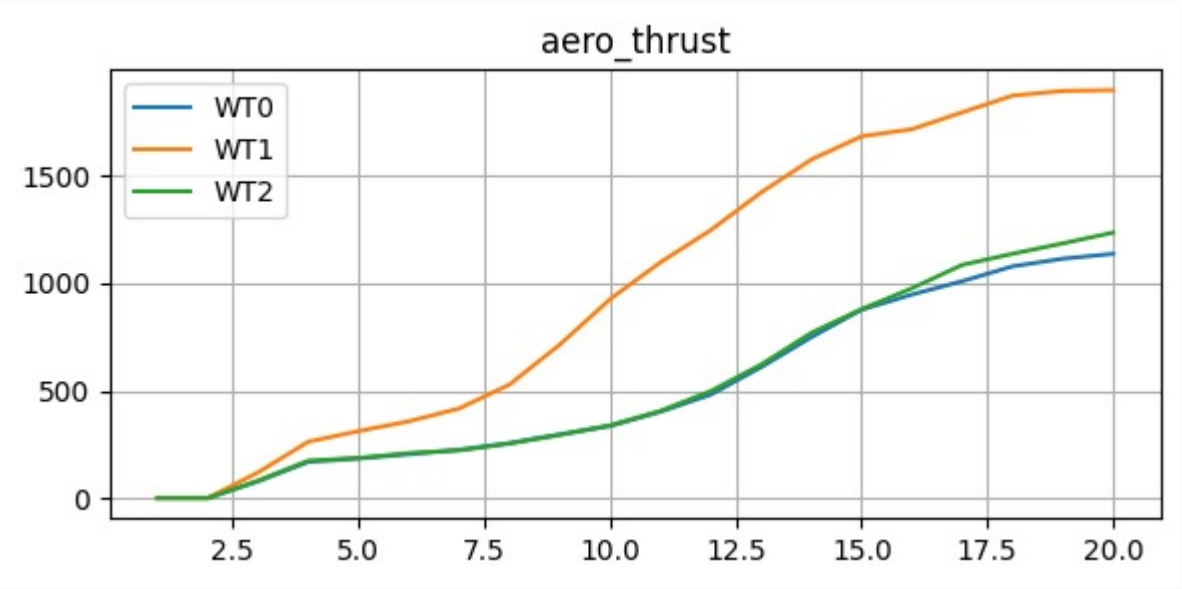
time	(time)	float64	1.0
wt	(wt)	int64	0 1 2
sensor	(sensor)	<U22	'aero_power' ... 'rotor_speed'

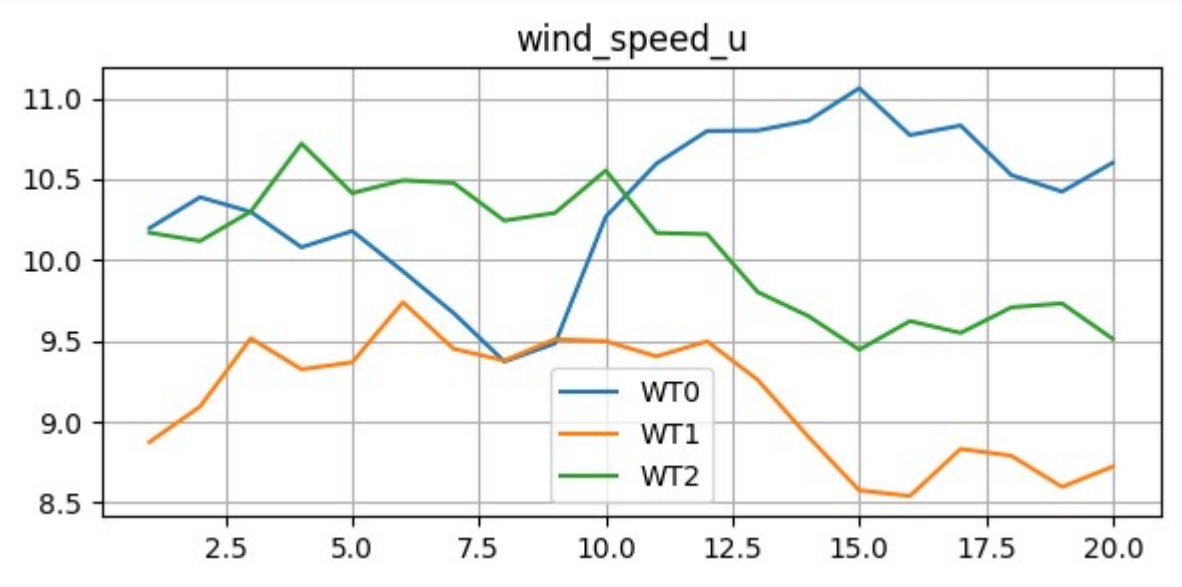
▶ Indexes: (3)

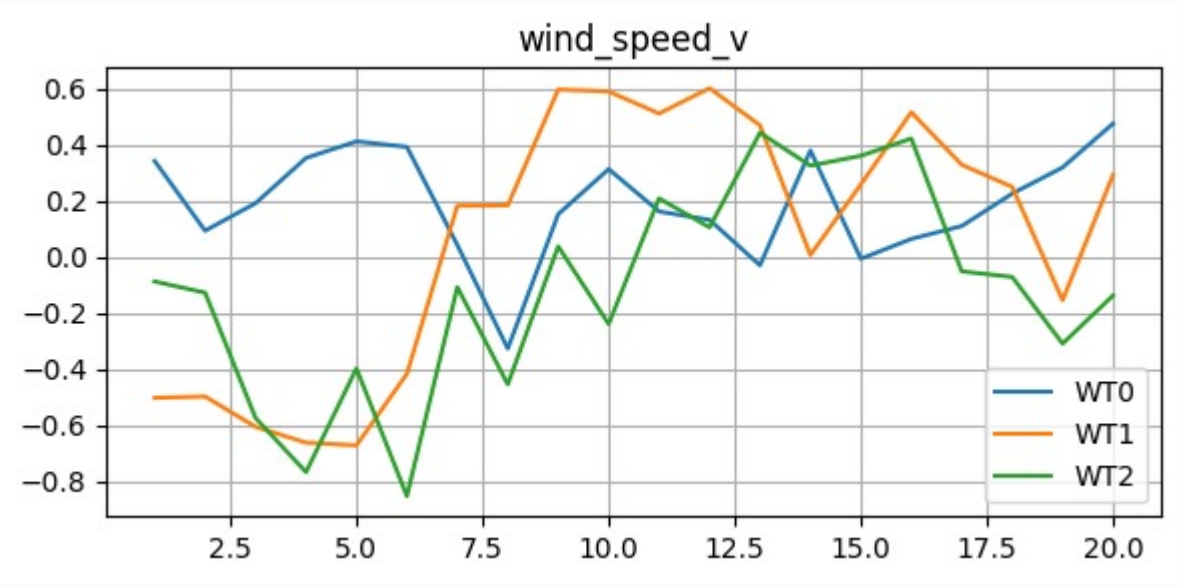
▶ Attributes: (0)

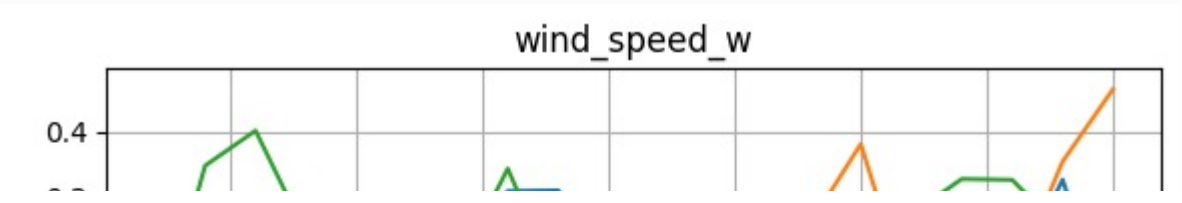
[16]:
```
for sensor in da.sensor:
    plt.figure(figsize=(6,3))
    for wt in da.wt:
        da.sel(sensor=sensor, wt=wt).plot(label=f'WT{wt.item()}')
    setup_plot(title=sensor.item())
```

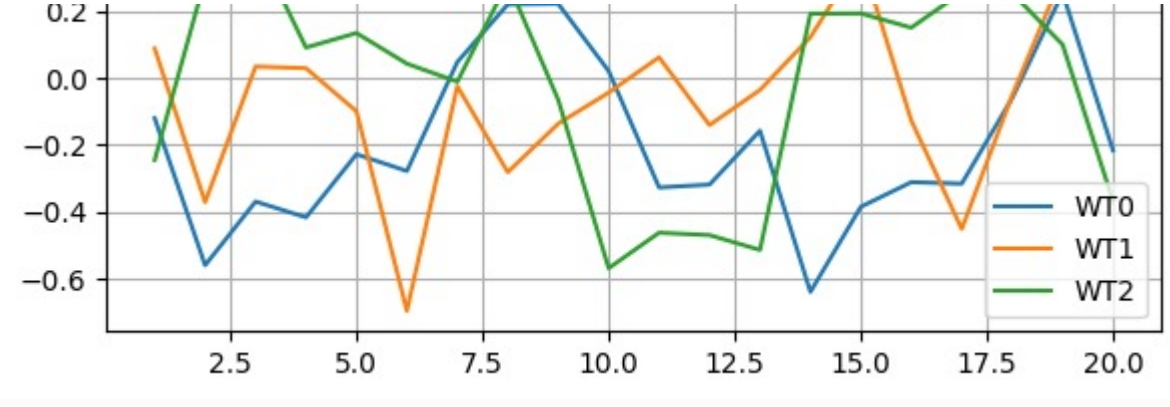


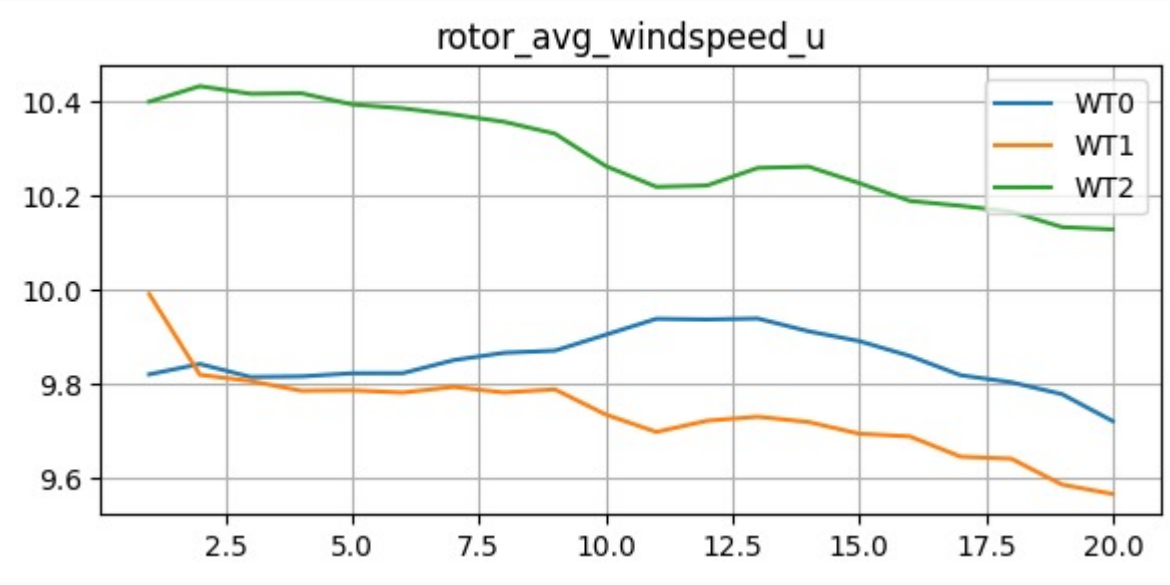

aero_thrust
WT0
WT1
WT2
1500
1000
500
0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

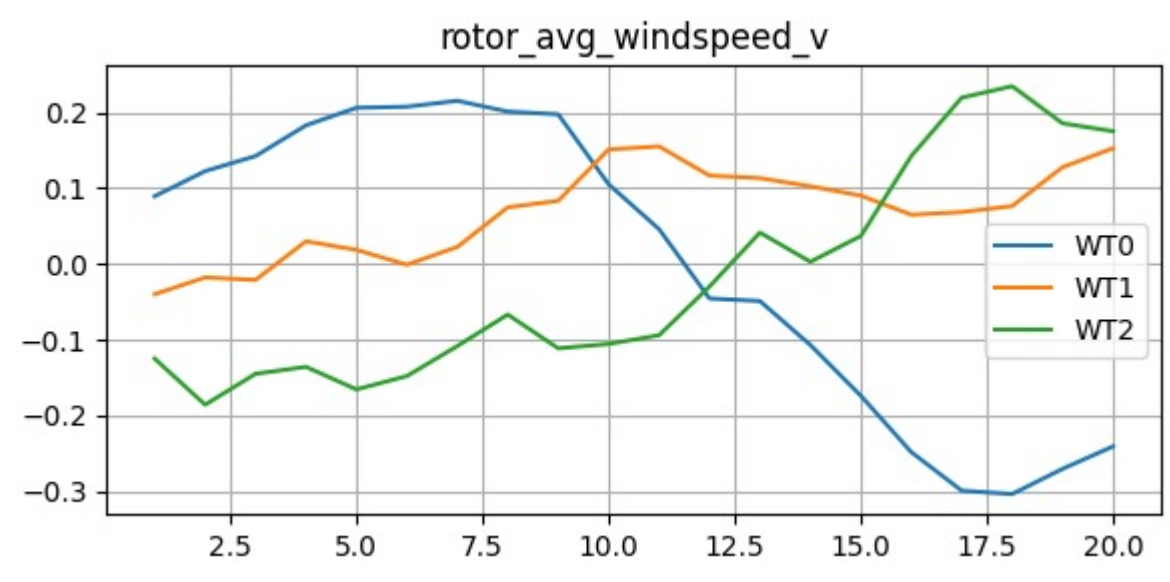

wind_speed_u
11.0
10.5
10.0
9.5
9.0
8.5
WT0
WT1
WT2
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

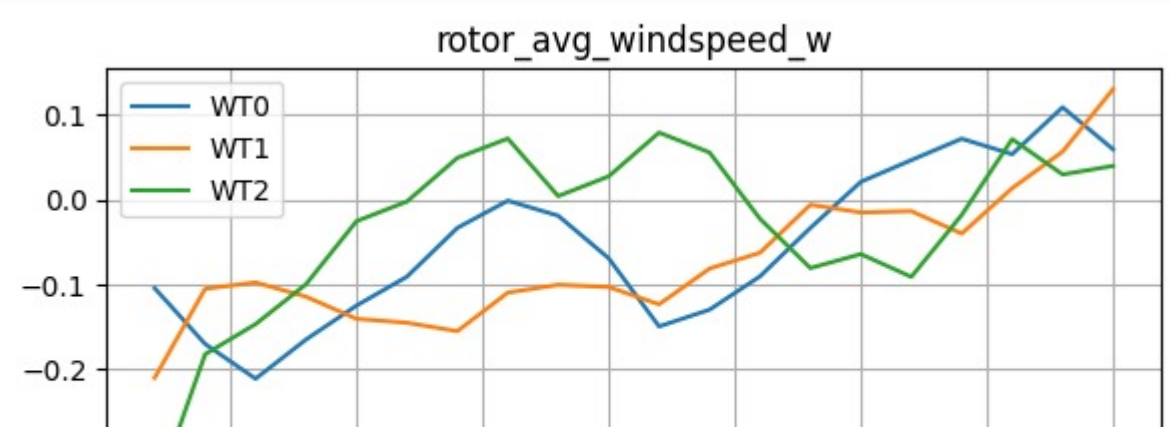

wind_speed_v
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
WT0
WT1
WT2
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

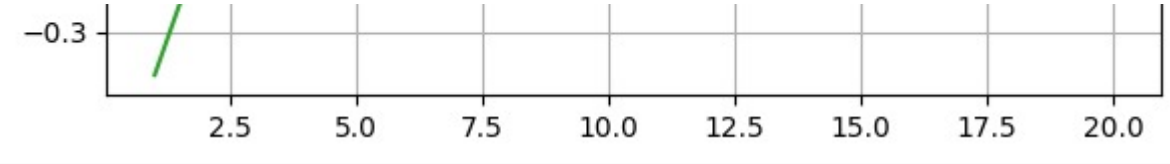

wind_speed_w
0.4

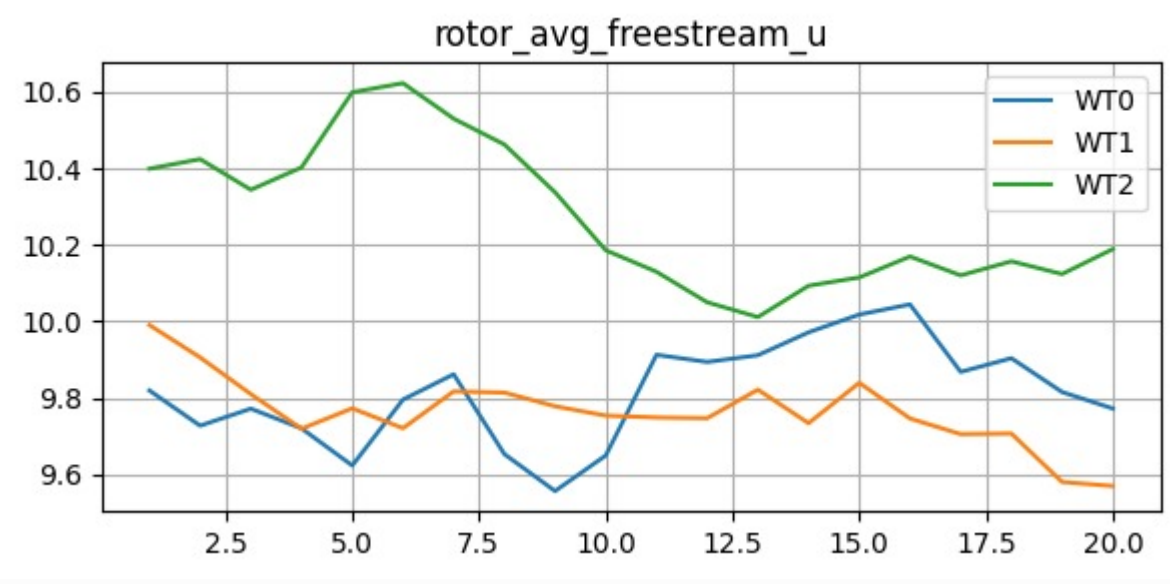

0.2
0.0
−0.2
−0.4
−0.6
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
WT0
WT1
WT2

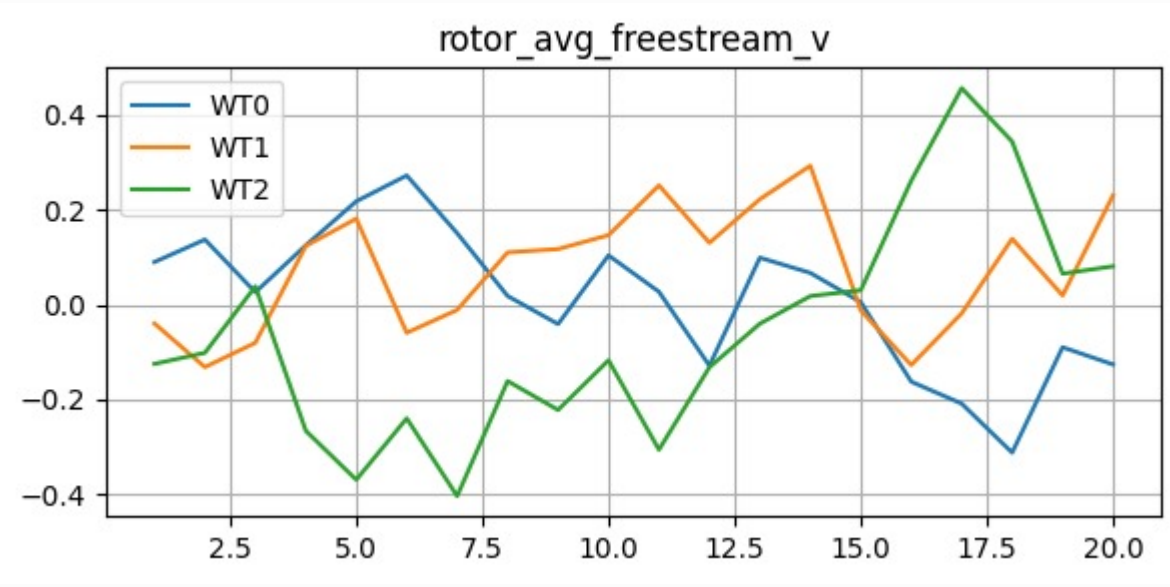

rotor_avg_windspeed_u
10.4
10.2
10.0
9.8
9.6
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
WT0
WT1
WT2

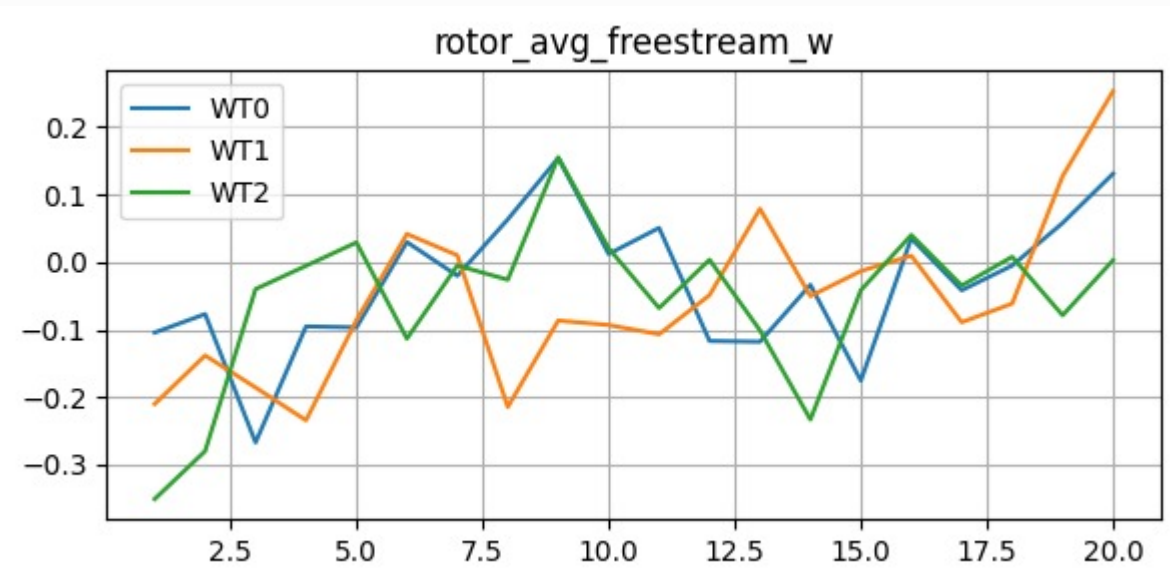

rotor_avg_windspeed_v
0.2
0.1
0.0
−0.1
−0.2
−0.3
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
WT0
WT1
WT2


rotor_avg_windspeed_w
0.1
0.0
−0.1
−0.2
WT0
WT1
WT2

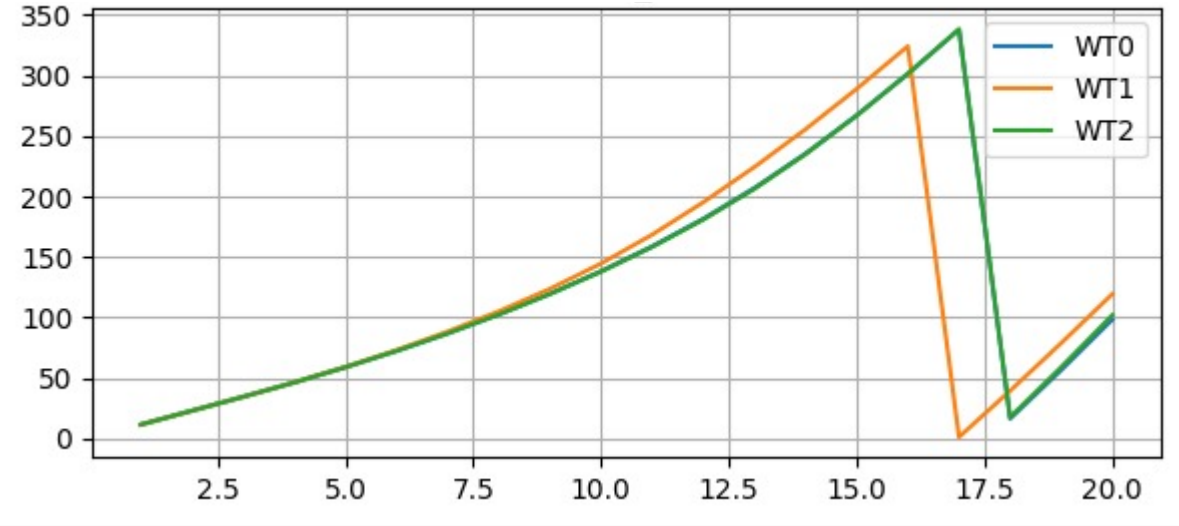

−0.3
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

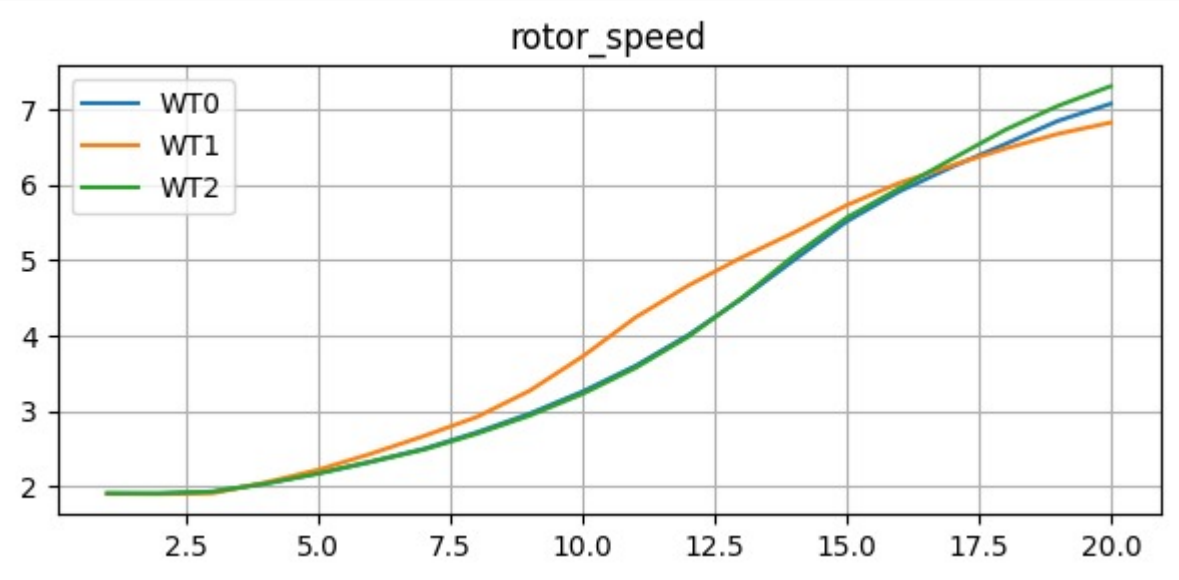

rotor_avg_freestream_u
WT0
WT1
WT2
10.6
10.4
10.2
10.0
9.8
9.6
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0


rotor_avg_freestream_v
WT0
WT1
WT2
0.4
0.2
0.0
−0.2
−0.4
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0


rotor_avg_freestream_w
WT0
WT1
WT2
0.2
0.1
0.0
−0.1
−0.2
−0.3
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

rotor_azi





HAWC2 result file

Output from HAWC2 can be obtained from a HAWC2 result file, when that file is written. When the file is written depends on

- htc.simulation.time_stop (result file output stops at time_stop, note, time_stop is overridden by htc.output.time)
- htc.output.data_format (`gtsdf`, `gtsdf64` and `hawc_ascii` are written during simulation while `hawc_binary` and `flex_int` are written at time stop)
- htc.output.buffer (`gtsdf`, `gtsdf64` and `hawc_ascii` every `buffer` time steps))
- htc.output.time

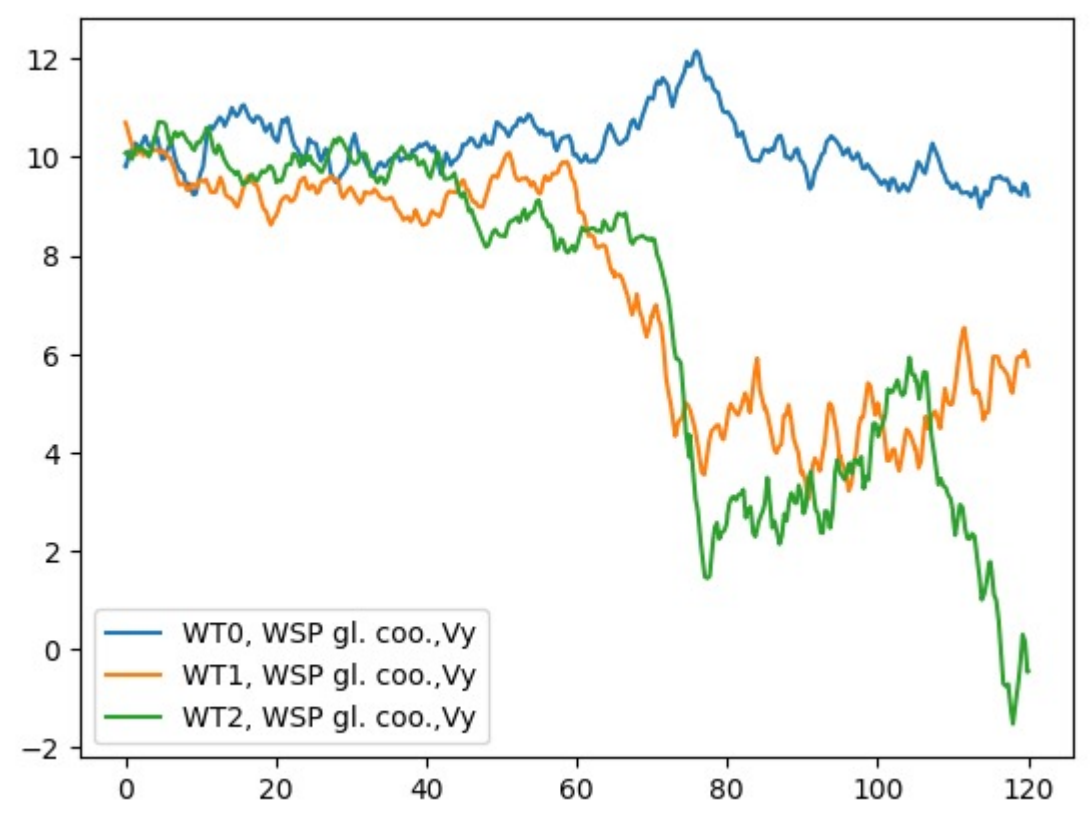
In this case, a result file has been written for all wind turbines

```
[17]: !ls C:\mmpe\gitlab\DYNAMIKS\.py311_dynamiks_jupyter7\Lib\site-packages\h2lib_tests\test_files\D
```

```
ls: cannot access 'C:mmpegitlabDYNAMIKS.py311_dynamiks_jupyter7Libsite-packagesh2lib_teststest_
```

[18]:
```
# copy and overwrite cached result files at cwd
if 0:
    import shutil
    import os
    os.makedirs('res/MyTestCase', exist_ok=True)
    for i in [0,1,2]:
        f = wts.htc_lst[i].output.filename.values[0] + ".hdf5"
        print (os.path.abspath(wts.htc_lst[i].modelpath + f))
        shutil.copy(os.path.abspath(wts.htc_lst[i].modelpath + f), os.path.abspath(f))
```

[19]:
```
from wetb.gtsdf import gtsdf
plt.figure()
for i in [0,1,2]:
    f = wts.htc_lst[i].output.filename.values[0] + ".hdf5"
    time, data, info = gtsdf.load(f)
    n = 13 # output sensor number
    plt.plot(time, data[:,n], label=f"WT{i}, {info['attribute_names'][n]}")
plt.legend();
```



[20]:
```
wts.h2.close()
```

```
 Creating file folder res/
Folder "res/" created.
Creating file/folder MyTestCase
Created folder: 'MyTestCase'
 Closing of external type2 DLL
 Closing of external type2 DLL
 Closing of external type2 DLL
 Closing of external type2 DLL
 Closing of external type2 DLL
 Elapsed time :    38.15625
 Creating file folder res/
Folder "res/" created.
Creating file/folder MyTestCase
Created folder: 'MyTestCase'
 Closing of external type2 DLL
 Closing of external type2 DLL
 Closing of external type2 DLL
```

```
 Closing of external type2 DLL
 Closing of external type2 DLL
 Elapsed time :    37.89062
 Creating file folder res/
Creating file/folder MyTestCase
Created folder: 'MyTestCase'
 Closing of external type2 DLL
 Closing of external type2 DLL
 Closing of external type2 DLL
 Closing of external type2 DLL
 Closing of external type2 DLL
 Elapsed time :    38.20703
```

Previous Next

Edit on Gitlab launch binder

Site

The `Site` object provides the free wind conditions at the site including mean wind, shear, turbulence etc.

Current implementations are:

- TurbulenceFieldSite

TurbulenceFieldSite

[1]:
```
# imports and default values
import numpy as np
import matplotlib.pyplot as plt
from py_wake.utils.plotting import setup_plot
from dynamiks.sites import TurbulenceFieldSite
from dynamiks.sites.turbulence_fields import MannTurbulenceField, RandomTurbulence
from dynamiks.sites.mean_wind import ConstantWindSpeed
from dynamiks.views import ZView, XYView
from dynamiks.utils.test_utils import DefaultDWMFlowSimulation

ws = 10
ti = 0.1
no_turb_field = RandomTurbulence(ti=0, ws=ws) # turbulence field without turbulence
```

The `TurbulenceFieldSite` implements a site defined by a wind speed (object) and a turbulence field

[2]:
```
help(TurbulenceFieldSite.__init__)
```

```
Help on function __init__ in module dynamiks.sites._site:

__init__(self, ws, turbulenceField, turbulence_offset=[0, 0, 0], turbulence_transport_speed=Non
    Site with mean wind and turbulence field

    Parameters
    ----------
    ws : int, float or MeanWind-object
        mean wind speed or MeanWind-object, e.g. ConstantWindSpeed
    turbulenceField : TurbulenceField
```

```
        E.g. MannTurbulenceField
    turbulence_offset : tuple(x,y,z), optional
        turbulence (x,y,z)-offset. Default is no offset
    turbulence_transport_speed : int, float or None, optional
        Turbulence field advection speed.
        If None (default), the speed is obtained from the mean wind speed
```

Mean wind

The mean wind argument can be either a constant uniform mean wind speed value

[3]:
```
site = TurbulenceFieldSite(ws, no_turb_field)
```

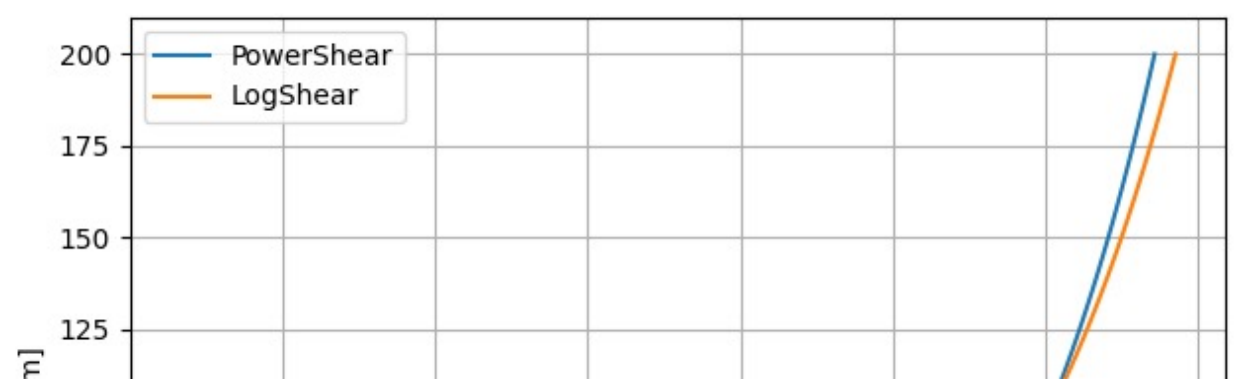
or a ConstantWindSpeed object

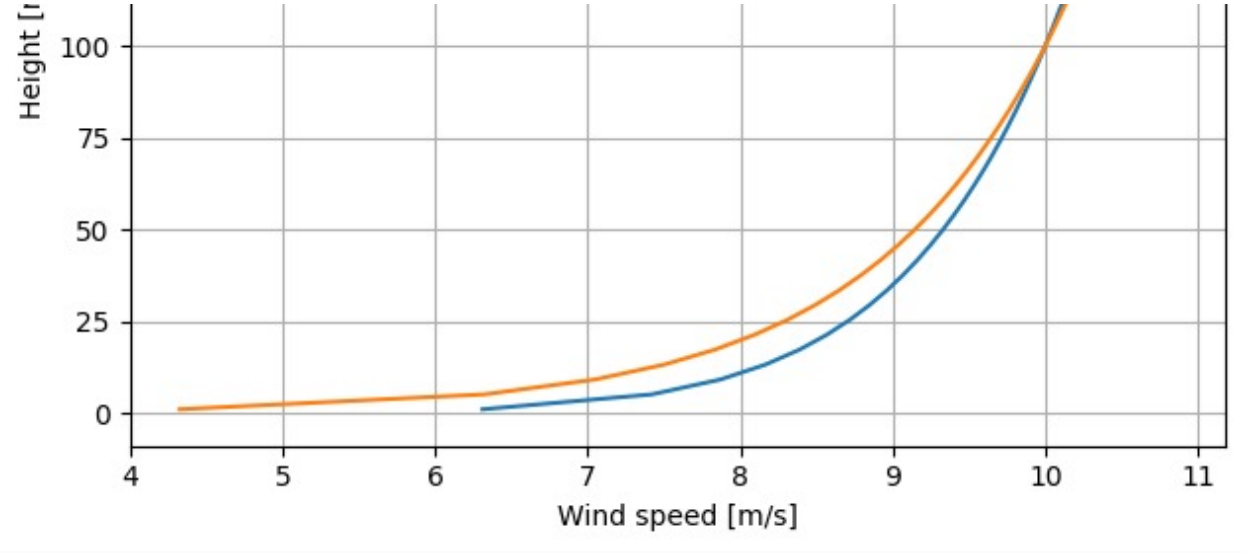
[4]:
```
const_ws = ConstantWindSpeed(ws=10)
site = TurbulenceFieldSite(const_ws, no_turb_field)
```

Shear

The latter `ConstantWindSpeed` option allows specification of vertical shear in terms of a PyWake shear object, e.g. `PowerShear` or `LogShear`

[5]:
```
import numpy as np
from py_wake.site.shear import PowerShear, LogShear


# PyWake shears
power_shear = PowerShear(h_ref=100, # reference height
                         alpha=0.1) # power shear exponent

log_shear = LogShear(h_ref=100, # reference height
                     z0=0.03)   # roughness length

view = ZView(x=0,y=0, z=np.linspace(1,200))

for shear in [power_shear, log_shear]:
    const_ws = ConstantWindSpeed(ws=10, shear=shear)
    site = TurbulenceFieldSite(const_ws, no_turb_field)
    u,v,w = site.get_windspeed(view)
    plt.plot(u, view.z, label=shear.__class__.__name__)
setup_plot(xlabel='Wind speed [m/s]', ylabel='Height [m]')
```





TurbulenceField

Current implementations

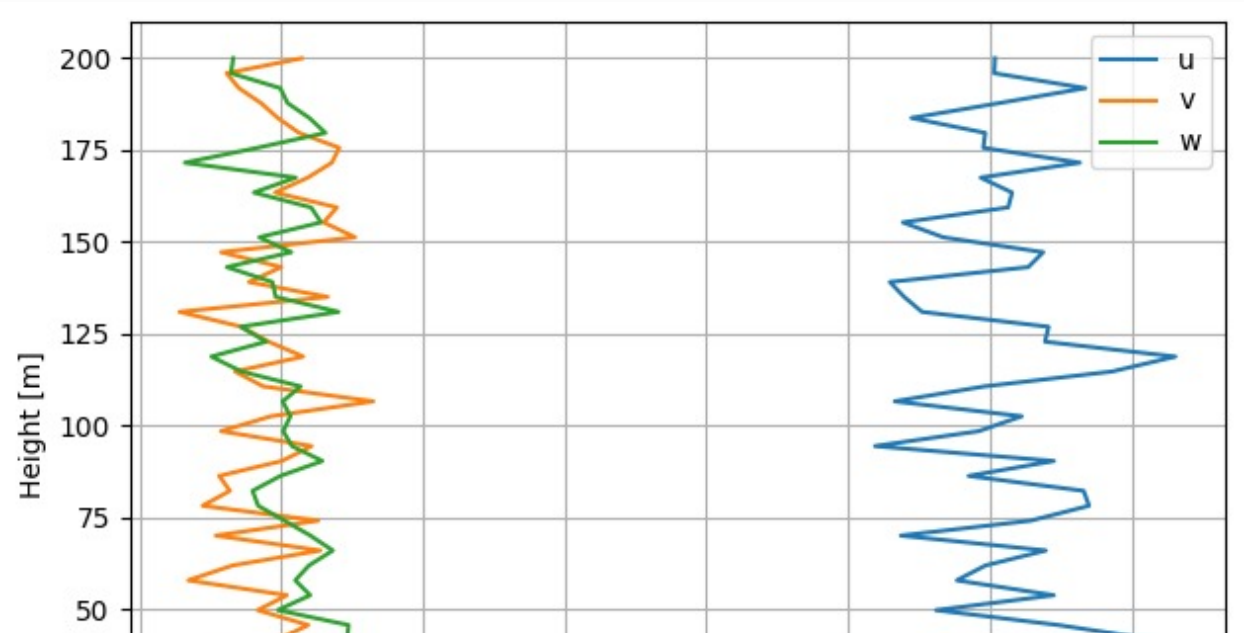
- RandomTurbulence
- MannTurbulenceField

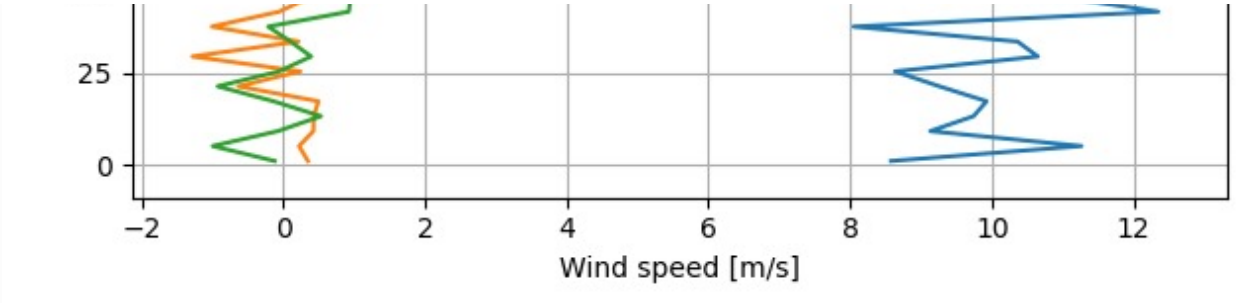
RandomTurbulence

The `RandomTurbulence` produces random turbulence. It is very fast but mainly usable for tests

[6]:
```
from dynamiks.sites.turbulence_fields import RandomTurbulence
site = TurbulenceFieldSite(10, RandomTurbulence(ti=ti, ws=ws))
```

[7]:
```
uvw = site.get_windspeed(view)
for ws, n in zip(uvw, 'uvw'):
    plt.plot(ws, view.z, label=n)
setup_plot(xlabel='Wind speed [m/s]', ylabel='Height [m]')
```





MannTurbulenceField

The `MannTurbulenceField` extends the `hipersim.MannTurbulenceField` from (see https://hipersim.pages.windenergy.dtu.dk/hipersim/MannTurbulenceField.html).

The turbulence can be generated via Hipersim and/or read from a netcdf or the tree-file binary Mann turbulence format.

The **turbulence field size** should cover the whole wind farm in all dimensions and time:

- Lx = Nx*dx: Farm length in direction of wind + Simulation time* turbulence transport speed.
- Ly = Ny*dy: Farm width (perpendicular to the wind)
- Lz = Nz*dz: from ground to blade tip height

The needed **grid resolution** depends on the wind turbine model. For HAWC2 (actuator line), Liew (2022) recommends a resolution of D/50.

In a study of the Lillgrund wind farm (Liew 2023), a turbulence field with 38880x1792x64 grid points is used. The size of this field is 50Gb (3 uvw components and 32bit floats). Note, that such big files usually requires a HPC system, as personal computers have far less memory.

Reference:

- Jaime Liew and Gunner Chr. Larsen 2022 J. Phys.: Conf. Ser. 2265 032049, https://doi.org/10.1088/1742-6596/2265/3/032049)
- Jaime Liew et al 2023 J. Phys.: Conf. Ser. 2626 012050

Generate turbulence field via Hipersim

Use the `MannTurbulenceField.generate` method to generate new turbulence fields. See documentation

[8]:
```
from dynamiks.sites.turbulence_fields import MannTurbulenceField
mtf = MannTurbulenceField.generate(alphaepsilon=.1, # use correct alphaepsilon or scale later
                                   L=33.6, # length scale
                                   Gamma=3.9, # anisotropy parameter
```

```
                                Nxyz=(1024,64,64), # should be large enough to cover whole f
                                dxyz=(2,2,2), # should be small enough to capture variations
                                seed=1, # seed for random generator
                                HighFreqComp=0, # the high frequency compensation is questic
                                double_xyz=(False, False, False), # turbulence periodicity i
                               )
```

You can **save** the generated turbulence field for later use, see Hipersim documentation

[9]:
```
mtf.to_netcdf(filename=None)
```

If filename=None (default), the name is autogenerated from the properties. In this case the filename is `Hipersim_mann_l33.6_ae0.1000_g3.9_h0_1024x64x64_2.000x2.00x2.00_s0001.nc`

Read turbulence field from file

When saving and loading Hipersim turbulence file, it is recommeneded to use the netcdf format as all data and attributes are stored in one file.

[10]:
```
mtf = MannTurbulenceField.from_netcdf('Hipersim_mann_l33.6_ae0.1000_g3.9_h0_1024x64x64_2.000x2.
```

[11]:
```
mtf.to_xarray()
```

[11]:

xarray.DataArray (**uvw**: 3, **x**: 1024, **y**: 64, **z**: 64)

```
array([[[[-0.35261983, -0.08925066, -0.23356584, ..., -1.1009454 ,
          -0.8976475 , -0.76379365],
         [-0.9250733 , -0.4931373 , -1.1856041 , ..., -0.7663951 ,
          -0.7959621 , -1.1620626 ],
         [-0.8972635 , -0.94220793, -1.0300303 , ..., -0.77264917,
          -0.5917986 , -0.7867458 ],
         ...,
         [-0.6236845 , -0.5403369 , -0.41061032, ..., -0.15890206,
          -0.06358869, -0.01130451],
         [-0.38042337, -0.46187547, -0.43752527, ..., -1.1627533 ,
          -0.7095258 , -0.2183148 ],
         [-0.6119078 ,  0.02988166,  0.13592918, ..., -1.3156966 ,
          -0.8855437 , -0.66417706]],

        [[-0.19947052, -0.3182267 , -0.037643  , ..., -1.2702348 ,
          -0.89583623, -0.6222654 ],
         [-0.9250941 , -0.7349631 , -0.55023694, ..., -0.95171165,
          -0.67152786, -0.673831  ],
         [-1.3296094 , -1.0069457 , -0.695421  , ..., -0.9798774 ,
          -1.1212653 , -0.9459772 ],
...
         [ 0.37815997,  0.36219364,  0.27982154, ...,  0.3620944 ,
```

```
       0.16375346,  0.14584586],
      [ 0.40218484,  0.20448992,  0.2411005 , ...,  0.6785034 ,
       0.4723086 ,  0.6304167 ],
      [ 0.2964676 ,  0.43353313,  0.37929705, ...,  0.6079931 ,
       0.48515218,  0.39158955]],

     [[ 0.7356683 ,  0.46824086, -0.03418609, ...,  0.47688082,
       0.55650324,  0.6406135 ],
      [ 0.44476473, -0.13761413, -0.32461864, ...,  0.37383848,
       0.2806024 ,  0.34730986],
      [-0.11733318, -0.36337373, -0.18907116, ...,  0.26693332,
       -0.17221646, -0.19901314],
      ...,
      [-0.15419303,  0.00421654,  0.08828714, ...,  0.27103174,
       0.35762107,  0.00870521],
      [ 0.3831636 ,  0.09581088,  0.22962908, ...,  0.7461332 ,
       0.7619566 ,  0.6896519 ],
      [ 0.5637457 ,  0.29805893,  0.32872304, ...,  0.67402774,
       0.84536624,  0.88036865]]]], dtype=float32)
```

▼ Coordinates:

x	(x)	int64	0 2 4 6 8 ... 2040 2042 2044 2046	
y	(y)	int64	0 2 4 6 8 ... 118 120 122 124 126	
z	(z)	int64	0 2 4 6 8 ... 118 120 122 124 126	
uvw	(uvw)	<U1	'u' 'v' 'w'	

▶ Indexes: (4)

▼ Attributes:

double_xyz :	[0 0 0]
name :	Hipersim_mann_l33.6_ae0.1000_g3.9_h0_1024x64x64_2.000x2.00x2.00_s0001
alphaepsilon :	0.1
L :	33.6
Gamma :	3.9

HighFreqComp :	0
Generator :	Hipersim
seed :	1

It is also possible to save and load the tree-file binary Mann turbulence format, see Hipersim documentation but the netcdf format is recommended as you do not risk to mix up files and/or attributes

Turbulence intensity and scaling

When dealing with turbulence intensity, $TI = U/u'$, of turbulence fields used for wind farms, some additional things need to be considered:

- U is the mean wind speed, but where and when, and does it include wakes
- u' is the fluctuations, but how long is the measurement period and what is the cut-off frequency of the measurement device
- Are you interested in the turbulence intensity of the theoretical Mann model spectrum or the actual turbulence field realization (in a point, an area or the whole turbulence field).
- Are U and u', the longitudinal, horizontal or total speeds

For a given mean wind speed, `U`, measurement period, `T`, and cut-off frequency, `cutoff_frq`, the turbulence intensity of the theoretical Mann model spectrum can be obtained by:

[12]:
```
U = 10
T = 600
cutoff_frq = 10
mtf.spectrum_TI(U=U, T=T, cutoff_frq=cutoff_frq)
```

[12]:
```
np.float64(0.14324855863543767)
```

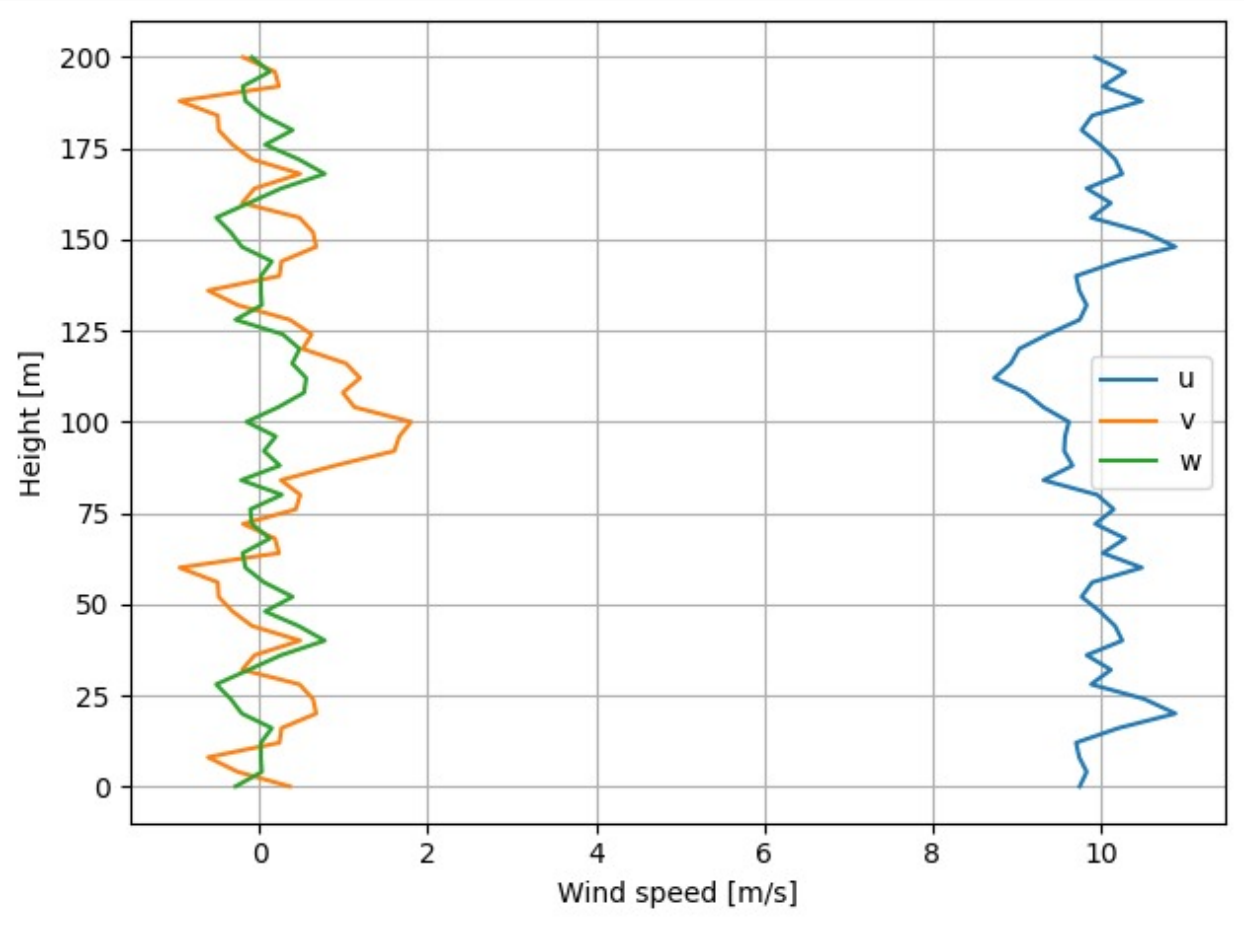
Scaling the turbulence field, such that the underlying theoretical Mann model spectrum has a specific turbulence intensity is done with the `scale_TI` method:

[13]:
```
print (f'Before: Box TI={mtf.uvw[0].std()/U:.3f}, alphaepsilon:{mtf.alphaepsilon:.3f}, theoreti
mtf.scale_TI(TI=0.1, U=10,  T=600, cutoff_frq=10)
print (f'After: Box TI={mtf.uvw[0].std()/U:.3f}, alphaepsilon:{mtf.alphaepsilon:.3f}, theoretic
```

```
Before: Box TI=0.129, alphaepsilon:0.100, theoretical spectrum TI 0.13
After: Box TI=0.090, alphaepsilon:0.049, theoretical spectrum TI 0.09
```

[14]:
```
from dynamiks.sites.turbulence_fields import RandomTurbulence
site = TurbulenceFieldSite(ws=10, turbulenceField=mtf)
```

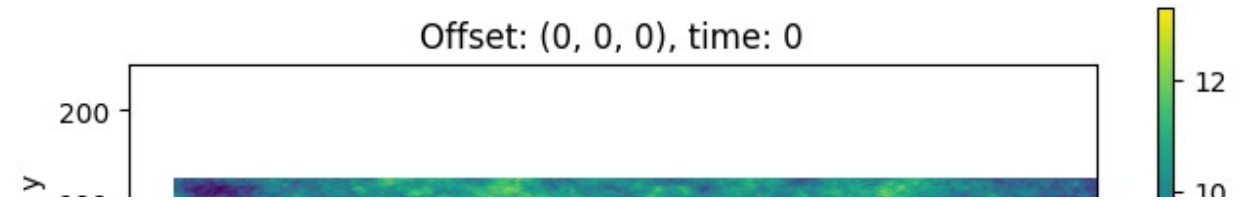
[15]:
```
view = ZView(x=0,y=0, z=np.linspace(1,200))
uvw = site.get_windspeed(view)
for ws, n in zip(uvw, 'uvw'):
    plt.plot(ws, view.z, label=n)
setup_plot(xlabel='Wind speed [m/s]', ylabel='Height [m]')
```

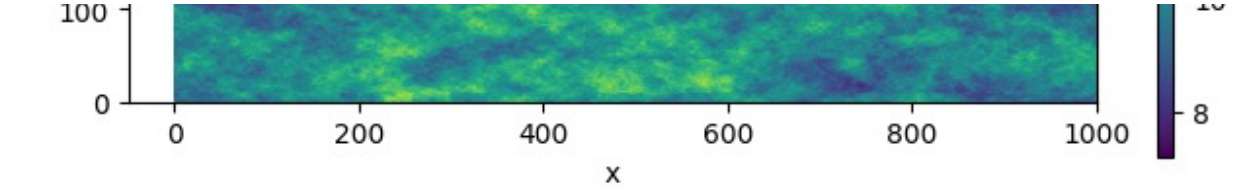


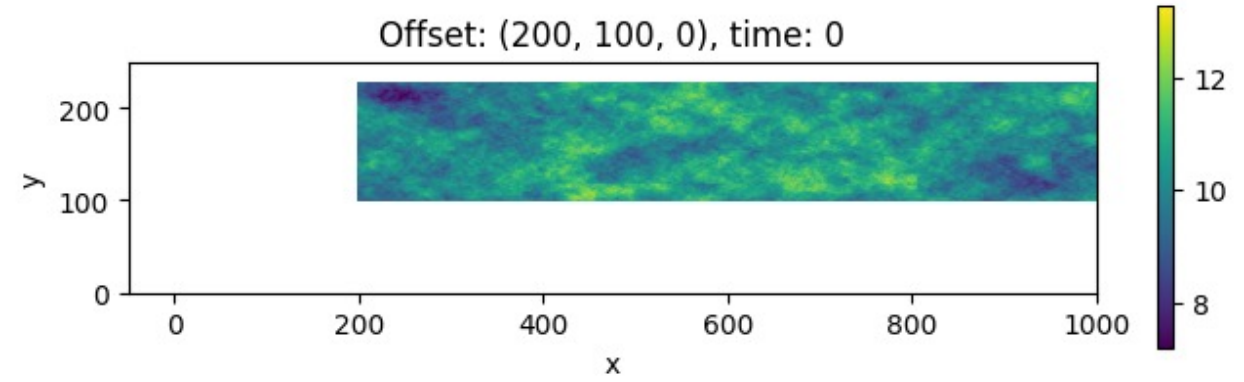
Turbulence offset

The `turbulence_offset` argument of `TurbulenceFieldSite` specifies the turbulence field origo at time=0

[16]:
```
view = XYView(z=100)
axes = plt.subplots(2,1, figsize=(8,5))[1]
for offset, ax in zip([(0,0,0), (200,100,0)], axes):
        site = TurbulenceFieldSite(ws=10, turbulenceField=mtf, turbulence_offset = offset)
        fs = DefaultDWMFlowSimulation(site=site)
        uvw = fs.get_windspeed(view, include_wakes=False, xarray=True)
        uvw.sel(uvw='u').plot(ax=ax)
        ax.axis('scaled')
        ax.set_xlim([-50,1000])
        ax.set_ylim([0,250])
        ax.set_title(f'Offset: {offset}, time: 0')
```



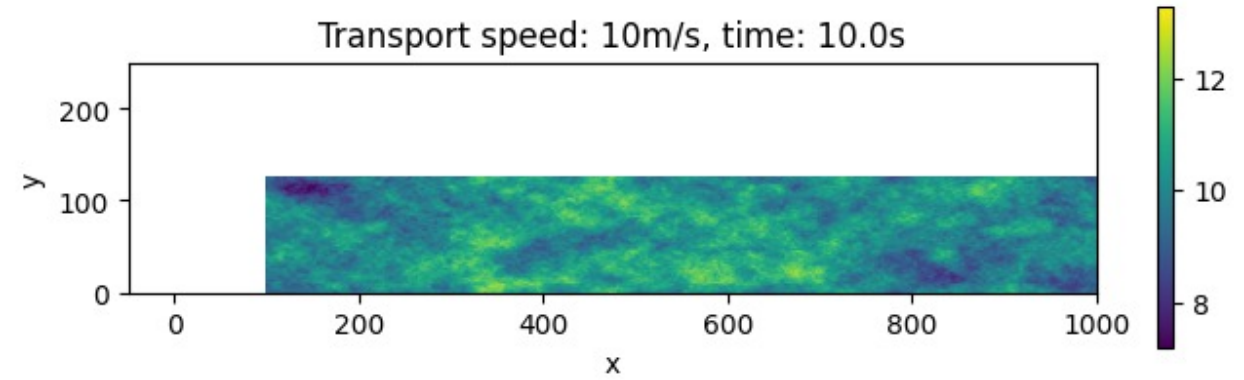


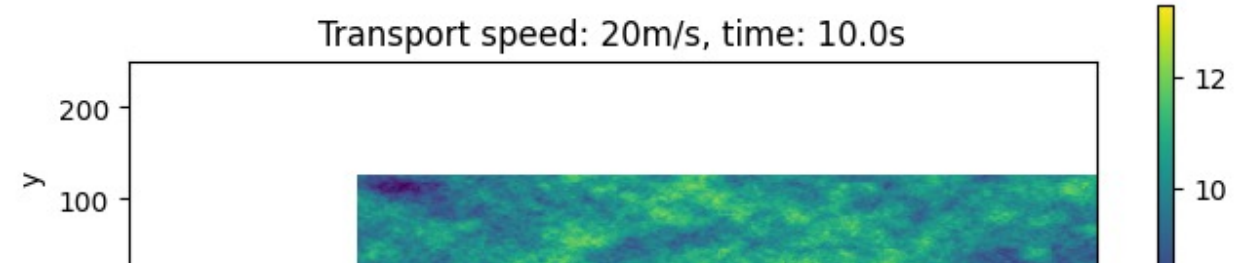


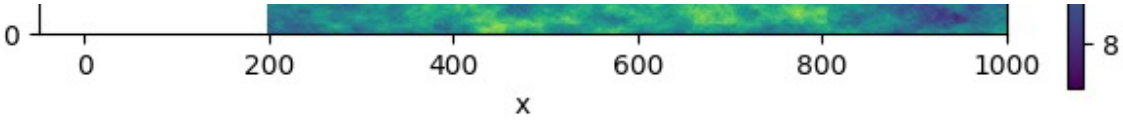
Turbulence transport speed

The `turbulence_transport_speed` argument of TurbulenceFieldSite specifies the advection speed of the turbulence field

[17]:
```
view = XYView(z=100)
axes = plt.subplots(2,1, figsize=(8,5))[1]
for transport_speed, ax in zip([10,20], axes):
        site = TurbulenceFieldSite(ws=10, turbulenceField=mtf, turbulence_transport_speed=trans
        fs = DefaultDWMFlowSimulation(site=site)
        fs.run(10) # run 10s
        uvw = fs.get_windspeed(view, include_wakes=False, xarray=True)
        uvw.sel(uvw='u').plot(ax=ax) # plot u component
        ax.axis('scaled')
        ax.set_xlim([-50,1000])
        ax.set_ylim([0,250])
        ax.set_title(f'Transport speed: {transport_speed}m/s, time: {fs.time}s')
```







LES precursor

Precursors are typically turbulence field extracted from CFD LES simulations, i.e. simulations where the turbulence has dimenstions $(uvw, time, x, y, z)$.

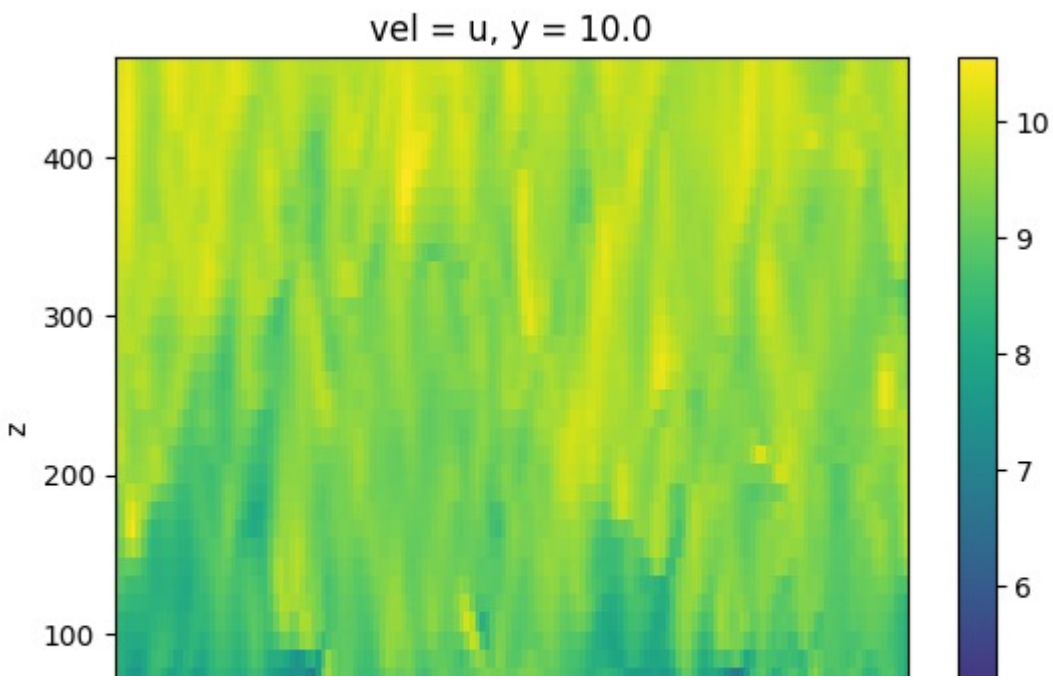
Often precursor fields are sampled at a plane some distance upstream of the wind farm or wind turbine, which reduces the dimensions to $(uvw, time, y, z)$.

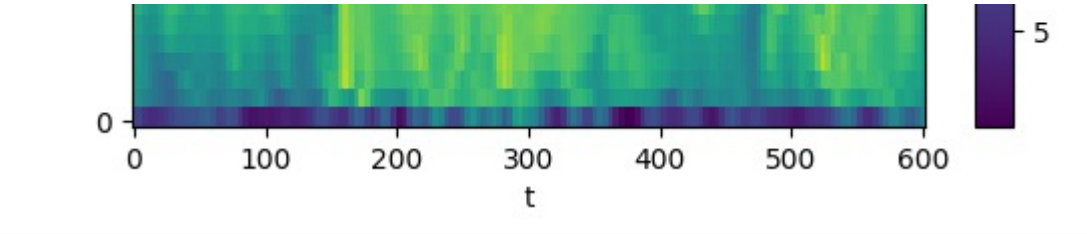
These can be used in Dynamiks similar to normal turbulence fields, but we need to convert the time axis into x coordinate and assume Taylor's frozen turbulence hypothesis, to obtain turbulence in other areas of the domain.

[18]:
```
import warnings
import xarray as xr
import numpy as np
from numpy import newaxis as na
import matplotlib.pyplot as plt
from dynamiks.sites import TurbulenceFieldSite
from dynamiks.utils.test_utils import DefaultDWMFlowSimulation, tfp # test_files path
from dynamiks.sites.turbulence_fields import TurbulenceField
from dynamiks.views import XYView, XZView, MultiView
from hipersim import Bounds
from py_wake.utils.plotting import setup_plot
```

First we load the precursor, in this case a downsampled precursor obtained from Ellipsys.

[19]:
```
da = xr.load_dataarray(tfp + "precursor.nc")
da.sel(vel='u').isel(y=0).plot(x='t')
```

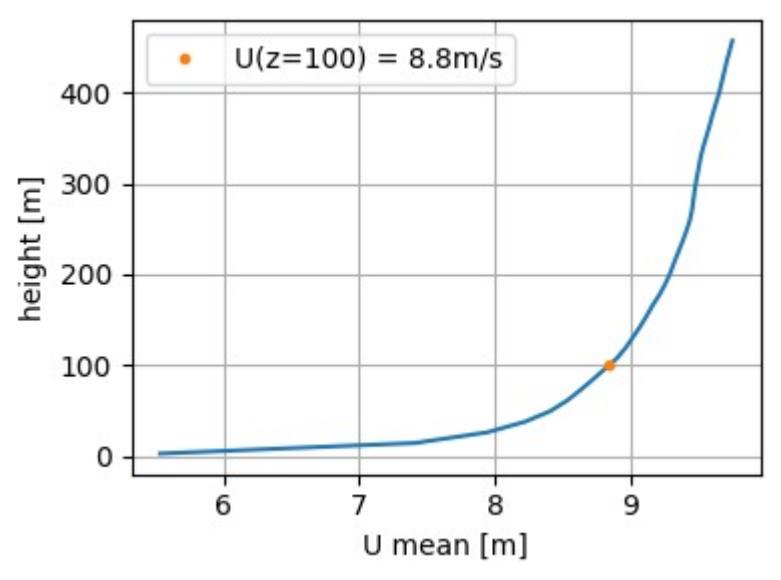




To map the time axis into x-coordinates, we need an advection speed. It can specified or extracted as the mean wind speed at a specific height. In this case we take the wind speed at $z = 100m$

[20]:
```
U_z = da.sel(vel='u').mean(('y', 't'))
advection_speed = U_z.interp(z=100).item()
```

[21]:
```
U_z.plot(y='z')
plt.plot(advection_speed,100,'.', label=f'U(z=100) = {advection_speed:.1f}m/s')
setup_plot(xlabel='U mean [m]', ylabel='height [m]', figsize=(4,3))
```

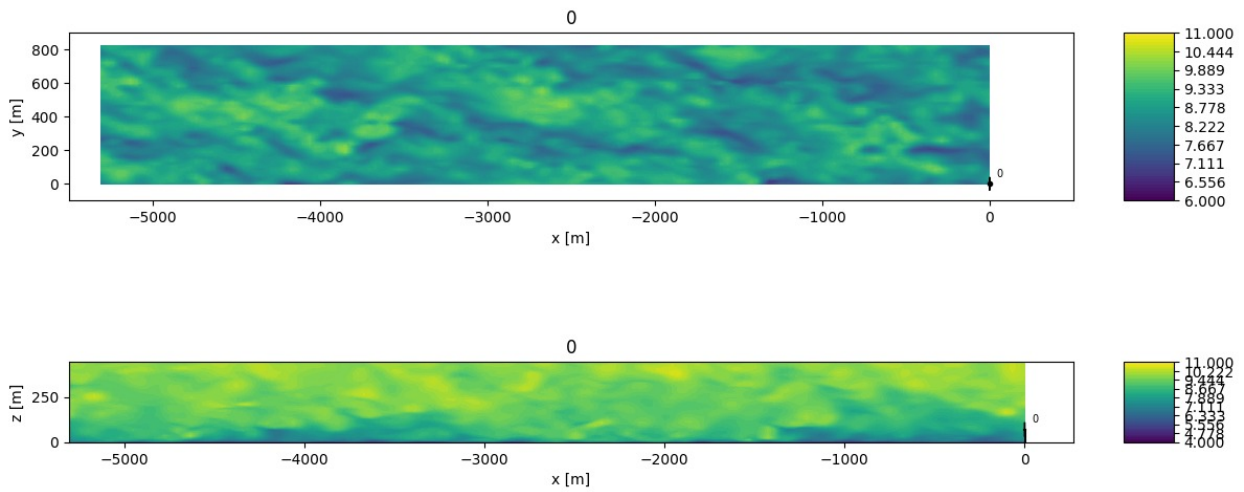


Having the advection speed, we can find the distance between time samples

[22]:
```
dt, dy, dz = [np.diff(v[:2])[0] for v in [da.t, da.y, da.z]]
dx = dt * advection_speed
```

We can now instantiate a site with the precursor turbulence and plot it

[23]:
```
def get_flowSimluation(bounds=Bounds.Error, offset=(0,0,0), da=da):
    Nxyz = Nx,Ny,Nz = da.shape[1:] # first dimension is uvw
    tf = TurbulenceField(da.values[:,::-1], # reverse x axis to move precursor forward
                         Nxyz=Nxyz, dxyz=[dx, dy, dz], bounds=bounds)
    site = TurbulenceFieldSite(ws=0, turbulenceField=tf, turbulence_transport_speed=advection_s
                               turbulence_offset=np.array([-(Nx-1)*dx,0,0]) + offset) # add box
    return DefaultDWMFlowSimulation(x=[0], y=[0], site=site)

fs = get_flowSimluation(Bounds.Repeat)
ax1,ax2 = plt.subplots(2,1, figsize=(12,6))[1]
fs.show(MultiView([XYView(ax=ax1, xlim=[-5500,500], ylim=[-100,900]), XZView(y=0, ax=ax2)]))
```



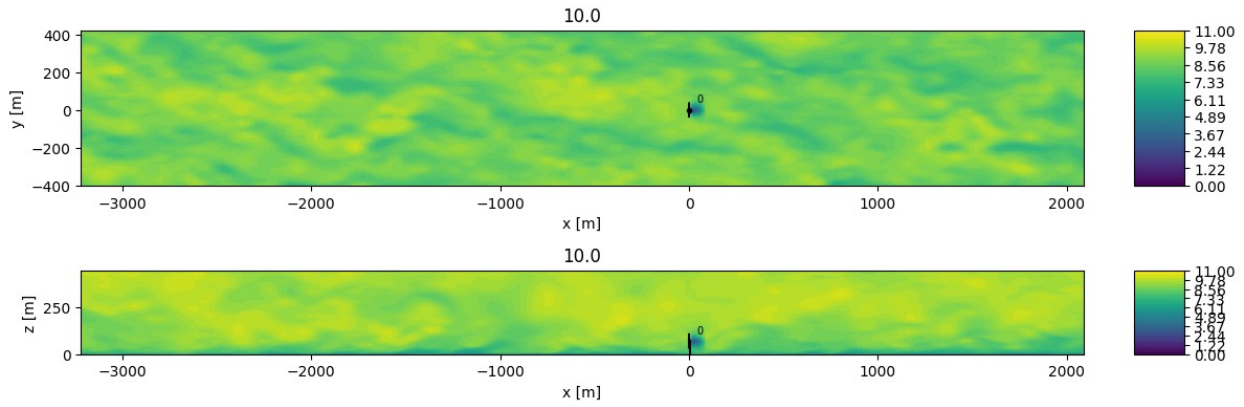
The turbine is located at $(0, 0)$, so half of the rotor and all wake particles are outside the precursor field. Before running the simulation we need to consider how to deal with areas outside the box. There are different options:

- Offset the box to cover the wind farm
- Repeat or mirror the box
- Use edge values
- Extend the the box size

Offset box

We can offset the precursor field such that it covers the turbine and all wake particles during the whole simulation.

[24]:
```
fs = get_flowSimluation(offset=(2000,-400,0), bounds=Bounds.Error)
ax1,ax2 = plt.subplots(2,1, figsize=(12,6))[1]
fs.visualize(10, MultiView([XYView(ax=ax1), XZView(y=0, ax=ax2)]))
```



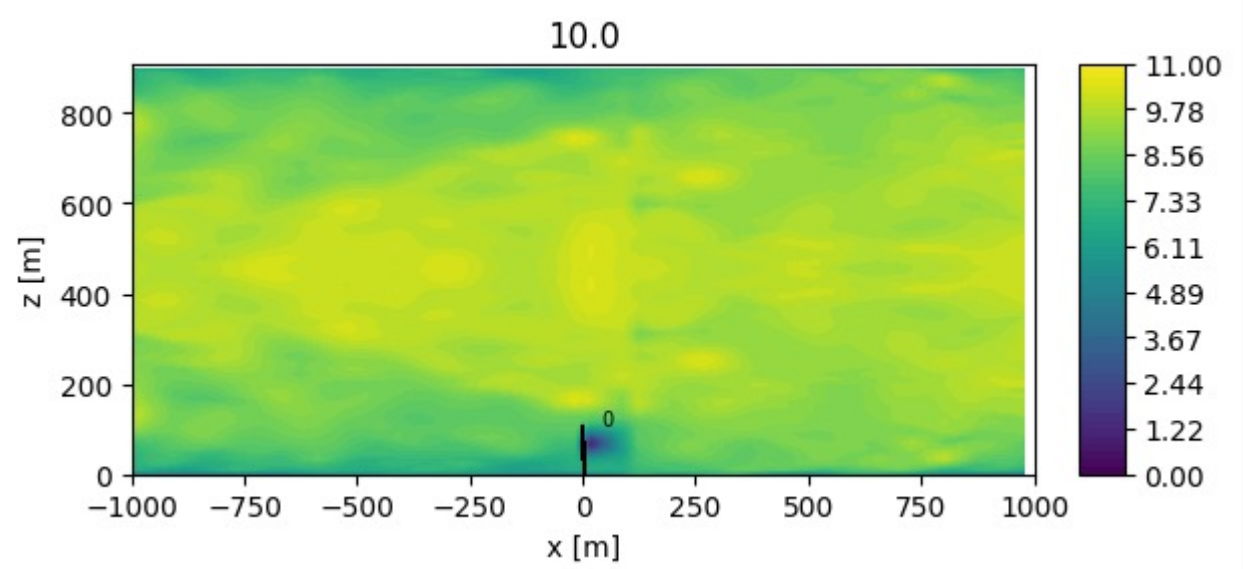
Repeat or mirror the box

In the visualization below, the box is repeated in the x direction and mirrored in the y and z direction.

Note:

- If the box is not periodic, repeating it may lead to discontinuities, as seen in the x direction near the turbine
- If the box is not homogeneous, mirroring or repeating it leads to wrong behavour as seen below where the low wind speed near the ground is applied again around $z = 900$

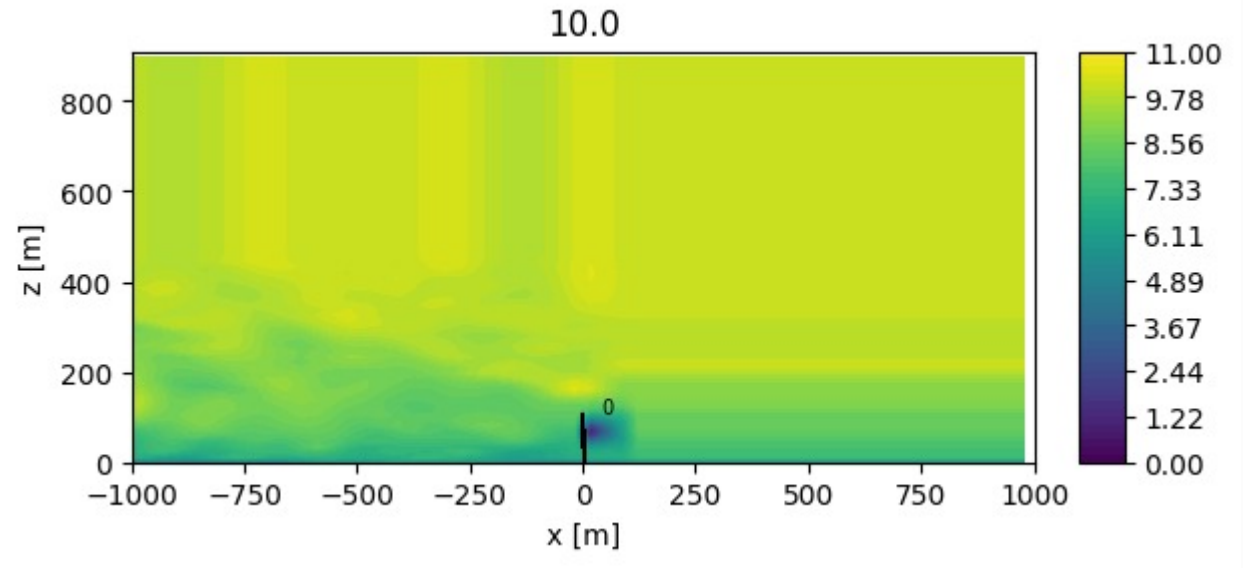
[25]:
```
fs = get_flowSimluation(bounds=[Bounds.Repeat,Bounds.Mirror,Bounds.Mirror])
fs.visualize(10, view=XZView(y=0, x=np.linspace(-1000,1000), z=np.linspace(0,910)))
```



Use Edge values

The bounds option `Bounds.Warning` gives a warning when requesting wind speed outside the box and returns the nearest valid value, i.e. the wind speed at the edge of the box.

[26]:
```
with warnings.catch_warnings(action='ignore'):
    fs = get_flowSimluation(bounds=[Bounds.Warning,Bounds.Warning,Bounds.Warning])
    fs.visualize(10, view=XZView(y=0, x=np.linspace(-1000,1000), z=np.linspace(0,910)))
```

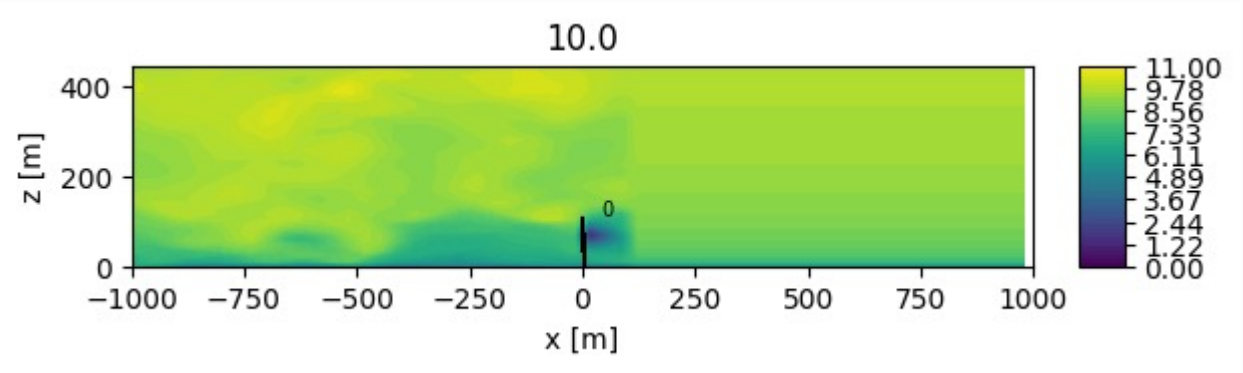


Extend the box

In case you know a better estimate of the wind speed outside the box, the wind field can be extended.

Below, the wind speed in 50 time steps in front of the box is set to the mean $uvw(z)$ profile.

[27]:
```
da_mean = da[:,:50]*0+da.mean(('y','t')) # 50 time steps of uvw(z) profile
coords = dict(vel=['u','v','w'], t=np.arange(0,200)*dt, y=da.y.values, z=da.z.values)
da_ext = xr.DataArray(np.concatenate([da_mean, da],1), dims=da.dims,coords=coords)
fs = get_flowSimluation(da=da_ext, offset=(50*dx,-200,0))
fs.visualize(10, view=XZView(y=0, x=np.linspace(-1000,1000)))
```



[ ]:

Previous Next

Edit on Gitlab launch binder

WindFarmFlowModel

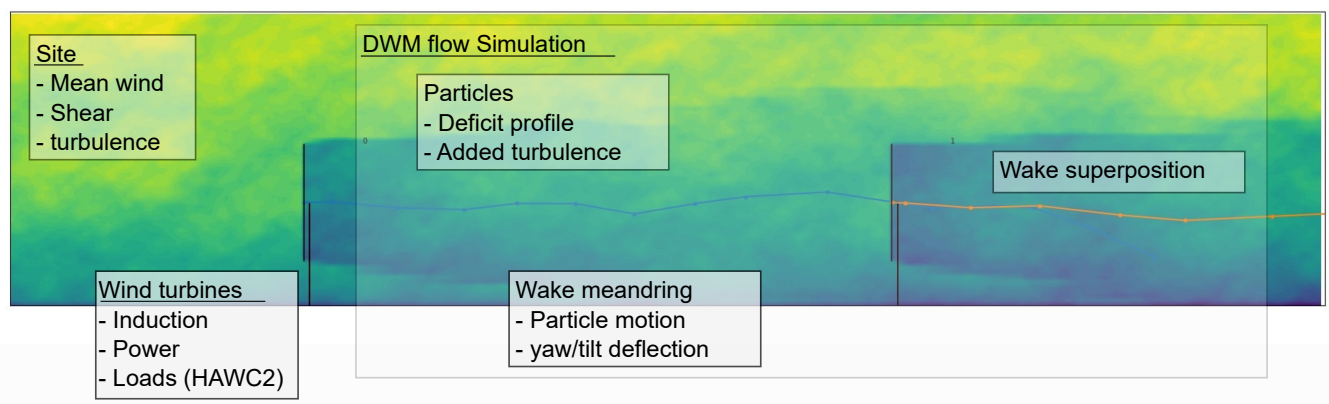
At the moment, only the `DWMFlowSimulation` has been implemented

Previous Next

Edit on Gitlab launch binder

DWM flow simulation

The Dynamic-Wake-Meandering Wind-farm-flow model, `DWMFlowSimulation`, is a particle-based flow model that uses a set of sub models to simulate the particles, their meandering, deficit and wake-induced added turbulence as well as wake superposition.



Each turbine, has an associated row of `n_particles` particles. These particles holds information and a function, provided by the `` `particleDeficitGenerator <#ParticleDeficitGenerator>`__ ``, which is able to calculate the deficit profile at downstream positions. Furthermore, they have access to the `addedTurbulenceModel`, which is able to calculate the wake-induced added turbulence.

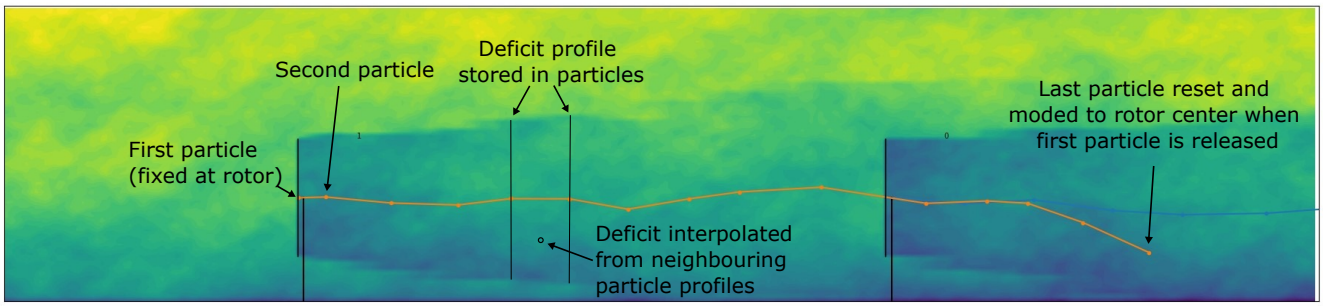
In each iteration, the time increments `dt` seconds and all particles, except the first, are moved (meandered) with the wind (mostly downstream, but also in the crosswind and vertical direction). The movement is simulated by the `ParticleMotionModel`.

The first particle is fixed to the rotor center and updated with the wind speed and information from the `` `WindTurbine <https://dynamiks.pages.windenergy.dtu.dk/Dynamiks/notebooks/WindTurbines.html>`__ ``, e.g. rotor position and induction/ct.

When the second particles has moved `d_particle` diameters downstream, the first particle is released, and the last particle is reset and fixed to the rotor center.

The wind speed at a given position is calculated in three steps:

1. Calculate the wake deficit from each wind turbine. To find the deficit from a wind turbine, its associated particles are searched to find the nearest up- and downstream particles. From the deficit profile of these two particles, the deficit value at the position is interpolated.
2. Having, the deficits from all upstream wind turbine, the sum of deficits is calculated by the `SuperpositionModel`, which defaults to linear sum.
3. Finally, the effective wind speed is calculated by subtracting the sum of deficits from the free-stream wind speed, which is provided by the `` `Site <https://dynamiks.pages.windenergy.dtu.dk/Dynamiks/notebooks/Site.html>`__ ``.



The API interface for `DWMFlowSimulation` is shown below

[1]:
```
from dynamiks.dwm import DWMFlowSimulation
help(DWMFlowSimulation.__init__)
```

```
Help on function __init__ in module dynamiks.dwm.dwm_flow_simulation:

__init__(self, site, windTurbines, particleDeficitGenerator, dt, particleMotionModel=<dynamiks.
    Parameters
    ----------
    site : Site
        site object, e.g. dynamiks.sites.TurbulenceFieldSite
        see https://dynamiks.pages.windenergy.dtu.dk/dynamiks/notebooks/Site.html
    windTurbines : WindTurbines
        windTurbines object e.g. dynamiks.wind_turbines.PyWakeWindTurbines, dynamiks.wind_turbi
        see https://dynamiks.pages.windenergy.dtu.dk/dynamiks/notebooks/WindTurbines.html
    particleDeficitGenerator : ParticleDeficitGenerator
        Model to instantiate the wake deficit model of new particles, e.g.
        dynamiks.dwm.PyWakeDeficitGenerator, dynamiks.dwm.jDWMAinslieGenerator
        see https://dynamiks.pages.windenergy.dtu.dk/dynamiks/notebooks/DWMFlowSimulation.html#
    dt : float
        Time increment in dynamiks (sub models, e.g. HAWC2, are allowed to have smaller increme
        Must be small enough to capture the interaction between flow model and wind turbines
    particleMotionModel : ParticleMotionModel, optional
        model to handle particles movement and velocity, e.g.
        dynamiks.particle_motion_models.ParticleMotionModel (default) or
        dynamiks.particle_motion_models.HillVortexParticleMotion
    d_particle : int or float, optional
        Distance between particles normalized with rotor diameter
        Default is 0.2D
        Must be small enough to capture non-linearities in wake and the particle meandering pat
```

```
    n_particles : int or None
        Number of particles associated with each turbine.
        If None, default, the number of particles is set to cover 120% of the farm size (at lea
    wind_direction : int or float, optional
        Wind direction used internally to rotate the wind farm (x is always the turbulence box
        Use the EastNorthView to see the wind from the right direction.
    step_handlers : list
        List of step handlers that is called in every time step.
        Step handlers can be:
        - step_handler, i.e. a function, f(flowSimulation)->None
        - (<time_start>, <time_step>, step_handler)
        - (<time_start>, <time_step>, [step_handler1, step_handler2, ...])
    superpositionModel : py_wake.SuperpositionModel, optional
        model used to sum up deficits from multiple wind turbines.
        Default model is LinearSum from py_wake.py_wake.superposition_models
    addedTurbulenceModel : AddedTurbulenceModel, optional
        AddedTurbulence model object
        Default is SynchronizedAutoScalingIsotropicMannTurbulence
    windTurbinesParticles : {WindTurbinesParticles, DistributedWindTurbinesParticles}, optional
        Default is WindTurbinesParticles
        Use DistributedWindTurbinesParticles to improve parallel performance
```

ParticleDeficitGenerator

Every time a particle is reset, the `WakeDeficitGenerator` models instantiates a `ParticleDeficitProfile` object with rotor size, rotor position, wind speed, induction and turbulence intensity and assigns it to the particle.

Current build-in options:

- jDWMAinslieGenerator
- PyWakeDeficitGenerator:

[2]:
```
# Setup site and wind turbines for examples below
import matplotlib.pyplot as plt
import numpy as np
from dynamiks.utils import doc_utils # use cached animations in sphinx documentation
from dynamiks.utils.test_utils import tfp, DemoSite, DemoWindTurbines
from dynamiks.dwm import DWMFlowSimulation
from dynamiks.views import Points, XView, XYView, YView
from py_wake.utils.plotting import setup_plot
from dynamiks.dwm.particle_deficit_profiles import jDWMAinslieGenerator, PyWakeDeficitGenerator
color_lst = ['#ff7f0e', '#2ca02c', '#d62728']

ws = 10 # mean wind speed
ti = 0.1 # turbulence intensity

# setup site
site = DemoSite(ws,ti)

# setup wind turbines
wts = DemoWindTurbines(x=[0], y=[0])
D = wts.diameter(0) # 80m
z_hub = wts.hub_height(0) # 70m
```

jDWMAinslieGenerator

The jDWMAinslieGenerator uses the Ainslie eddy viscosity model implemented in jdwm

[3]:
```
from dynamiks.dwm.particle_deficit_profiles.ainslie import jDWMAinslieGenerator
help(jDWMAinslieGenerator.__init__)
```

```
Help on function __init__ in module dynamiks.dwm.particle_deficit_profiles.ainslie:

__init__(self, boundaryConditionModel=<class 'jDWM.BoundaryCondition.madsen'>, viscosity_model=
    Parameters
    ----------
    boundaryConditionModel : jDWM.BoundaryConditionModel, optional
        boundary condition
    viscosity_model : jDWM.EddyViscosityModel, optional
        Eddy viscosity model, class or object (TI will be overriden, so it can be set to TI=1)
    solver : jDWM.Solver, optional
        solver
    scale_with_freestream : bool, optional
        Specify if normalized ainlie deficit should be scaled with freestream (i.e. without wak
        effective wind speed including wakes (default)
    r_max : int or float
        Width of axisymmetric control volume [R] (normalized by rotor radius)
    n_r : int
        Number of points in axisymmetric control volume
```

See jdwm for a list of boundary condition models and eddy viscosity models and solvers.

The solvers are described mathematically in https://wes.copernicus.org/articles/8/1387/2023/

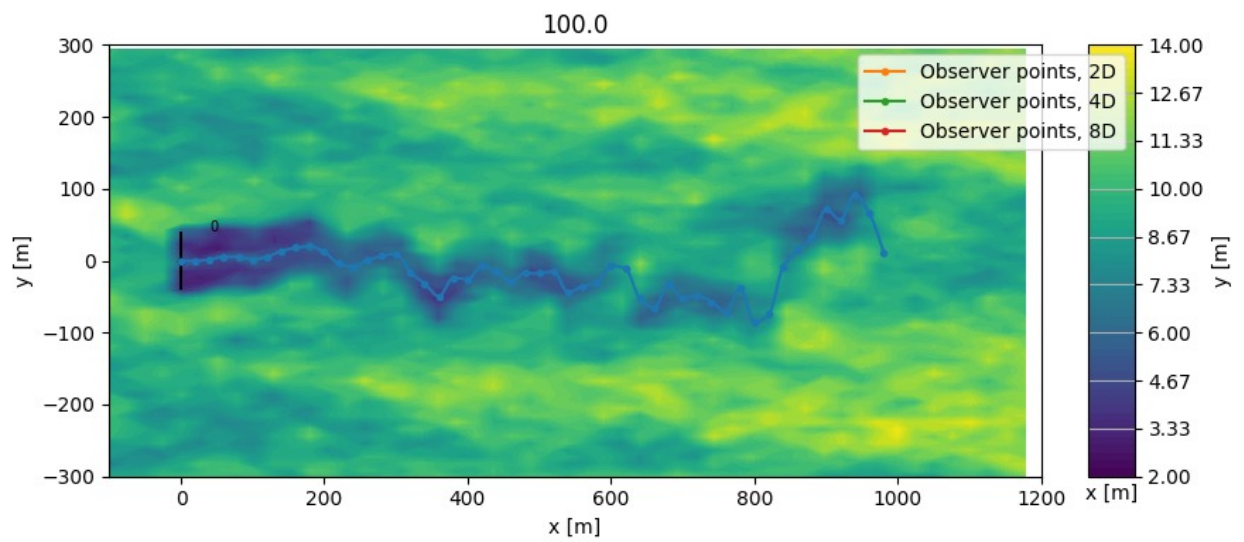
Note, `r_max` must be large enough to allow the deficit to mix with the surrondings. If set too low, the deficit will stop dissipation and continue infinitely downstream

[4]:
```
import jDWM.BoundaryCondition
import jDWM.EddyViscosityModel
import jDWM.Solvers

ainslie_wake = jDWMAinslieGenerator(boundaryConditionModel=jDWM.BoundaryCondition.madsen,
                                    viscosity_model=jDWM.EddyViscosityModel.madsen,
                                    solver=jDWM.Solvers.implicit(),
                                    scale_with_freestream=False, # scale deficit with local win
                                    r_max=3, # max width [R]
                                    n_r=51, # number of stations
                                    )
```

Example

[5]:
```
from dynamiks.utils.data_dumper import DataDumper
from dynamiks.visualizers import ParticleVisualizer
y_lst = np.linspace(-3*D, 3*D, 200) # observer points
d_lst = np.array([2,4,8])

# setup data dumper saving wind speed at observer points
xy_view = XYView(x=d_lst*D, z=z_hub, y=y_lst, adaptive=False)
```

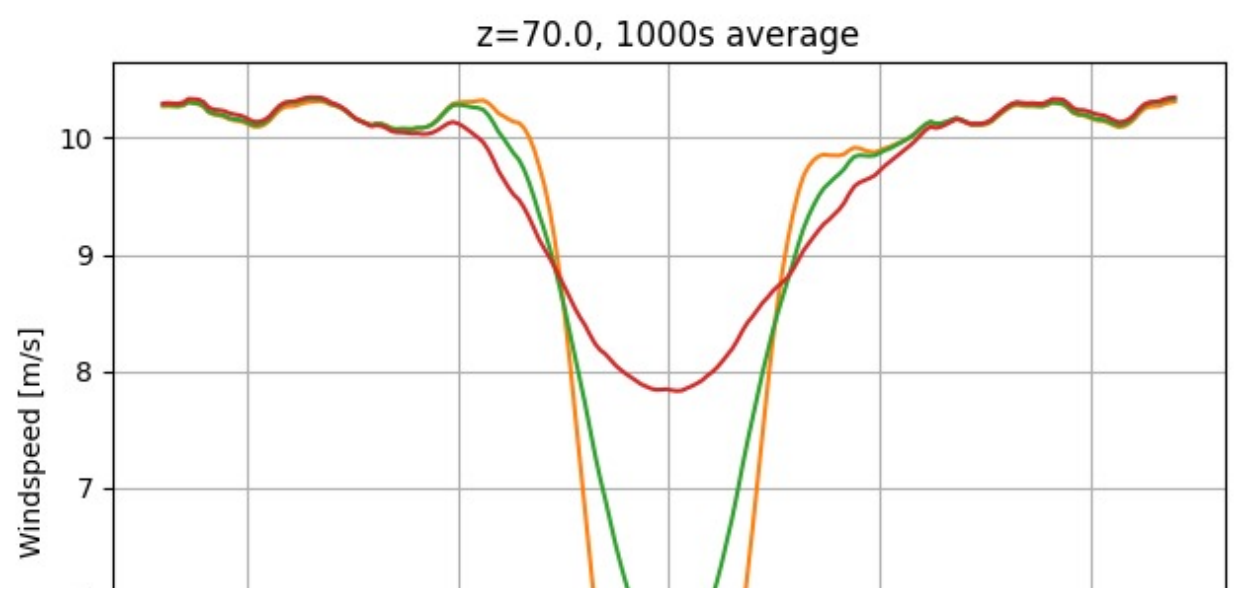
```
ws_dumper = DataDumper(lambda fs : fs.get_windspeed(xy_view, include_wakes=True),
                       coords={'uvw':['u','v','w'], 'D':d_lst, 'y':y_lst})


fs = DWMFlowSimulation(site,DemoWindTurbines(), ainslie_wake, 1, addedTurbulenceModel=None, n_p
                       step_handlers=[ws_dumper])

fs.run(100) # run 100s to propagate wake at beyond 8D

#plot situation
ax = plt.figure(figsize=(12,4)).gca()
fs.show(view=XYView(z=70,x=np.linspace(-100,1200), y=np.linspace(-300,300), ax=ax, visualizers=
for d, c in zip(d_lst, color_lst):
    plt.plot(y_lst*0+d*D,y_lst, '.-', color=c, label=f'Observer points, {d}D')
setup_plot(xlabel='x [m]', ylabel='y [m]')
```

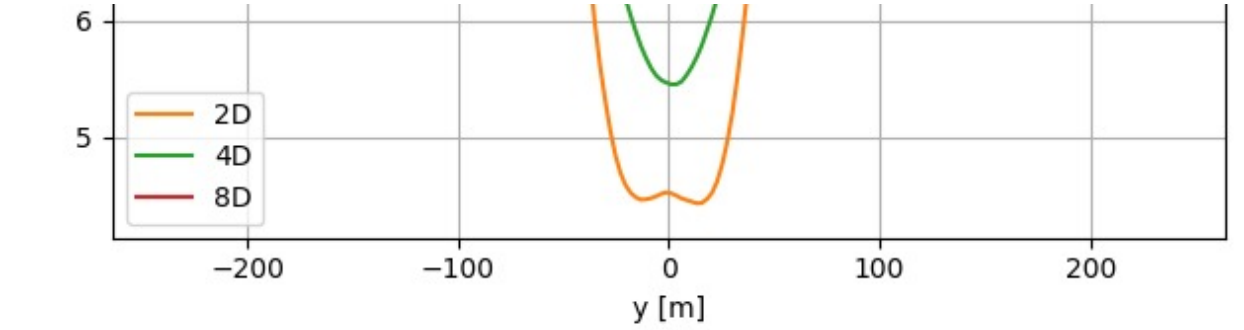


[6]:
```
fs.run(fs.time+1000, verbose=True)
```

100% 1000/1000 [00:10<00:00, 93.74it/s]

[7]:
```
data_array = ws_dumper.to_xarray()[100:] # exclude first 100s
for d, c in zip(data_array.D.values, color_lst):
    data_array.sel(uvw='u', D=d).mean('time').plot(color=c, label=f'{d}D')
setup_plot(ylabel='Windspeed [m/s]', xlabel='y [m]', title=f'z={z_hub}, 1000s average')
```





PyWakeDeficitGenerator

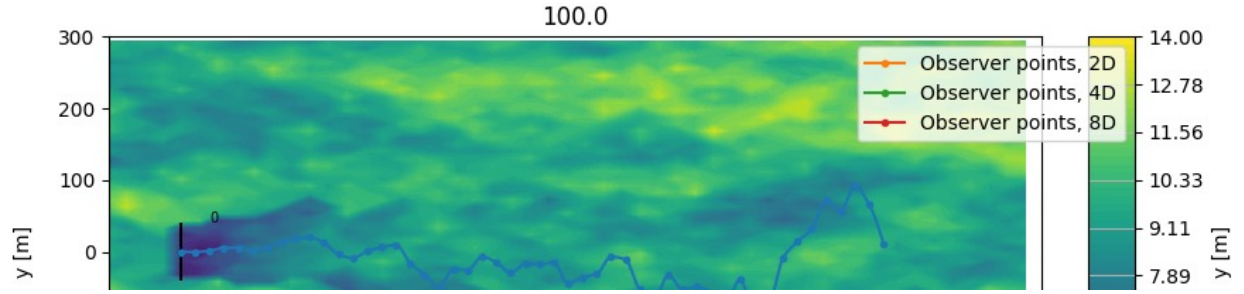
The `PyWakeDeficitGenerator` allows you to use any PyWake wake deficit models.

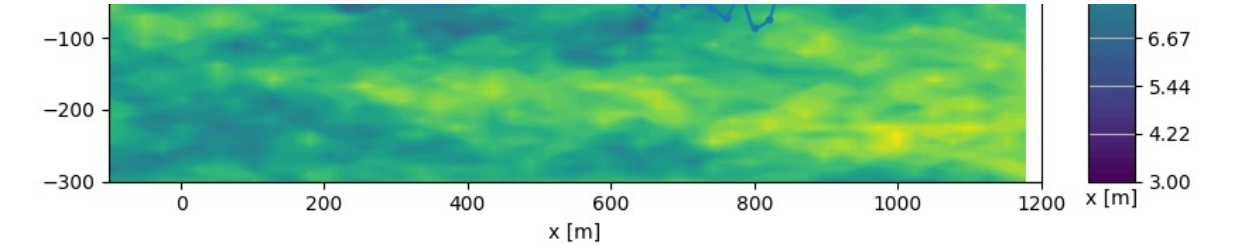
Note: Most PyWake models are static models calibrated to fit measurements or LES-CFD. This means that their empirical expansion includes the dynamic meandering, that Dynamiks puts on top of the static model. A recalibrated expansion model/factor is therefore needed.

[8]:
```
from py_wake.deficit_models import NOJDeficit
noj_wake = PyWakeDeficitGenerator(deficitModel=NOJDeficit(rotorAvgModel=None))
```

Example

[9]:
```
from dynamiks.utils.data_dumper import DataDumper
from dynamiks.visualizers import ParticleVisualizer
y_lst = np.linspace(-3*D, 3*D, 200) # observer points
d_lst = np.array([2,4,8])

# setup data dumper saving wind speed at observer points
xy_view = XYView(x=d_lst*D, z=z_hub, y=y_lst, adaptive=False)
ws_dumper = DataDumper(lambda fs : fs.get_windspeed(xy_view, include_wakes=True),
                       coords={'uvw':['u','v','w'], 'D':d_lst, 'y':y_lst})

fs = DWMFlowSimulation(site,DemoWindTurbines(), noj_wake, 1, addedTurbulenceModel=None, n_parti
                       step_handlers=[ws_dumper])

fs.run(100) # run 99s to propagate wake at beyond 8D

#plot situation
ax = plt.figure(figsize=(12,4)).gca()
fs.show(view=XYView(z=70,x=np.linspace(-100,1200), y=np.linspace(-300,300), ax=ax, visualizers=
for d, c in zip(d_lst, color_lst):
    plt.plot(y_lst*0+d*D,y_lst, '.-', color=c, label=f'Observer points, {d}D')
setup_plot(xlabel='x [m]', ylabel='y [m]')
```



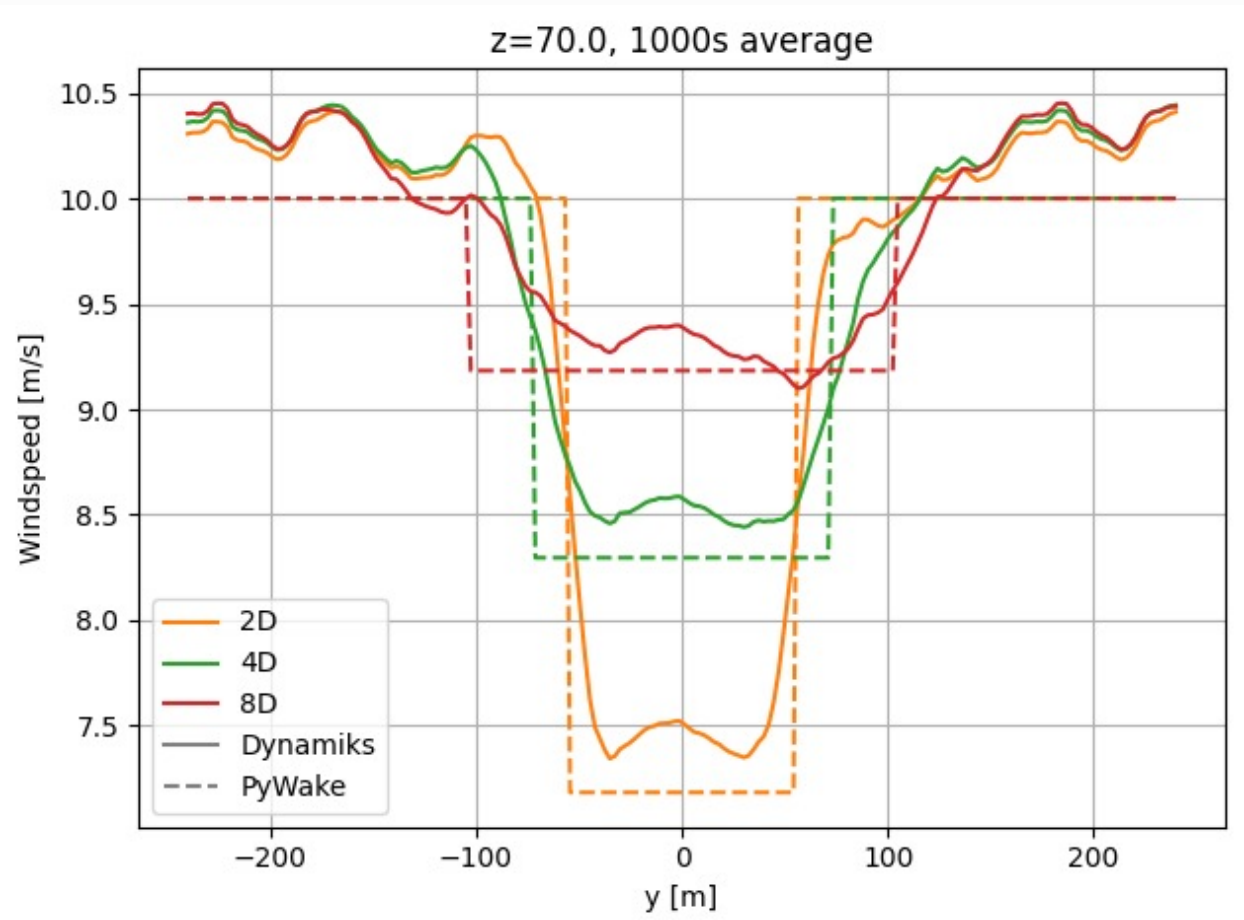


[10]:
```
fs.run(fs.time+1000, verbose=True)
```

100% 1000/1000 [00:04<00:00, 232.47it/s]

[11]:
```
# calculate the corresponding wind speed using PyWake (static)
from py_wake.deficit_models.noj import NOJDeficit
from py_wake.examples.data.hornsrev1 import V80
from py_wake.site._site import UniformSite
from py_wake.wind_farm_models.engineering_models import PropagateDownwind
from py_wake.flow_map import XYGrid
wfm = PropagateDownwind(UniformSite(), V80(), NOJDeficit())
pywake_data = wfm([0],[0],ws=10,wd=270).flow_map(XYGrid(x=d_lst*D, y=y_lst, h=z_hub))
```

[12]:
```
data_array = ws_dumper.to_xarray()

for d, c in zip(data_array.D.values, color_lst):
    data_array.sel(uvw='u', D=d).mean('time').plot(color=c, label=f'{d}D')
    pywake_data.WS_eff.sel(x=d*D).plot(color=c, ls='--')
plt.plot([],'-',color='gray', label='Dynamiks')
plt.plot([],'--',color='gray', label='PyWake')
setup_plot(ylabel='Windspeed [m/s]', xlabel='y [m]', title=f'z={z_hub}, 1000s average')
```



Particle motion models

The particle motion model is responsible for the movement of the particles. Currently, two models exists: ParticleMotionModel (default) and HillVortexParticleMotion

It is not evident, how to best model the particle motion and therefore, the models leave some options for the user on how the particles should be move in the x (downstream) as well as in the y/z (lateral/vertical) directions.

The downstream particle advection speed will affect how changes in the wake, e.g. caused by induction changes due to inflow changes and turbine control (derating, yaw, shut down/start up) will propagate downstream. It will, however, not affect the momentum conservation as the concentration of particles has no impact on the wake deficit.

The `ParticleMotionModel` optionally takes into account wakes from upstream wind turbines, but does not consider effects of its own induction.

The `HillVortexParticleMotion` model extends the `ParticleMotionModel` with movement due to self induction, which means that the particles will deflect in case of yaw/tilt misalignment.

ParticleMotionModel

The API interface to the `ParticleMotionModel` is show below

[13]:
```
from dynamiks.dwm.particle_motion_models import ParticleMotionModel
help(ParticleMotionModel.__init__)
```

```
Help on function __init__ in module dynamiks.dwm.particle_motion_models:

__init__(self, x_speed=<XSpeed.Global: 1>, temporal_filter=<dynamiks.dwm.particle_motion_models
    Models the motion (position and velocity) of the particles

    The velocity in the y and z direction are implicitly set to 'Particle'
    (i.e. the v and w speed at the current particle position)

    Note
    - The time ('current' or 'moment when the particle was released') may reflect
    a low pass filtered average (see the temporal_filter argument)
    - The position ('Rotor' or 'Particle') may represent a spatial average (see the spatial_fil
    - The temporal and spatial filter models the same effect in different ways and probably onl

    parameters
    ---------
    x_speed : SpeedType
        Particle advection speed along x axis (downstream)
        - Global: advection speed = turbulence transport speed
        - Rotor: advection speed = wind speed (U) at the rotor position at the moment
        when the particle was emitted
        - Particle: advection speed = current wind speed (U) at the current particle position
    temporal_filter : float, function or ndarray, optional
        The arguments for the temporal filter that is applied to the wind speed to obtain the p
        if float: cut-off frequency for a temporal first order low pass filter
        if function: function, f(U,D)->float, returning the cut-off freqency for a temporal fir
```

```
        (e.g. CutOffFrqLarsen2008 or CutOffFrqLio2021(default)), where U is the turbulence tran
        D is the largest rotor diameter.
        If ndarray: Array of second-order filter coefficients, must have shape (n_sections, 6).
        Each row corresponds to a second-order section, with the first three columns providing
        coefficients and the last three providing the denominator coefficients.
    spatial_filter : NodeRotorAvgModel (from py_wake.rotor_avg_models.rotor_avg_model) or None,
        If NodeRotorAvgModel: The particle speed is set to a spatial (weighted) average of the
        a set of nodes (scaled with  before applying the temporal low-pass filter.
        Examples are PyWake NodeRotorAvgModel, e.g. CGIRotorAvg, see
        https://topfarm.pages.windenergy.dtu.dk/PyWake/notebooks/RotorAverageModels.html)
        If None (default), the wind speed at the rotor center or current particle position
        is used without spatial average
    include_wakes : bool, optional
        Include wake from upstream wind turbines in the wind speed used to obtain the particle
```

XSpeed

The `x_speed` argument determines the downstream speed of the particles. It can be set to either:

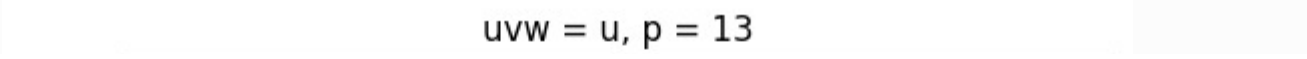
- `XSpeed.Global`: All particles advect downstream with constant turbulence transport speed (`temporal_filter`, `spatial_fitler` and `include_wakes` are disregarded)
- `XSpeed.Rotor`: Each particle advects with constant speed, but the speed is determined as the wind speed at the rotor when the particle is released.
- `XSpeed.Particle`: Each particle advects with the local wind speed

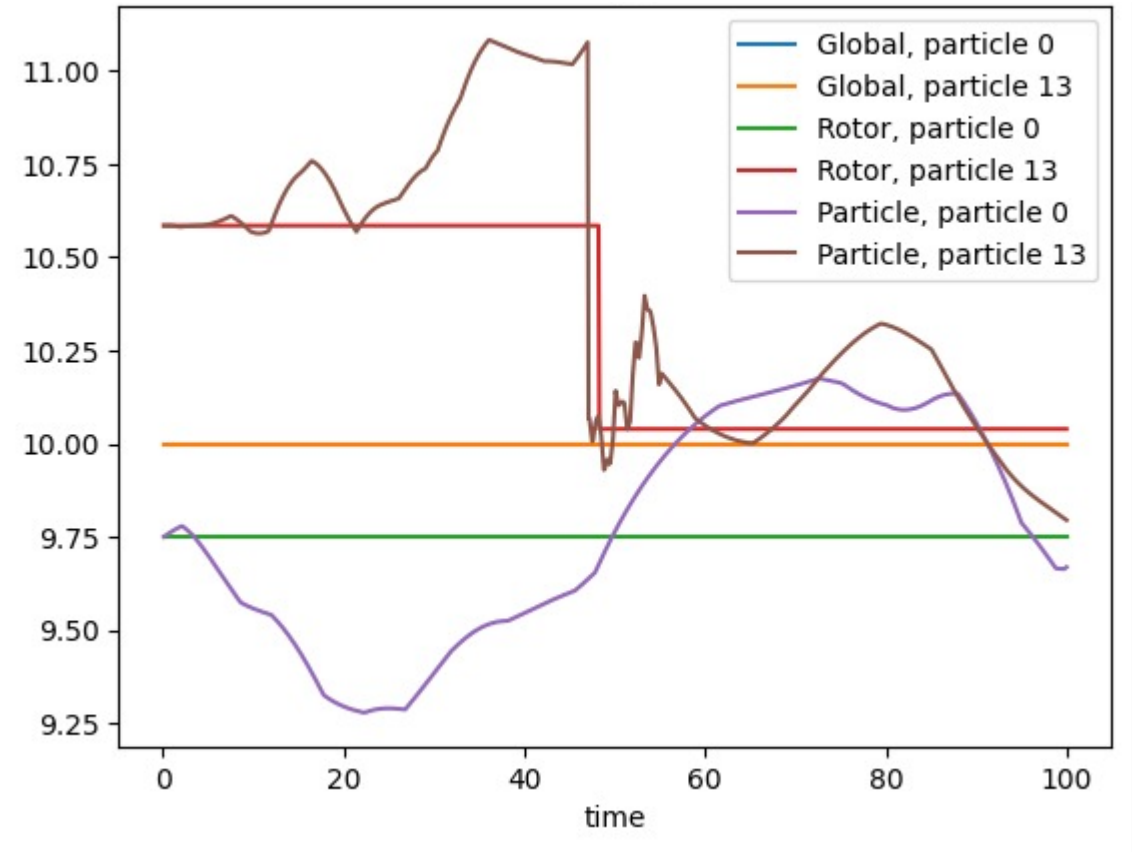
When using `XSpeed.Rotor` and `XSpeed.Particle`, consider using a Temporal filter and/or Spatial filter. In addition, the `include_wakes` argument should probably be set to `True`

Example showing the effect of XSpeed

In this example the downstream advection speed of two different particles are plotted for the three different XSpeed options

[14]:
```
from dynamiks.utils.test_utils import DefaultDWMFlowSimulation
from dynamiks.dwm.particle_motion_models import ParticleMotionModel, XSpeed
from dynamiks.utils.data_dumper import DataDumper
import matplotlib.pyplot as plt
import numpy as np
for x_speed in [XSpeed.Global, XSpeed.Rotor, XSpeed.Particle]:
    data_dumper = DataDumper(lambda fs:fs.particle_velocity_uip[:,0], coords={'uvw':['u','v','w
    particleMotionModel=ParticleMotionModel(x_speed=x_speed, temporal_filter=None)
    fs = DefaultDWMFlowSimulation(ti=0.05, d_particle=1, n_particles=20, dt=0.1, particleMotior
    fs.run(100)
    da = data_dumper.to_xarray()
    for p in [0,13]:
        da.sel(uvw='u', p=p).plot(label=f'{x_speed.name}, particle {p}')

plt.legend();
```





The `Global` option results in both particles being moved with the turbulence transport speed, i.e. 10 m/s in this case.

The `Rotor` option results in particle 0 begin moved with 9.75 m/s. Particle 13 starts with a speed of 10.6 m/s, but after around 50s, it is reused as the new boundary particle, and therefore moved back to the rotor and its speed is reset to the current rotor speed.

The `Particle` option results in a time-varying speed. Also in this case, a significat step is seen when the particle is moved from far-downstream to the rotor to become the new boundary particle.

Temporal filters

The wake deficit (i.e. the particles in this context) is assumed to be advected downstream with the current, local wind speed. How “current” and how “local” that is, is, however, difficult to quantify.

The `ParticleMotionModel` applies an optional temporal filter to the instant wind speed at the particle position. Note, that the Spatial filter is similar, it may be possible to replaced the spatial filter by lowering temporal cut-off frequency.

Larsen (2008), argues that the temporal cut-off frequency should be $U/(2D)$, as shorter wave lengths will not be able to move a 1D wide wake same direction. Lio (2021) investigated the meandering using lidar data and found a cut-off frequency of $U/(4D)$ to be more appropriate.

References:

- Larsen, G. C., Madsen, H. A., Thomsen, K., and Larsen, T. J. Wake meandering: a pragmatic approach, Wind Energy, 11, 377–395, https://doi.org/10.1002/we.267, 2008.
- Lio, W. H., Larsen, G. C., and Thorsen, G. R.: Dynamic wake tracking using a cost-effective LiDAR and Kalman filtering: Design, simulation and full-scale validation, Renew Energ., 172, 1073–1086, 2021.

Examples

[15]:
```
from dynamiks.dwm.particle_motion_models import CutOffFrqLarsen2008, CutOffFrqLio2021, CutOffFr
from scipy import signal
temporal_filter = .1 # cut-off frequency = 0.1 Hz
temporal_filter = CutOffFrqLarsen2008 # cut-off frequency = U/(2*D) Hz
temporal_filter = CutOffFrqLio2021 # cut-off frequency = U/(16*D) Hz
temporal_fitler = CutOffFrq(d) # cut-off frequency = U/(d*D) Hz

# N'th order Butterworth low pass filter with cut-off frequency(Wn) = 0.1 Hz
dt = 1 # time step
temporal_filter = signal.butter(N=1, Wn=.1, btype='low', fs=1 / dt, output='sos')
```

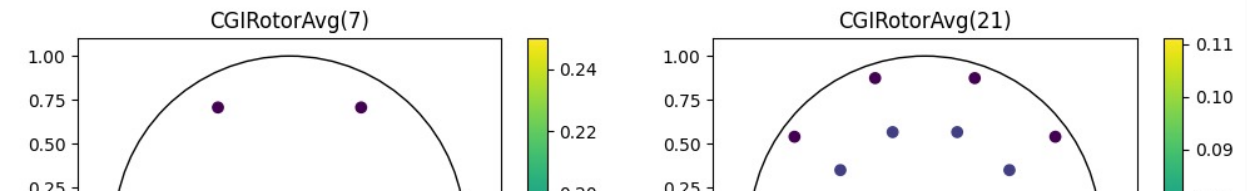
Spatial filters

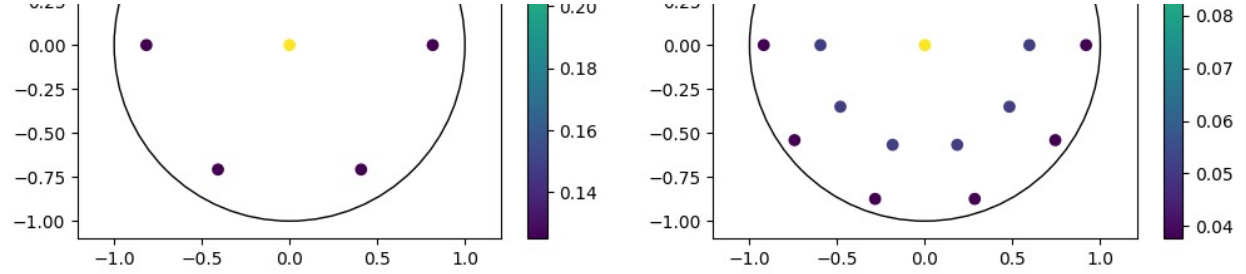
The easiest way to specify the spatial filter is via a PyWake `RotorAvgModel`, see https://topfarm.pages.windenergy.dtu.dk/PyWake/notebooks/RotorAverageModels.html.

These models provides a number of normalized node positions and associated weights. The most relevant are problably the `CGIRotorAvg` model with either `7` or `21` nodes, see below.

The position of the nodes are scaled with the wake width, which is typically 2σ for gaussian profiles.

[16]:
```
from py_wake.rotor_avg_models import CGIRotorAvg
axes = plt.subplots(1,2, figsize=(12,4))[1]
for ax, (name,sf) in zip(axes, [('CGIRotorAvg(7)', CGIRotorAvg(7)),
                                ('CGIRotorAvg(21)',CGIRotorAvg(21))]):

    c = ax.scatter(sf.nodes_x, sf.nodes_y, c=sf.nodes_weight)
    ax.add_patch(plt.Circle((0,0),1,fill=False))
    plt.colorbar(c)
    ax.axis('equal')
    ax.set_title(name)
```



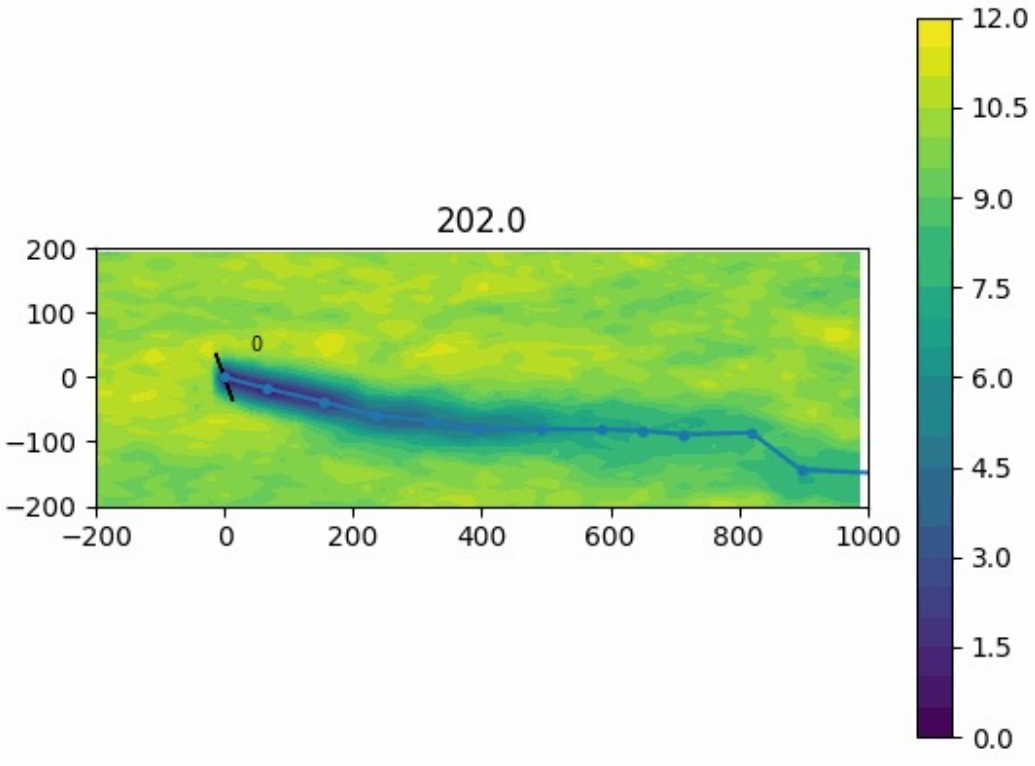


HillVortexParticleMotion

The `HillVortexParticleMotion` model extends the default `ParticleMotionModel` with a model of the self induction of the emitting wind turbine. The self induction is modeled using Hill Vortex theory. The self-induction is perpendicular to the rotor, which means that yaw and tilt misalignment make the particles deflect

[17]:
```
from dynamiks.dwm.particle_motion_models import HillVortexParticleMotion
from dynamiks.utils.test_utils import DefaultDWMFlowSimulation
from dynamiks.visualizers import ParticleVisualizer
import numpy as np
from dynamiks.views import XYView

fs = DefaultDWMFlowSimulation(ti=0.05, d_particle=1, n_particles=20, particleMotionModel=HillVc
fs.windTurbines.yaw = 20
fs.run(200)

fs.visualize(fs.time + 50, view=XYView(x=np.linspace(-200, 1000), y=np.linspace(-200, 200), z=7
             id='HillVortexParticleMotion')
```

[17]:

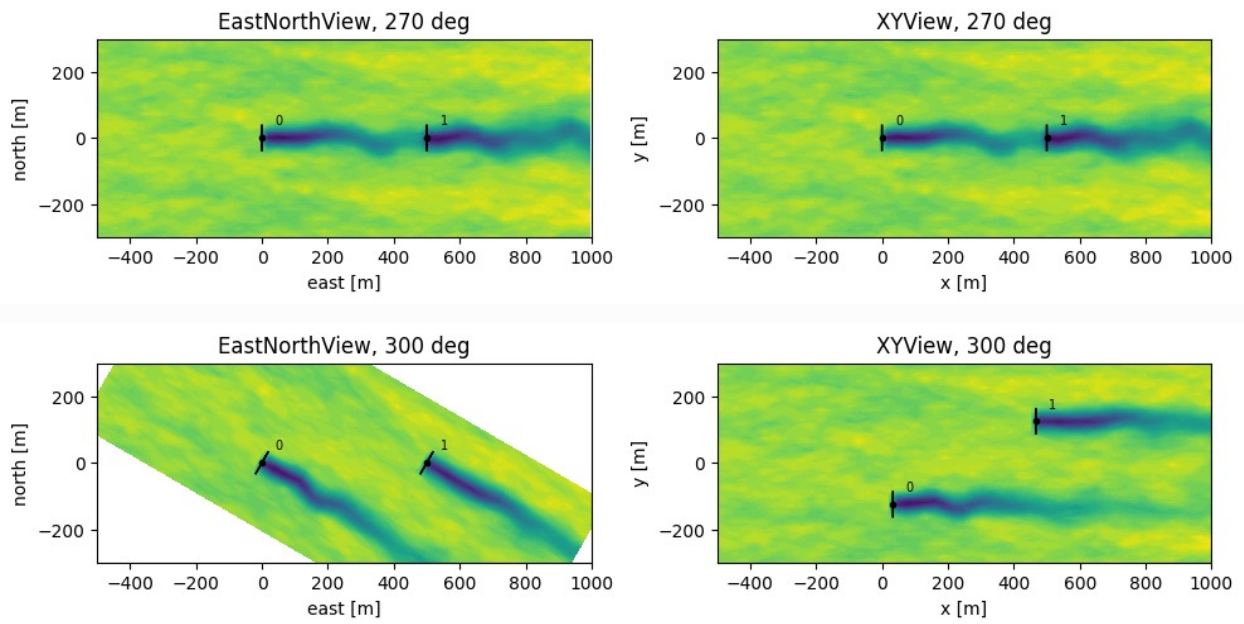


Wind direction

The `wind_direction` argument can be used simulate different wind directions. The default direction si 270 deg, where the `XYView` equals the `EastNorthView`. Internally in Dynamiks, the wind farm is rotated instead of the wind such that the turbulence transport direction is always along the x axis.

Example

[18]:
```
import numpy as np
import matplotlib.pyplot as plt
from dynamiks.utils.test_utils import DefaultDWMFlowSimulation
from dynamiks.views import XYView, EastNorthView, MultiView
from dynamiks.visualizers.flow_visualizers import Flow2DVisualizer
from py_wake.utils.plotting import setup_plot

for wd in [270,300]:
    ax2,ax1 = plt.subplots(1,2, figsize=(10,4))[1]
    fs = DefaultDWMFlowSimulation(x=[0,500],y=[0,0], ti=0.05, d_particle=1, n_particles=20, wir
    fs.run(100)
    fs.show(MultiView([
        XYView(x=np.linspace(-500,1000),y=np.linspace(-300,300),z=70, ax=ax1, flowVisualizer=Fl
               title=f'XYView, {wd} deg', xlabel='x [m]', ylabel='y [m]'),
        EastNorthView(x=np.linspace(-500,1000),y=np.linspace(-300,300),z=70, ax=ax2, flowVisual
                      title=f'EastNorthView, {wd} deg', xlabel='east [m]', ylabel='north [m]')]
           block=False)
```



Previous Next

Edit on Gitlab launch binder

Visualization

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
from dynamiks.utils.test_utils import DefaultDWMFlowSimulation

fs = DefaultDWMFlowSimulation(x=[0,500],y=[0,0], ti=0.05, d_particle=1, n_particles=20)
fs.run(100)
```

The flow simulation object has three visualization methods:

- show: Plot the current state
- visualize: Animate and show the simulation
- animate: Animate and save the animation to a file

All three methods takes a `view` (see documentation) as optional input.

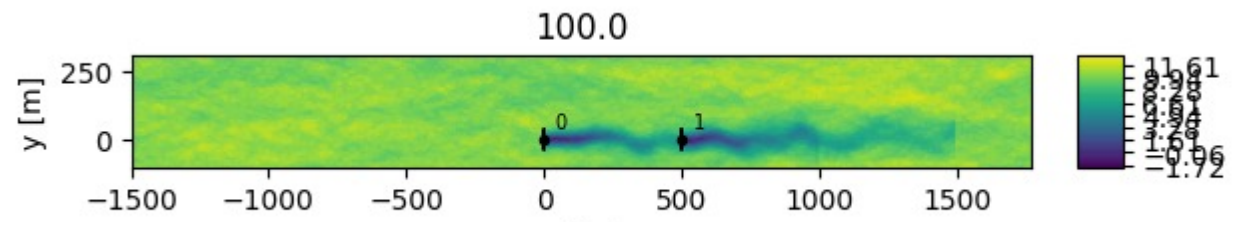
The `View` defaults to the default `XYView` (all turbulence (x,y)-grid points at average hub height) with a `Flow2DVisualizer` (see documentation) visualizing the u component of the flow

Visualization methods

Show

The `show(view=None, block=True)` method plots the current state. `block` is passed to `` `pyplot.show `` <https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.show.html>`__

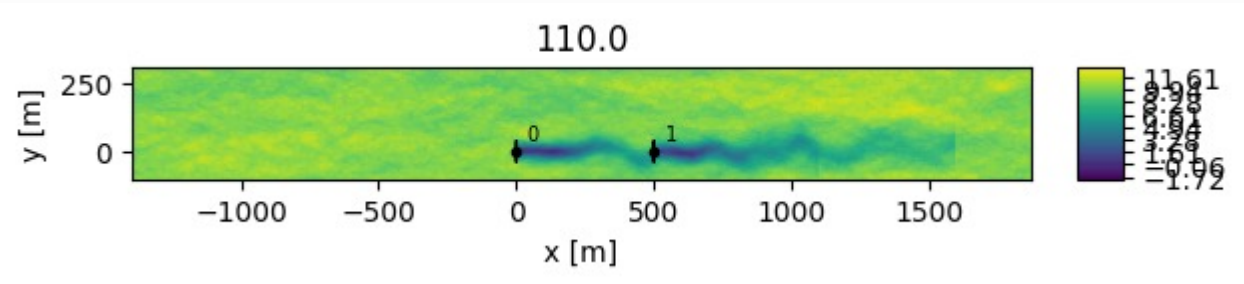
[2]:
```
fs.show()
```



Visualize

`visualize(time_stop, view=None, dt=None)` runs the simulation until time `time_stop` and visualizes the state for every `dt` seconds. If `dt=None` every time step is visualized.

[3]:
```
fs.visualize(fs.time+10)
```

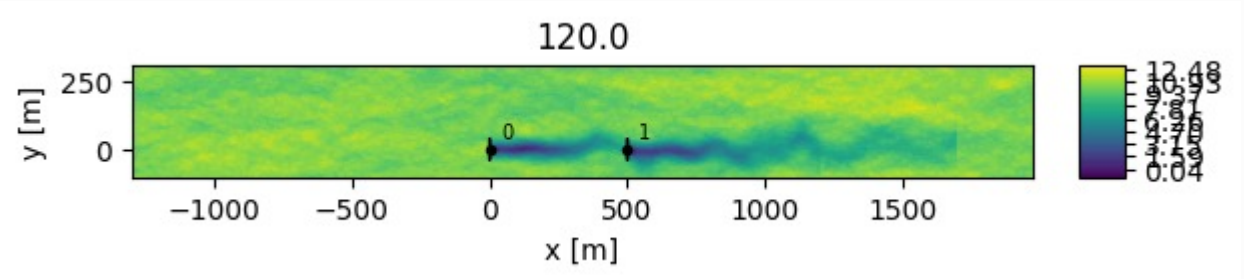


Animate

`animate(time_stop, filename, view=None, dt=None, interval=None)` runs the simulation until time `time_stop` and saves the animation to `filename`. A frame is saved for every `dt` seconds (If dt=None every time step is visualized) and `interval` specifies the time between each frame (in ms) in the playback. If `interval=None` the animation move plays in real time, while `interval<dt` results in faster playback

[4]:
```
# save animation
ani = fs.animate(fs.time+10, filename='visualization.gif');
```

100% 10/10 [00:05<00:00, 1.64it/s]



[5]:
```
# show saved gif animation
from IPython.display import HTML
HTML('<img src="visualization.gif">')
```

[5]:

Previous Next

Edit on Gitlab launch binder

Views

Views can be used to specify a point, line (1D views), plane (2D Views), volume (3D Views) in the domain to visualize or extract data from..

The parent class, (View) does nothing, but can be used for custom plots.

Multiple views can be combined into a MultiView object that shows, visualizes or animates multiple views simultaneously.

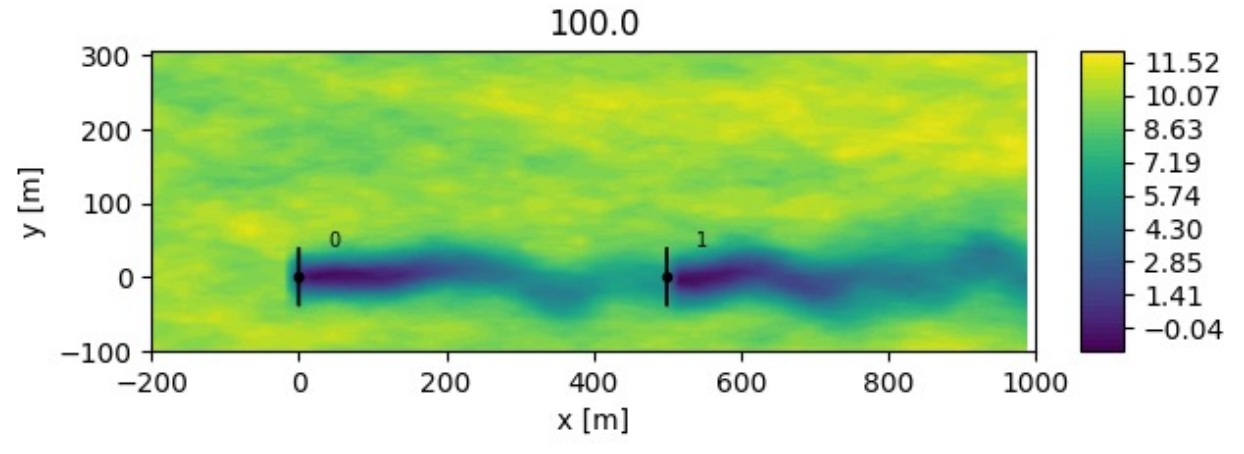
1D and 2D views take a number of inputs, e.g.:

- `flowVisualizer <>`__: plots the flow of the view. Added by default for 1D views and 2D Views
- `visualizers <>`__: List of additional plot functions, e.g. WindDirectionVisualizer, ParticleVisualizer or custom functions
- adaptive: controls whether the view should align with the nearest turbulence grid points or interpolate to the exact specified points.

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
from dynamiks.utils.test_utils import DefaultDWMFlowSimulation
from dynamiks.views import View, Points, XView, YView, ZView, XYView, XZView, YZView, EastNorth
from dynamiks.visualizers.flow_visualizers import Flow2DVisualizer
from dynamiks.visualizers import Visualizer, WindDirectionVisualizer, ParticleVisualizer, WindT

fs = DefaultDWMFlowSimulation(x=[0,500],y=[0,0], ti=0.05, d_particle=1, n_particles=20)
fs.run(100)
```

Visualization example

[2]:
```
view=XYView(x=np.linspace(-200,1000),z=70)
fs.show(view=view) # alternatively fs.visualizer(time_stop, view) or fs.animate(time_stop, view
```



Data extraction example

[3]:
```
view = XYView(x=np.linspace(-200,1000, 5), y=np.linspace(-200,200,3),z=70, adaptive=False)
fs.get_windspeed(xyz=view, include_wakes=True, xarray=True).sel(uvw='u')
```

[3]:

xarray.DataArray (**x**: 5, **y**: 3)

```
array([[10.52108134, 10.27947355,  9.77004316],
       [ 9.21008304,  0.04640486,  9.24646216],
       [10.70446799,  6.0073502 , 10.43323905],
       [10.78502925,  2.67699093, 10.97093775],
       [10.22901104,  6.03169402, 10.60620095]])
```

▼ Coordinates:

uvw	()	<U1	'u'
x	(x)	float64	-200.0 100.0 400.0 700.0 1e+03
y	(y)	float64	-200.0 0.0 200.0

▶ Indexes: (2)

▼ Attributes:

z:	70

Points

Points takes lists of `x`, `y` and `z` coordinates of the point of interest as input. The lists must have the same length.

[4]:
```
view = Points(x=[0,100,150], y=[50,75,100], z=[70,80,90])
fs.get_windspeed(xyz=view, include_wakes=True, xarray=True)
```

[4]:

xarray.DataArray (**uvw**: 3, **i**: 3)

```
array([[ 9.90302232e+00,  9.75718356e+00,  1.02187086e+01],
       [ 3.31847088e-01, -1.54851118e-02, -2.87045445e-01],
       [-5.55169456e-02,  1.16946489e-04, -8.39394000e-02]])
```

▼ Coordinates:

uvw	(uvw) <U1 'u' 'v' 'w'	
i	(i) int64 0 1 2	

▶ Indexes: (2)

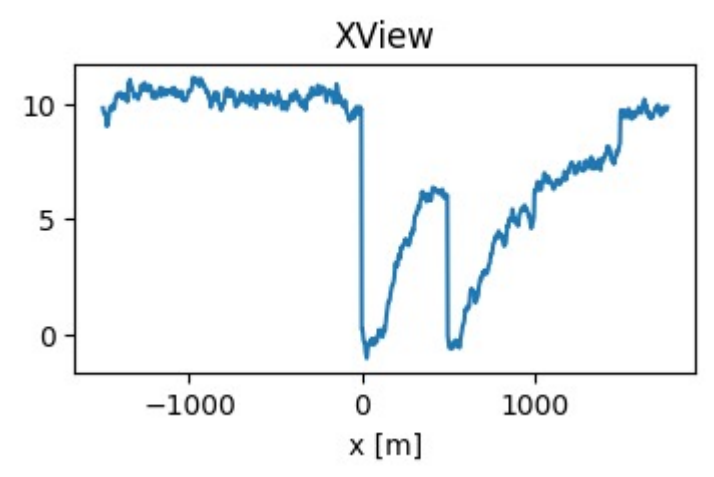
▼ Attributes:

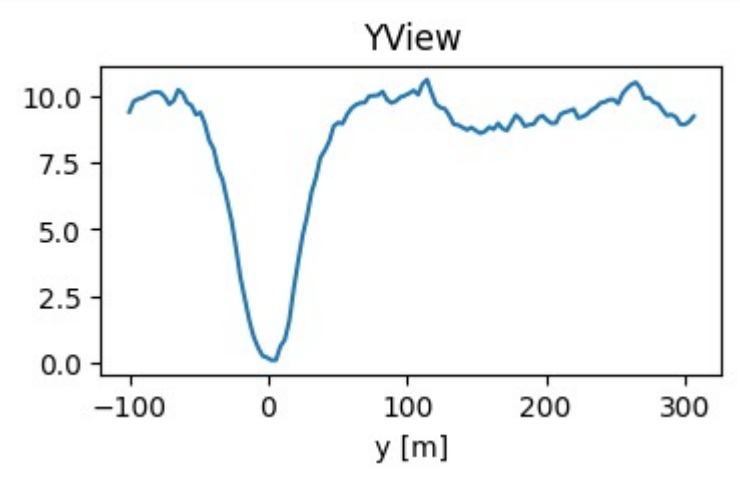
x :	[0 100 150]
y :	[50 75 100]
z :	[70 80 90]

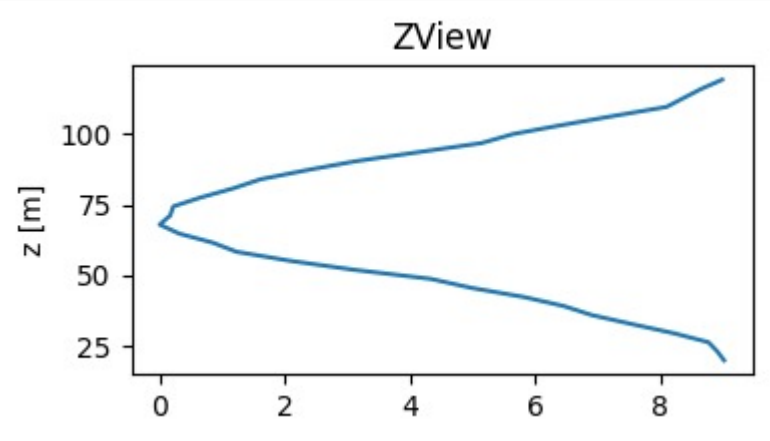
1D Views

The 1D views, `XView`, `YView` and `ZView` represent a line in the domain in the x,y and z direction respectively.

[5]:
```
from dynamiks.views import XView, YView, ZView
for view in [XView(x=None, y=0, z=70, title='XView'),
                 YView(x=100, y=None, z=70, title='YView'),
                 ZView(x=100, y=0, z=None, title='ZView')]:
    plt.figure(figsize=(4,2))
    fs.show(view)
```





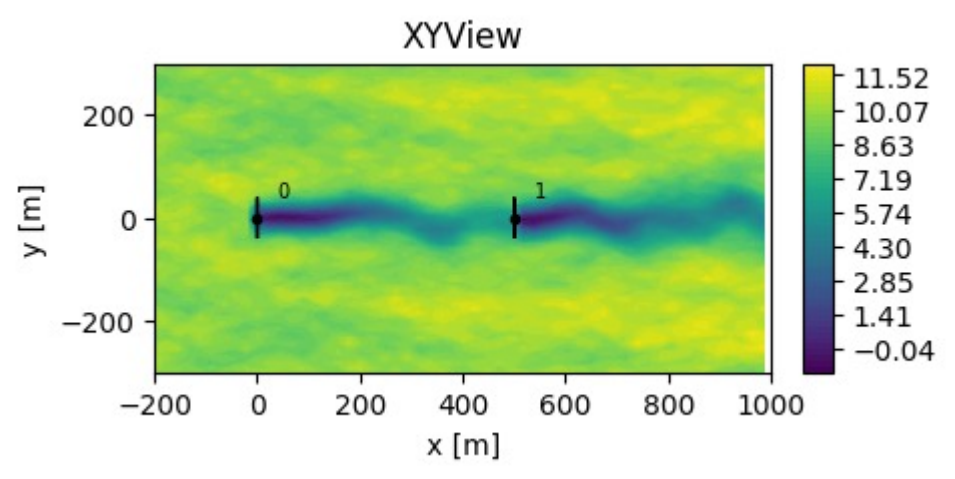


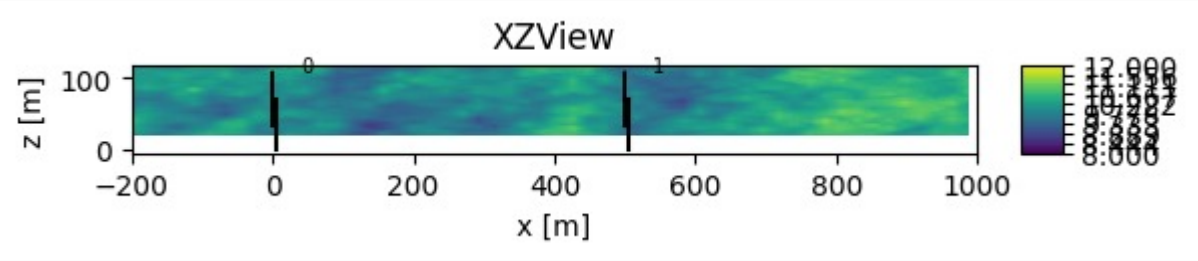
2D Views

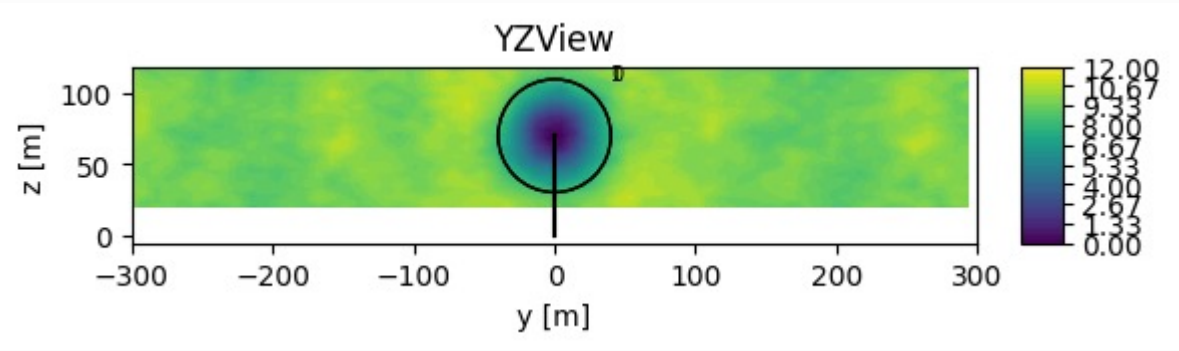
The 2D views, `XYView`, `XZView` and `YZView` represent a plane in the domain.

Note, it is possible to set the `z` argument of `XYView` to `None`, in which case the mean hub height is used.

[6]:
```
for view in [XYView(z=None, x=np.linspace(-200,1000), y=np.linspace(-300,300), title='XYView'),
             XZView(x=np.linspace(-200,1000), y=-125, title='XZView'),
             YZView(x=100, y=np.linspace(-300,300), title='YZView')]:
    plt.figure(figsize=(6,2))
    fs.show(view=view)
```

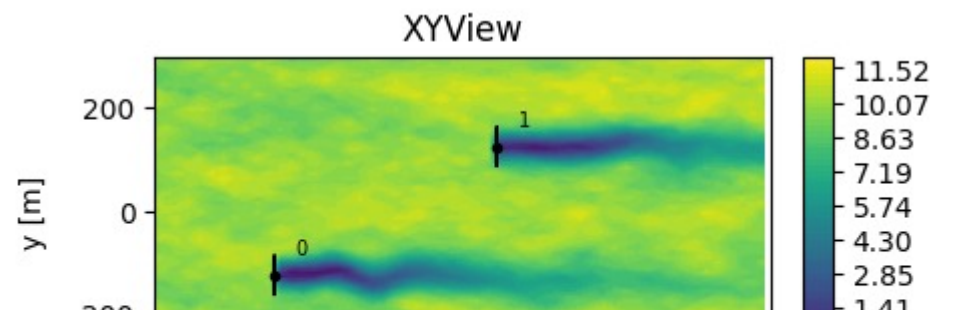


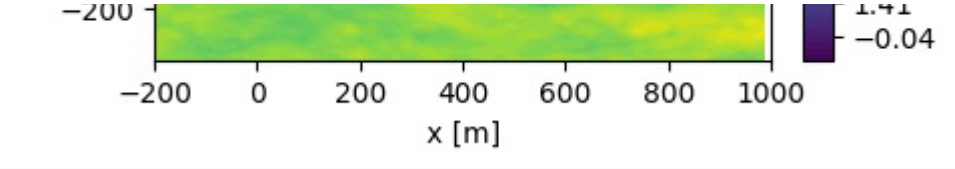


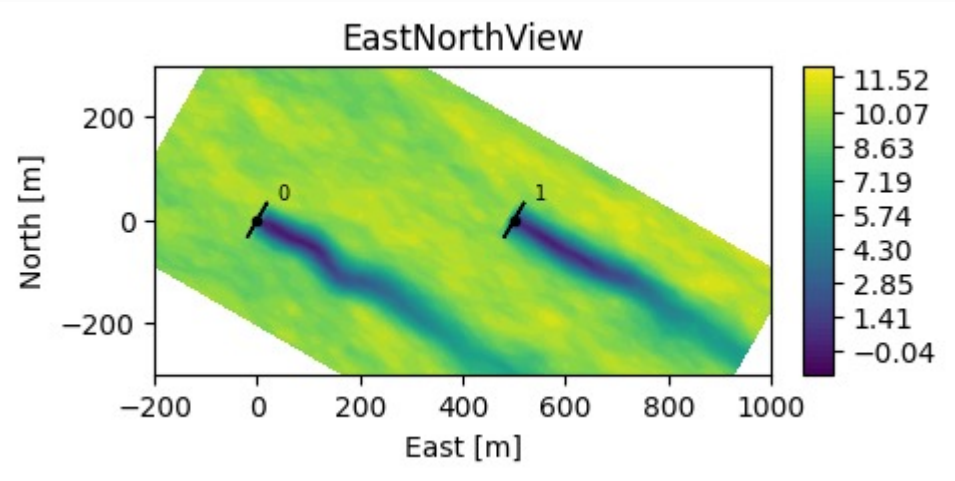


In Dynamiks, the wind direction is aligned with the x axis. This means that the farm is rotated when simulating other wind directions than 270deg. In this case the `EastNorthView` may be more appropriate

[7]:
```
fs300 = DefaultDWMFlowSimulation(x=[0,500],y=[0,0], ti=0.05, d_particle=1, n_particles=20, winc
fs300.run(100)

for view in [XYView(x=np.linspace(-200,1000), y=np.linspace(-300,300), z=70, title='XYView'),
             EastNorthView(x=np.linspace(-200,1000), y=np.linspace(-300,300), z=70, title='East
             ]:
    plt.figure(figsize=(6,2))
    fs300.show(view=view)
```







3D Views

The 3D view, `XYZView` represent a volume in the domain.

[8]:
```
from dynamiks.views import XYZView
view = XYZView(x=np.linspace(0,1000, 2), y=np.linspace(-200,200,3), z=np.linspace(0,200,4), ada
fs.get_windspeed(view, include_wakes=True, xarray=True).sel(uvw='u')
```

[8]:

xarray.DataArray (**x**: 2, **y**: 3, **z**: 4)

```
array([[[ 9.26077178,  9.49693096,  9.45151344,  9.14770068],
        [ 9.72400682,  0.18019037,  9.62042874,  9.62796699],
        [ 9.40774063,  9.09099916,  9.47601453,  9.2423559 ]],

       [[11.18759792, 10.23484625, 11.04388082, 11.13318111],
        [ 9.37567311,  6.02109833,  9.74339093,  9.8227184 ],
        [11.35826938, 10.59592663, 11.30597151, 10.99513351]]])
```

▼ Coordinates:

uvw	()	<U1	'u'
x	(x)	float64	0.0 1e+03
y	(y)	float64	-200.0 0.0 200.0
z	(z)	float64	0.0 66.67 133.3 200.0

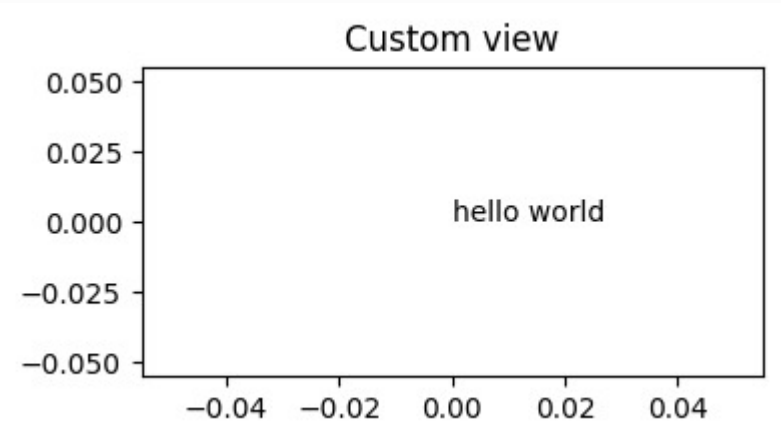
▶ Indexes: (3)

▶ Attributes: (0)

View

All views inherrits from the `View` super class, which does not draw anything by default, but it takes the `` `visualizers `` <#Visualizers>`__ input, which allows specification of custom functions

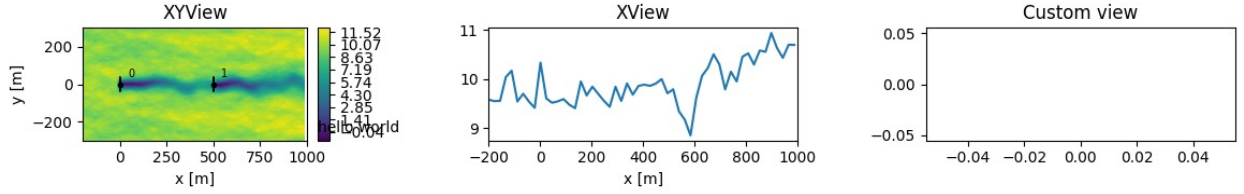
[9]:
```
plt.figure(figsize=(4,2))
fs.show(View(visualizers=[lambda fs: plt.annotate("hello world",(0,0))], title='Custom view'))
```



MultiView

`MultiView` allow `show`, `visualize` and `animate` to generate and display multiple view simulatenously, e.g. on subplot axes

[10]:
```
axes = plt.subplots(1,3, figsize=(12,2))[1]
view = MultiView([XYView(x=np.linspace(-200,1000), y=np.linspace(-300,300), z=70, title='XYView
                  XView(x=np.linspace(-200,1000), y=-125, z=70, title='XView', ax=axes[1]),
                  View(visualizers=[lambda fs: plt.annotate("hello world",(0,0))], title='Custo
fs.show(view=view)
```



View arguments

Visualizers

The visualizer argument to views allow specification of a list of additional predefined or custom plot functions.

Some predefined visualizers are:

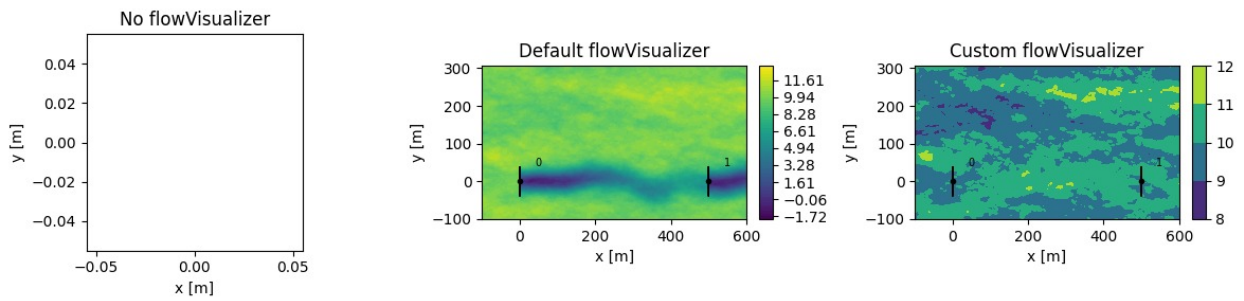
- `FlowVisualizer` (added to 1 and 2D views by default)
- `WindTurbineVisualizer` (added to 2D views by default)
- `ParticleVisualizer`
- `WindDirectionVisualizer`, see below

FlowVisualizer

1D and 2D views take the `flowVisualizer` input argument, which defaults to `None`. The argument enables three options:

- `None` (default): A `Flow1DVisualizer` or `Flow2DVisualizer` is added to the `visualizers` list
- `False`: The flow is not plotted
- `Flow1DVisualizer` or `Flow2DVisualizer`: Allows more options:
  - `uvw`: flow component to plot, defaults to `u`, combinations, e.g. `uv` is supported
  - `include_wakes`: include wakes, default is `True`
  - `levels`: color levels (`Flow2DVisualizer` only). Number of colorlevels or list of level values. Default is 55 levels with limits dynamically adapting to the min/max observed values.
  - `colorbar`: include colorbar (`Flow2DVisualizer` only). Default is True

[11]:
```
axes = plt.subplots(1,3, figsize=(12,3))[1]

view = MultiView([XYView(z=None, ax=axes[0], flowVisualizer=False, title='No flowVisualizer'),
                  XYView(z=None, ax=axes[1], xlim=[-100,600], title='Default flowVisualizer'),
                  XYView(z=None, ax=axes[2], xlim=[-100,600],
                         flowVisualizer=Flow2DVisualizer(uvw='uv', include_wakes=False, levels=
fs.show(view)
```



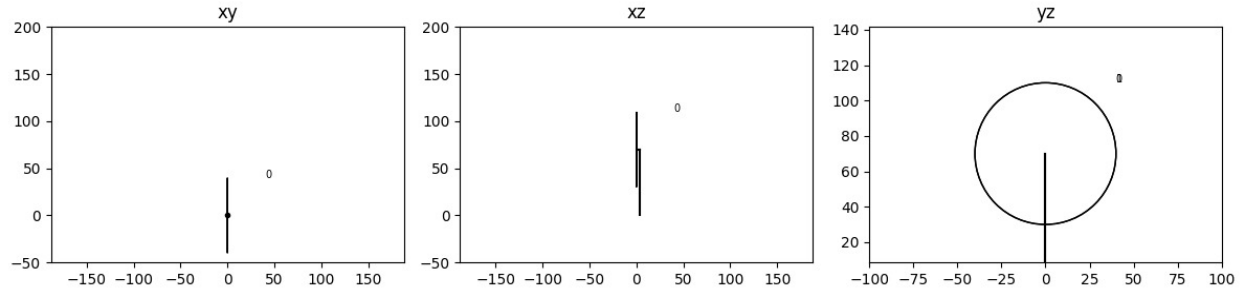
WindTurbineVisualizer

The `WindTurbineVisualizer` is automatically added the the list of visualizers for 2D views. It can be added for custom views, `View` (a `plane` field is required).

The actual turbine visualization depends on the turbine class. For `PyWakeWindTurbines` draws a projection of the disk

[12]:
```
axes = plt.subplots(1,3, figsize=(12,3))[1]
view_lst = []
for ax, plane in zip(axes, ['xy','xz','yz']):
    view = View(visualizers=[WindTurbineVisualizer()], ax=ax, xlim=[-100,100], ylim=[-50,200],

    view.plane = plane
    view_lst.append(view)
fs.show(MultiView(view_lst))
```

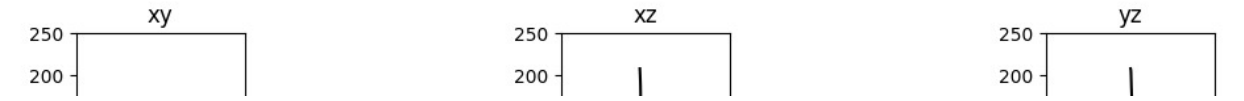
```
Ignoring fixed x limits to fulfill fixed data aspect with adjustable data limits.
Ignoring fixed x limits to fulfill fixed data aspect with adjustable data limits.
Ignoring fixed y limits to fulfill fixed data aspect with adjustable data limits.
Ignoring fixed x limits to fulfill fixed data aspect with adjustable data limits.
Ignoring fixed x limits to fulfill fixed data aspect with adjustable data limits.
Ignoring fixed y limits to fulfill fixed data aspect with adjustable data limits.
```

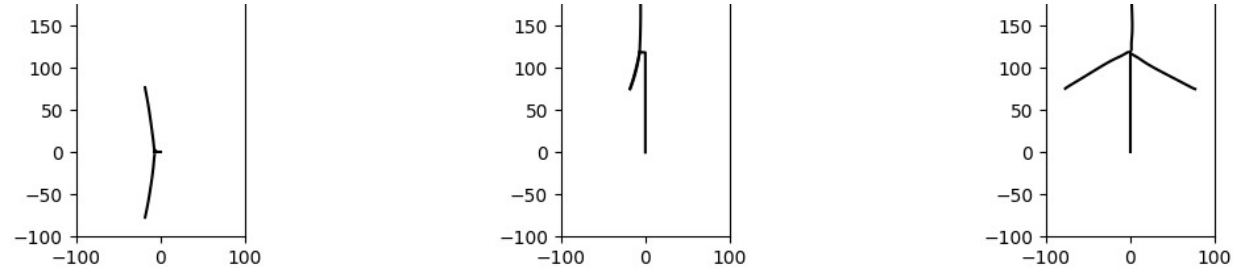


while `HAWC2WindTurbines` draws the rotor with lines representing the dynamic deflected blades.

Note, the tower and shaft are drawn as straight lines, which are connected in an estimated tower top point.

[13]:
```
from h2lib_tests.test_files import tfp as h2lib_tfp
from dynamiks.wind_turbines.hawc2_windturbine import HAWC2WindTurbines

htc_filename_lst = [h2lib_tfp + "DTU_10_MW/htc/DTU_10MW_RWT.htc"]
wt_h2 = HAWC2WindTurbines(x=[0], y=[0], htc_filename_lst=htc_filename_lst, types=0, suppress_ou
fs_h2 = DefaultDWMFlowSimulation(x=[0,500],y=[0,0], windTurbines=wt_h2, ti=0.05, d_particle=1,
axes = plt.subplots(1,3, figsize=(12,3))[1]
view_lst = []
for ax, plane in zip(axes, ['xy','xz','yz']):
    view = View(visualizers=[WindTurbineVisualizer()], ax=ax, xlim=[-100,100], ylim=[-100,250],

    view.plane = plane
    view_lst.append(view)
fs_h2.show(MultiView(view_lst))
```

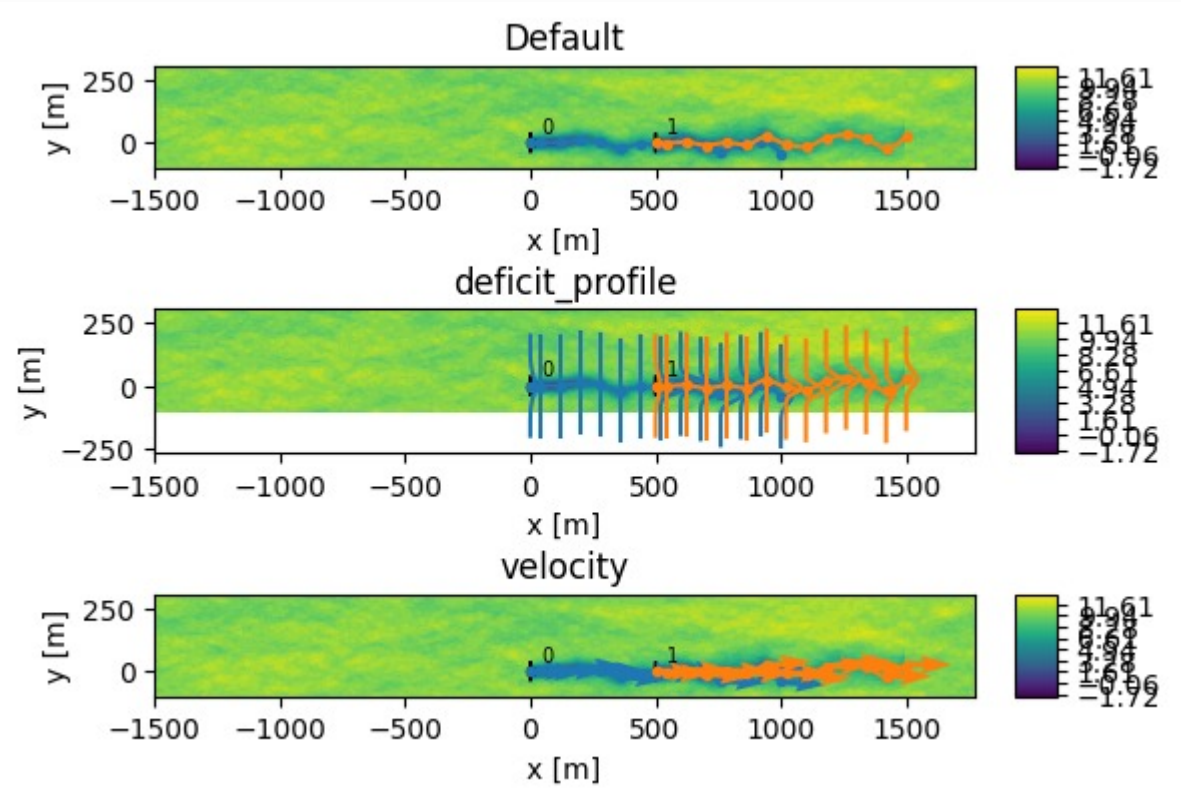




ParticleVisualizer

The `ParticleVisualizer` shows the deficit particles and their trajectories

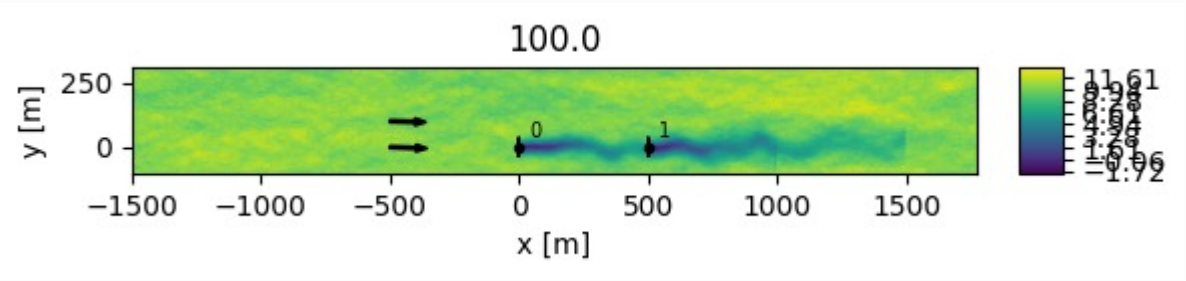
[14]:
```
axes = plt.subplots(3, 1, figsize=(6,4))[1]

fs.show(view=MultiView([
    XYView(z=None, ax=axes[0], visualizers=[ParticleVisualizer(deficit_profile=False, velocity=
    XYView(z=None, ax=axes[1],visualizers=[ParticleVisualizer(deficit_profile=True, velocity=Fa
    XYView(z=None, ax=axes[2], visualizers=[ParticleVisualizer(deficit_profile=False, velocity=
    ))
```



WindDirectionVisualizer

The `WindDirectionVisualizer` draws arrow(s) indicating the wind direction at the specified position(s). The size of the arrows is controlled by the `scale` argument

[15]:
```
plt.figure(figsize=(6,2))
fs.show(view=XYView(z=None, visualizers=[
    WindDirectionVisualizer(x=[-500,-500],y=[0,100],z=[70,70],scale=10),
]))
```

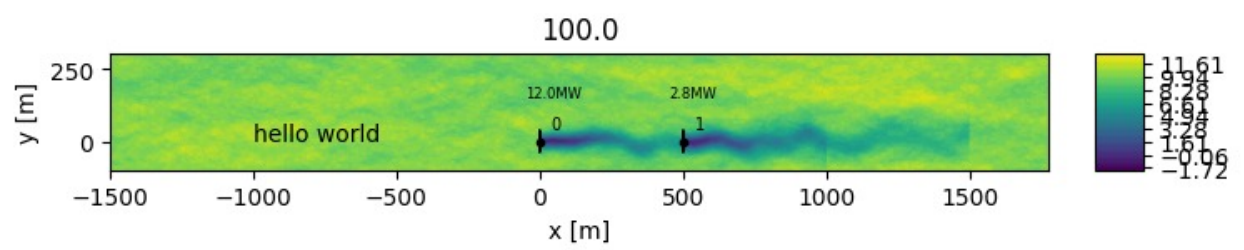


Custom visualizer

Custom plotting functions can also be appended to the `visualizers` list.

It can either be a pure function or a subclass of `Visualizer`. The pure-function option plots on the current or hard-coded axes, while the `Visualizer` subclass object has access to the axes from its parent `View`

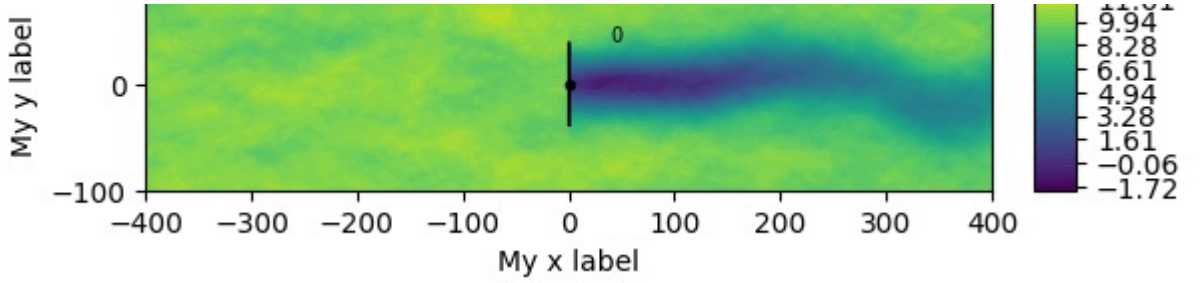
[16]:
```
plt.figure(figsize=(8,2))
def hello_world(flowSimulation):
    plt.annotate('hello world', (-1000,0)) # plot on current axis


class PowerPlotter(Visualizer):
    def __call__(self, fs):
        current_power = fs.windTurbines.sensors.to_xarray(dataset=True).power[-1]
        for (x,y,z), wt_power in zip(fs.windTurbines.positions_xyz.T, current_power):
            self.ax.annotate(f'{wt_power/1e5:.1f}MW', (x-50,y+150), fontsize=6) # self.ax point

fs.show(view=XYView(z=None, visualizers=[
    hello_world,
    PowerPlotter(),
]))
```



Custom axis initialization arguments

Most `View` take the optional axis initialization arguments: `xlim`, `ylim`, `xlabel`, `ylabel`, `title`, `axis` and `clear`

[17]:
```
plt.figure(figsize=(6,3))
fs.show(XYView(z=None, xlim=[-400,400], ylim=[-100,100], xlabel='My x label', ylabel='My y labe
```



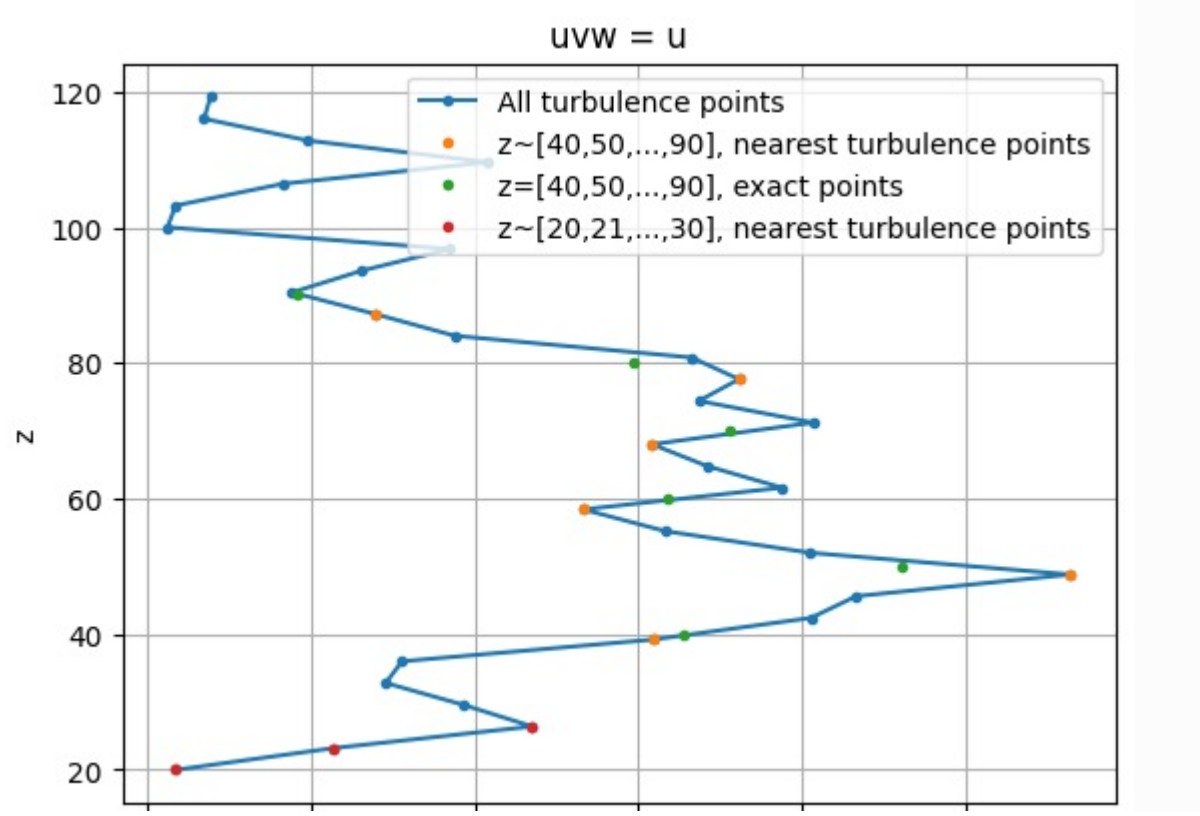
Adaptive views

Most view takes the argument, `adaptive`, which controls whether the view should align with the nearest turbulence grid points or interpolate to the exact specified points.

If `adaptive=True` (default), unspecified axes, will default to the grid in the turbulence box

The benefit of adaptive views is that the interpolation step can be disregarded. The result is faster simulation and more correct turbulence (as coarse linear interpolation reduces the turbulent variations)

[18]:
```
for view, l in [(ZView(x=100,y=0, adaptive=True), 'All turbulence points'),
                (ZView(x=100,y=0, z=np.arange(40,100,10),adaptive=True), 'z~[40,50,...,90],
                (ZView(x=100,y=0, z=np.arange(40,100,10),adaptive=False), 'z=[40,50,...,90]
                (ZView(x=100,y=0, z=np.arange(20,30),adaptive=True), 'z~[20,21,...,30], nea
               ]:
    fs.get_windspeed(xyz=view, include_wakes=False, xarray=True).sel(uvw='u').plot(label=l, y='

plt.legend()
plt.grid()
```



Note, the reason why the green points not match the blue line is because the blue line is extracted at nearest turbulence grid point, (x,y)=(100,-0.8), while the non-adaptive green dots are extracted at exactly (x,y)=(100,0).

Previous Next

Edit on Gitlab launch binder

Interaction

There are two ways to interact with Dynamiks:

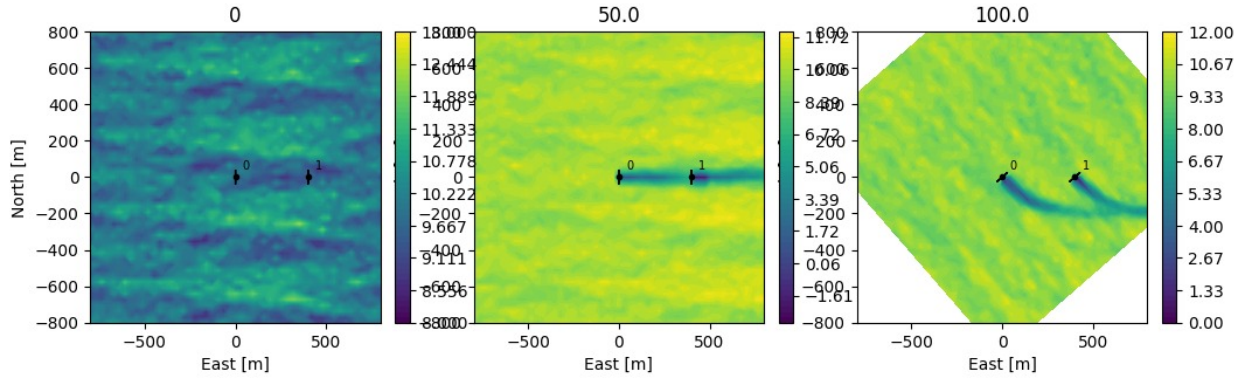
1. Make your own custom time-stepping loop and add the interaction in the loop
2. Use Dynamiks' time-stepping loop and interact via step handlers

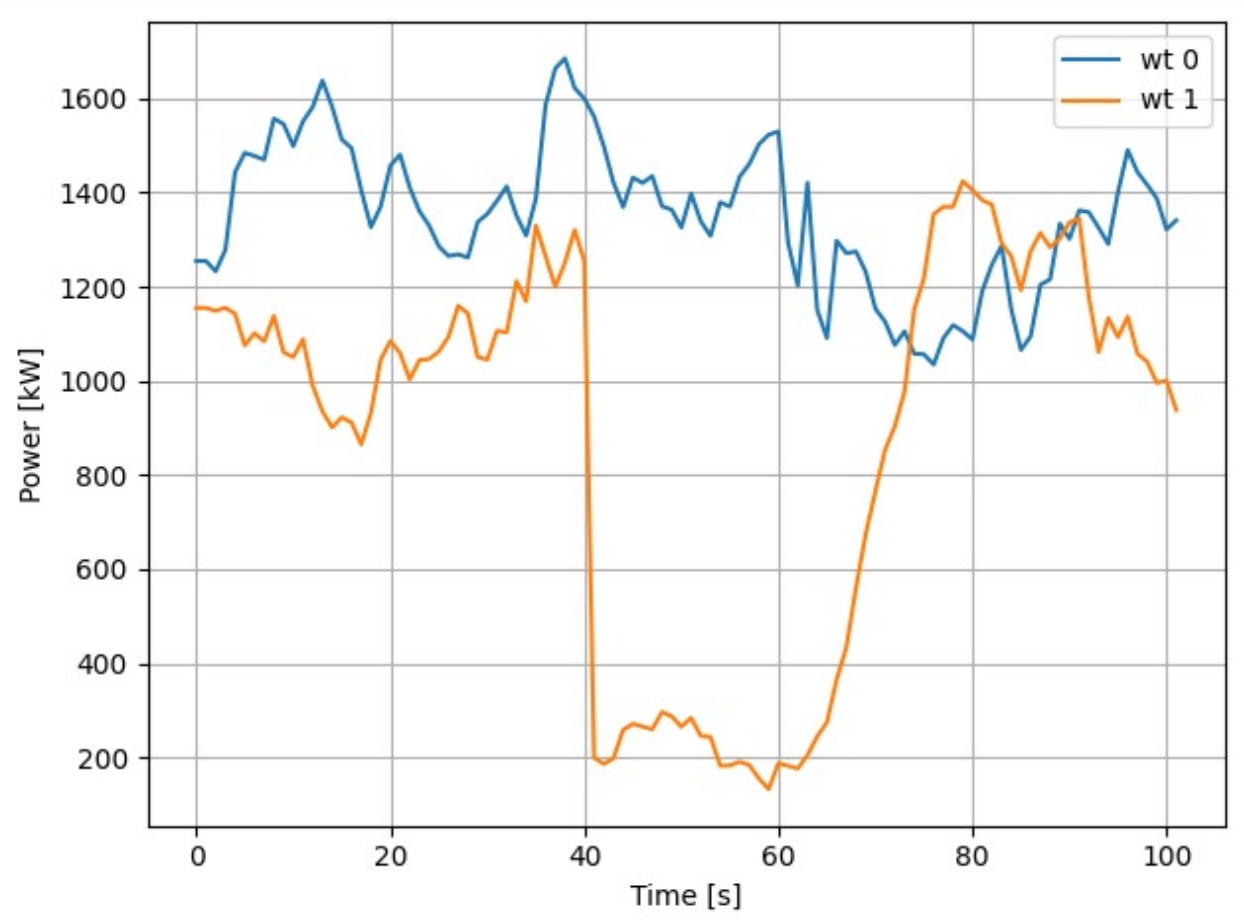
[1]:
```
import numpy as np
import xarray as xr
import matplotlib.pyplot as plt
from dynamiks.utils.test_utils import DefaultDWMFlowSimulation
from dynamiks.dwm.particle_motion_models import ParticleMotionModel, XSpeed
from dynamiks.visualizers import ParticleVisualizer
from dynamiks.visualizers.flow_visualizers import Flow2DVisualizer
from dynamiks.utils.data_dumper import DataDumper
from dynamiks.views import YView, EastNorthView, XYView
from py_wake.utils.plotting import setup_plot
```

Custom time-stepping loop

[2]:
```
fs = DefaultDWMFlowSimulation(x=[0,400],y=[0,0], ti=.05)

data = []
u_lst=[]
power_lst = []
wd_lst = []
time_lst = range(0,102)

axes = plt.subplots(1,3, figsize=(12,4))[1]
for t in time_lst: # custom time-stepping loop
    fs.run(t)  # step dynamiks

    # plot the flow field every 50s
    if t%50==0:
        view=EastNorthView(z=70, x=np.linspace(-800,800), y=np.linspace(-800,800),ax=axes[t//50
        fs.show(view, block=False)

    # change wind direction one deg per time step
    if t>50:
        fs.wind_direction +=1
```

```
    power_lst.append(fs.windTurbines.power()/1000)

da = xr.DataArray(power_lst, coords={'time':time_lst, 'wt':[0,1]})
plt.figure()
for wt in da.wt:
    da.sel(wt=wt).plot(label=f'wt {wt.item()}')
setup_plot(xlabel='Time [s]', ylabel='Power [kW]', title='')
```





Step handlers

Step handlers are functions that is executed in the end of Dynamiks' time-stepping loop.

```
# dynamiks time-stepping loop
while fs.time<time_stop:
    fs.time+=dt
```

```
    ...
    for step_handler in self.step_handlers:
        step_handler(self)
```

Step handlers must take `flowSimulation`-object as input, `f(flowSimulation) -> None`

An example of a function that appends the time and u component to a list, as we also did above

[3]:
```
def plot_flow(flowSimulation):
    view=EastNorthView(z=70, x=np.linspace(-800,800), y=np.linspace(-800,800),flowVisualizer=Fl
                       ax=plt.gcf().axes[int(flowSimulation.time//50)])
    flowSimulation.show(view, block=False, )
```

another example is a function that changes the wind direction, as we also did above

[4]:
```
def wd_changer(fs):
    fs.wind_direction+=1
```

DataDumpers

The `DataDumper` class is a special kind of step handlers. It appends data to a list, similar to the `u_lst_dumper` example above, but it has a function to export the data to a xarray dataset.

`DataDumper` takes a function, `f(flowSimulation) -> data` and an optionally dictionary with the coordinates of the data

[5]:
```
def get_power(fs): # f(flowSimulation) -> (power_wt1, power_wt2)
    return fs.windTurbines.power()/1000

power_dumper = DataDumper(get_power,
                          coords={'wt':[0,1]}) # coordinates of first axis
```

Specifying step handlers

The `FlowSimulation` class takes a list of step handlers as input

```
fs = FlowSimulation(..., step_handlers=[my_step_handler1, my_step_handler2])
```

Alternatively, step handlers, can be appended afterwards:

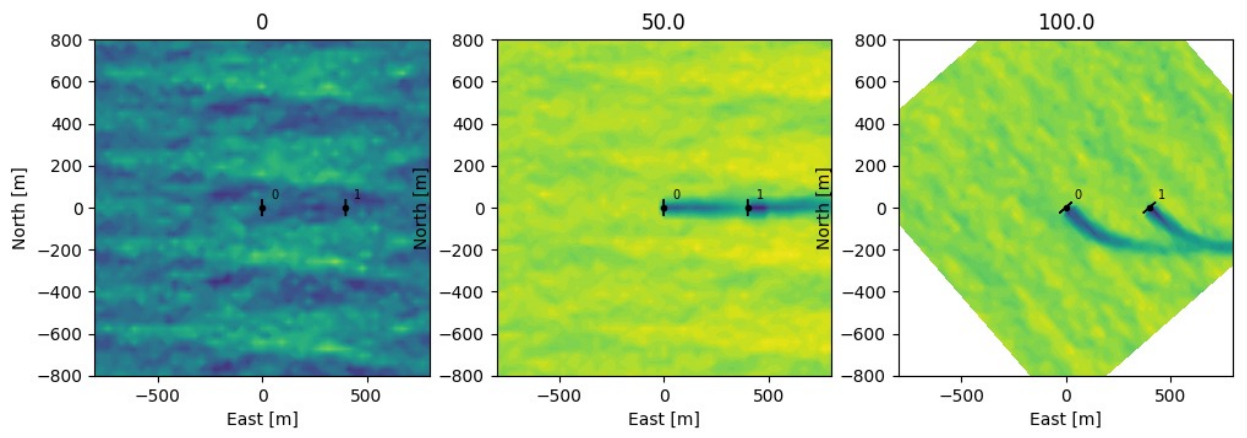
```
fs = FlowSimulation(...)
fs.step_handlers.append(my_step_handler3)
```

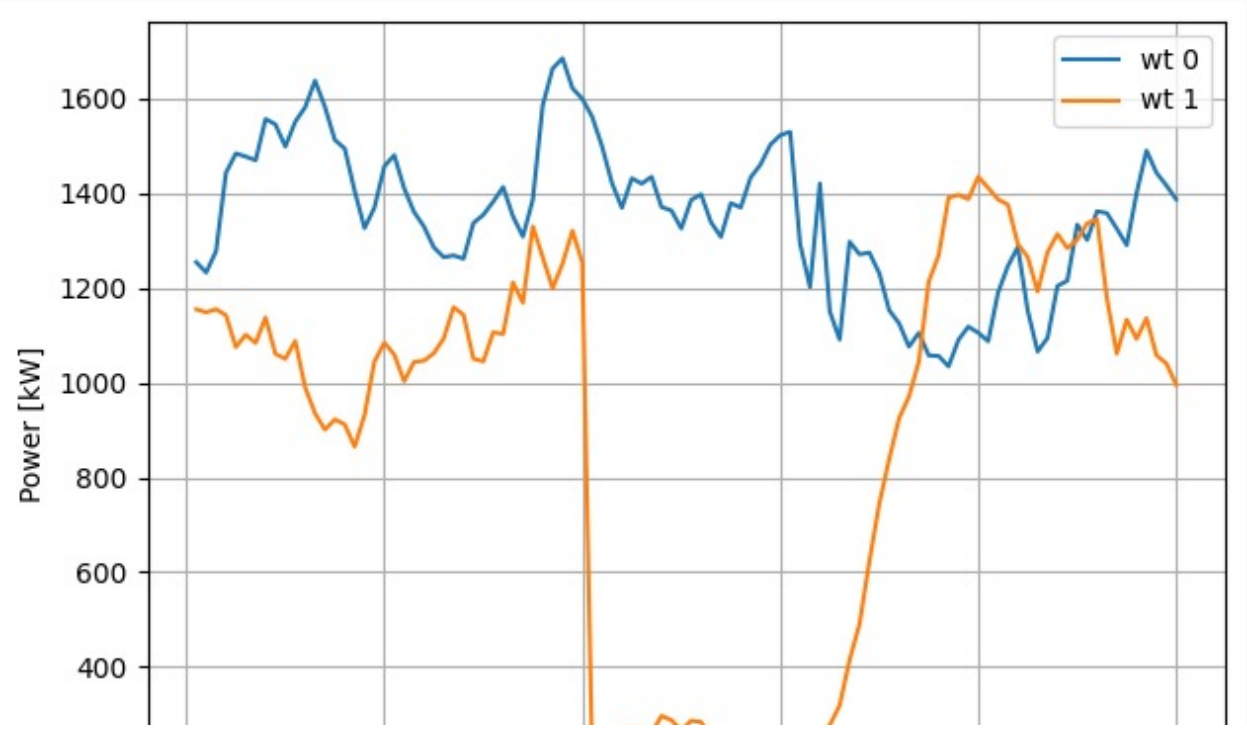
Custom `start` and `step`

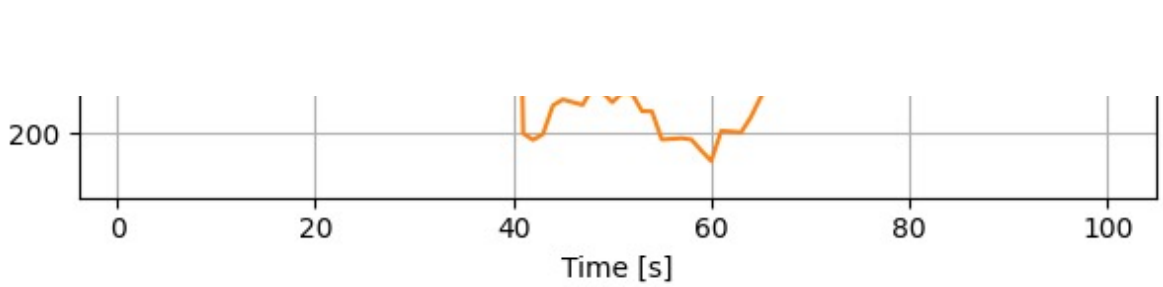
It is possible to specify a custom start time and step time for the step handlers like this:

```
step_handlers=[my_step_handler1,          # executes every time step
               ((10,5), my_step_handler2) # executes every 5th second after the first 10s, i.e. [15
               ]
```

[6]:
```
axes = plt.subplots(1,3, figsize=(12,6))[1]
fs = DefaultDWMFlowSimulation(x=[0,400],y=[0,0], ti=.05,
                              step_handlers=[((0,50), plot_flow),
                                             power_dumper,
                                             ((50,1), wd_changer)])
plot_flow(fs) # plot flow of t=0
fs.run(100) # run dynamiks time-stepping loop

da = power_dumper.to_xarray()
plt.figure()
for wt in da.wt:
    da.sel(wt=wt).plot(label=f'wt {wt.item()}')
setup_plot(xlabel='Time [s]', ylabel='Power [kW]', title='')
```







Previous

Next

Wind farm yaw control, simple

Assume we have a wind farm with three wind turbines and we want to optimize the total farm production via yaw misalignment control.

- Wind turbines: 3 x DTU 10MW reference wind turbines
- Distance: 5D (892m)
- Wind speed: 10 m/s
- Wind direction: 240-300°
- Turbulence intensity: 10%

Simplifications:

- Power-ct relation follows cosine law
-

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
from py_wake.examples.data.dtu10mw._dtu10mw import DTU10MW
from py_wake.utils.plotting import setup_plot
from dynamiks.utils import doc_utils
```

[2]:
```
wt = DTU10MW()
wt_x = np.arange(3) * 5 * wt.diameter()
wt_y = wt_x * 0
U = 10
wd_lst = np.arange(240, 301,2)
TI = .06
```

Find optimial yaw settings

Optimize with Topfarm and PyWake

To find the optimal yaw settings for wind directions from 240-300° we set up a Topfarm optimization with $yaw(wt, wd)$ as design variables and using PyWake AEP as the objective to

maximize.

Alternatively you can use the hard coded refernce values below

[3]:
```
# setup PyWake AEP and gradient function
from py_wake.wind_farm_models.engineering_models import PropagateDownwind
from py_wake.site._site import UniformSite
from py_wake.deficit_models.gaussian import NiayifarGaussianDeficit
from py_wake.deflection_models.gcl_hill_vortex import GCLHillDeflection
from py_wake.turbulence_models.crespo import CrespoHernandez
from py_wake.utils.gradients import autograd
from py_wake.rotor_avg_models.rotor_avg_model import CGIRotorAvg


wt = DTU10MW()
wfm = PropagateDownwind(UniformSite(ws=U, ti=TI), wt, NiayifarGaussianDeficit(),
                        deflectionModel=GCLHillDeflection(),
                        turbulenceModel=CrespoHernandez(),
                        rotorAvgModel=CGIRotorAvg(21))
wt_x = np.arange(3) * 5 * wt.diameter()
wt_y = wt_x * 0
wd_lst = np.arange(240, 301,2)
yaw = np.ones((3, len(wd_lst))) # one deg misalignment as initial guess to get out of local min

def aep(yaw):
    return wfm.aep(wt_x, wt_y, yaw=yaw.reshape((3, len(wd_lst))), tilt=0, wd=wd_lst)

def daep(yaw):
    return [autograd(aep)(yaw)]

def plot(yaw,wd):
    wfm(wt_x, wt_y, yaw=yaw, tilt=0, wd=wd).flow_map().plot_wake_map()
```

[4]:
```
%%capture
# setup topfarm problem and optimize
try:
    import topfarm
except ModuleNotFoundError:
    !pip install topfarm # install topfarm if not installed
```

[5]:
```
from topfarm._topfarm import TopFarmProblem
from topfarm.cost_models.cost_model_wrappers import AEPCostModelComponent
cost_comp = AEPCostModelComponent(input_keys=['yaw'], n_wt=len(yaw.flatten()),
                                  cost_function=aep, cost_gradient_function=daep)
tf = TopFarmProblem(design_vars={'yaw': (yaw.flatten(), -40, 40)}, cost_comp=cost_comp, n_wt=le
```

```
INFO: checking out_of_order...
INFO:     out_of_order check complete (0.000374 sec).
INFO: checking system...
INFO:     system check complete (0.000010 sec).
INFO: checking solvers...
INFO:     solvers check complete (0.000081 sec).
INFO: checking dup_inputs...
INFO:     dup_inputs check complete (0.000022 sec).
INFO: checking missing_recorders...
INFO:     missing_recorders check complete (0.000004 sec).
INFO: checking unserializable_options...
INFO:     unserializable_options check complete (0.000106 sec).
```

```
INFO: checking comp_has_no_outputs...
INFO:     comp_has_no_outputs check complete (0.000017 sec).
INFO: checking auto_ivc_warnings...
INFO:     auto_ivc_warnings check complete (0.000003 sec).
```

[6]:
```
tf.optimize()
yaw_tabular = tf.state['yaw'].reshape((3, len(wd_lst)))
yaw_tabular = np.round(yaw_tabular).astype(int)
print(yaw_tabular)
```

```
INFO: checking out_of_order...
INFO:     out_of_order check complete (0.000115 sec).
INFO: checking system...
INFO:     system check complete (0.000013 sec).
INFO: checking solvers...
INFO:     solvers check complete (0.000078 sec).
INFO: checking dup_inputs...
INFO:     dup_inputs check complete (0.000018 sec).
INFO: checking missing_recorders...
INFO:     missing_recorders check complete (0.000004 sec).
INFO: checking unserializable_options...
INFO:     unserializable_options check complete (0.000086 sec).
INFO: checking comp_has_no_outputs...
INFO:     comp_has_no_outputs check complete (0.000018 sec).
INFO: checking auto_ivc_warnings...
INFO:     auto_ivc_warnings check complete (0.000003 sec).
Optimization terminated successfully    (Exit mode 0)
            Current function value: -29.651020222966046
            Iterations: 98
            Function evaluations: 104
            Gradient evaluations: 98
Optimization Complete
-----------------------------------
[[  0   0   0   0   0   0  -1  -2  -4  -6  -8 -11 -15 -18 -22  25  22  18
   15  11   8   6   4   2   1   0   0   0   0   0   0]
 [  0   0   0   0   0   0  -1  -2  -4  -6  -8 -11 -14 -17 -20  24  20  17
   14  11   8   6   4   2   1   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0]]
```

[7]:
```
print(str(yaw_tabular.tolist()).replace(" ",""))
```

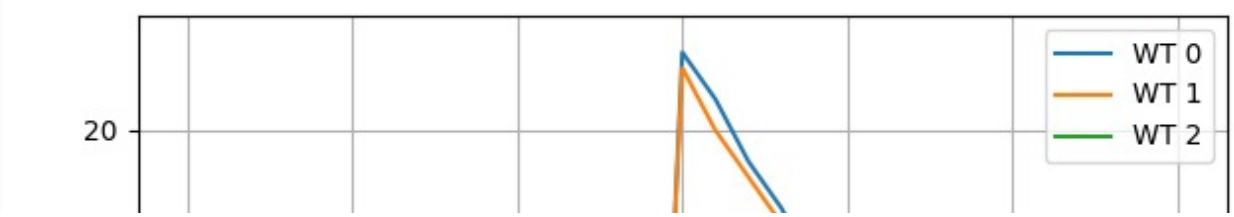
```
[[0,0,0,0,0,0,-1,-2,-4,-6,-8,-11,-15,-18,-22,25,22,18,15,11,8,6,4,2,1,0,0,0,0,0,0],[0,0,0,0,0,0
```

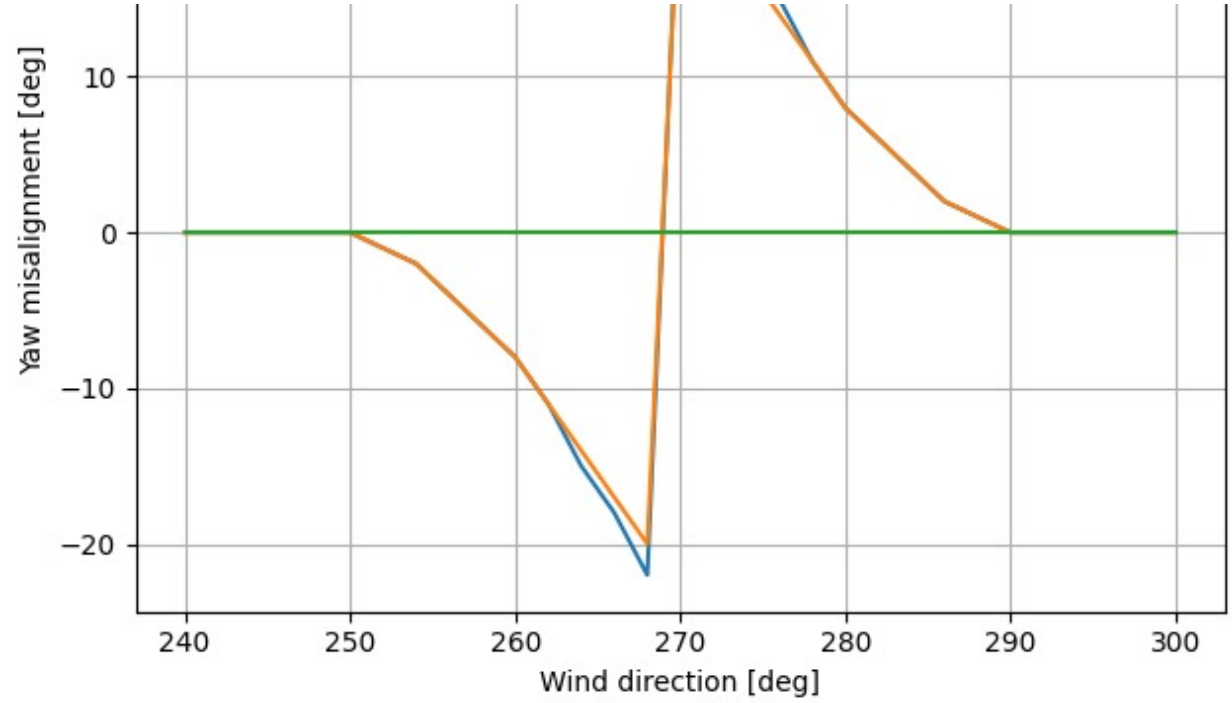
Hard coded yaw tabular

[8]:
```
yaw_tabular=np.array([[0,0,0,0,0,0,-1,-2,-4,-6,-8,-11,-15,-18,-22,25,22,18,15,11,8,6,4,2,1,0,0,
                      [0,0,0,0,0,0,-1,-2,-4,-6,-8,-11,-14,-17,-20,24,20,17,14,11,8,6,4,2,1,0,0,
                      [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]])
```

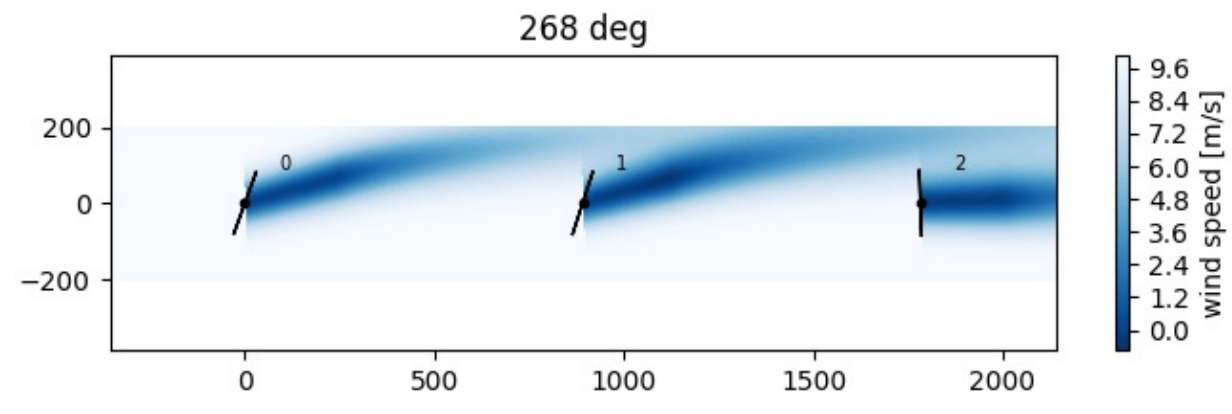
Plot optimial yaw settings

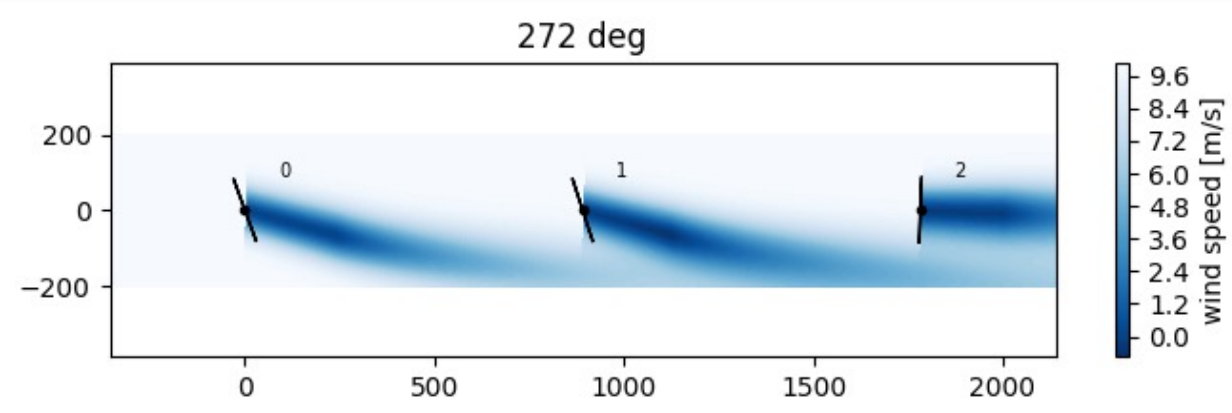
[9]:
```
for i, y_ in enumerate(yaw_tabular):
    plt.plot(wd_lst, y_, label=f'WT {i}')
setup_plot(xlabel='Wind direction [deg]', ylabel='Yaw misalignment [deg]')
```





[10]:
```
for wd in [268,272]:
    plt.figure(figsize=(8,2))
    plot(yaw_tabular[:,wd_lst==wd][:,0],wd)
    plt.title(f'{wd} deg')
```





Dynamiks wind farm control

Wind farm controller

[11]:
```
def simple_wind_farm_controller(flowSimulation):
    wd = flowSimulation.wind_direction
    wd_index = np.argmin(np.abs(wd_lst - wd))
    yaw = yaw_tabular[:,wd_index]
    flowSimulation.windTurbines.yaw = yaw
```

Wind direction changer

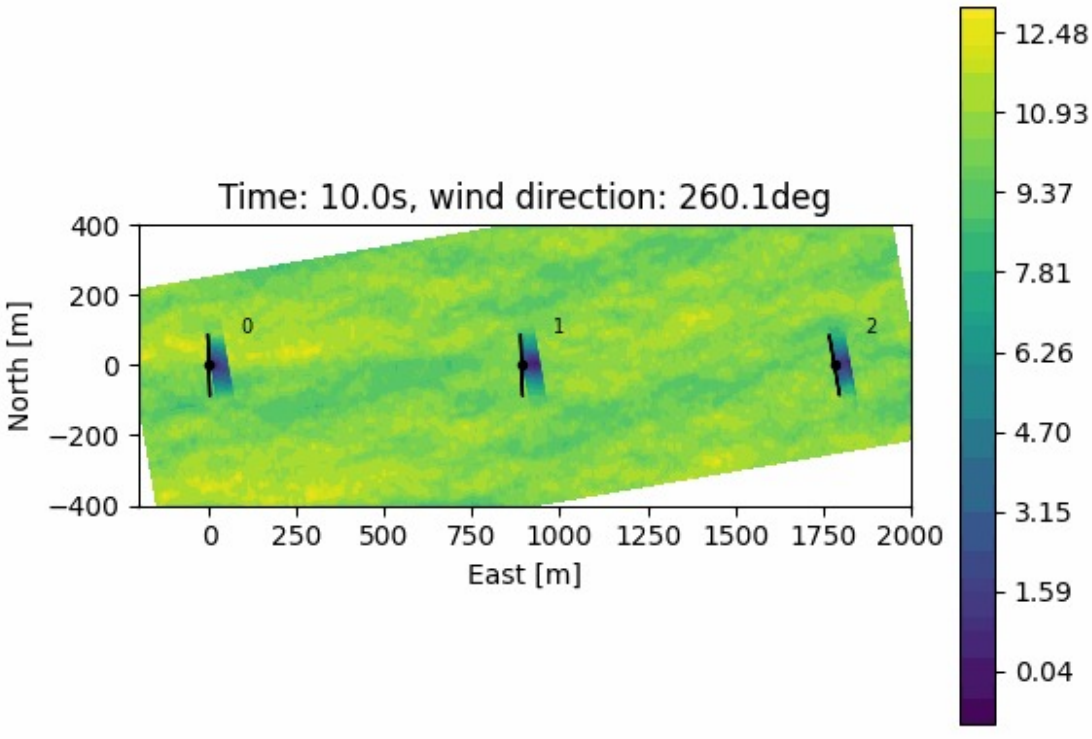
[12]:
```
def wind_direction_changer(flowSimulation):
    flowSimulation.wind_direction = 260+flowSimulation.time/100
```

Dynamiks DWMFlowSimulation

[13]:
```
from dynamiks.utils.test_utils import DefaultDWMFlowSimulation, DemoSite
from dynamiks.dwm.particle_motion_models import HillVortexParticleMotion
from dynamiks.wind_turbines.pywake_windturbines import PyWakeWindTurbines
from dynamiks.views import XYView, EastNorthView, MultiView

wts = PyWakeWindTurbines(x=wt_x, y=wt_y, windTurbine=DTU10MW())
fs = DefaultDWMFlowSimulation(windTurbines=wts, particleMotionModel=HillVortexParticleMotion(),
                              d_particle=.1, n_particles=100, ti=TI, ws=U,
                              step_handlers=[wind_direction_changer, simple_wind_farm_controller])
```

[14]:
```
fs.visualize(2000, dt=10, interval=.1, view=EastNorthView(
    x=np.linspace(-200, 2000, 500), y=np.linspace(-400, 400),
    visualizers=[lambda fs: plt.title(f'Time: {fs.time}s, wind direction: {fs.wind_direction}de
```

[14]:



[15]:
```
fs.run(2000, verbose=1)
power_yaw_control = wts.sensors.to_xarray(dataset=True).power
```

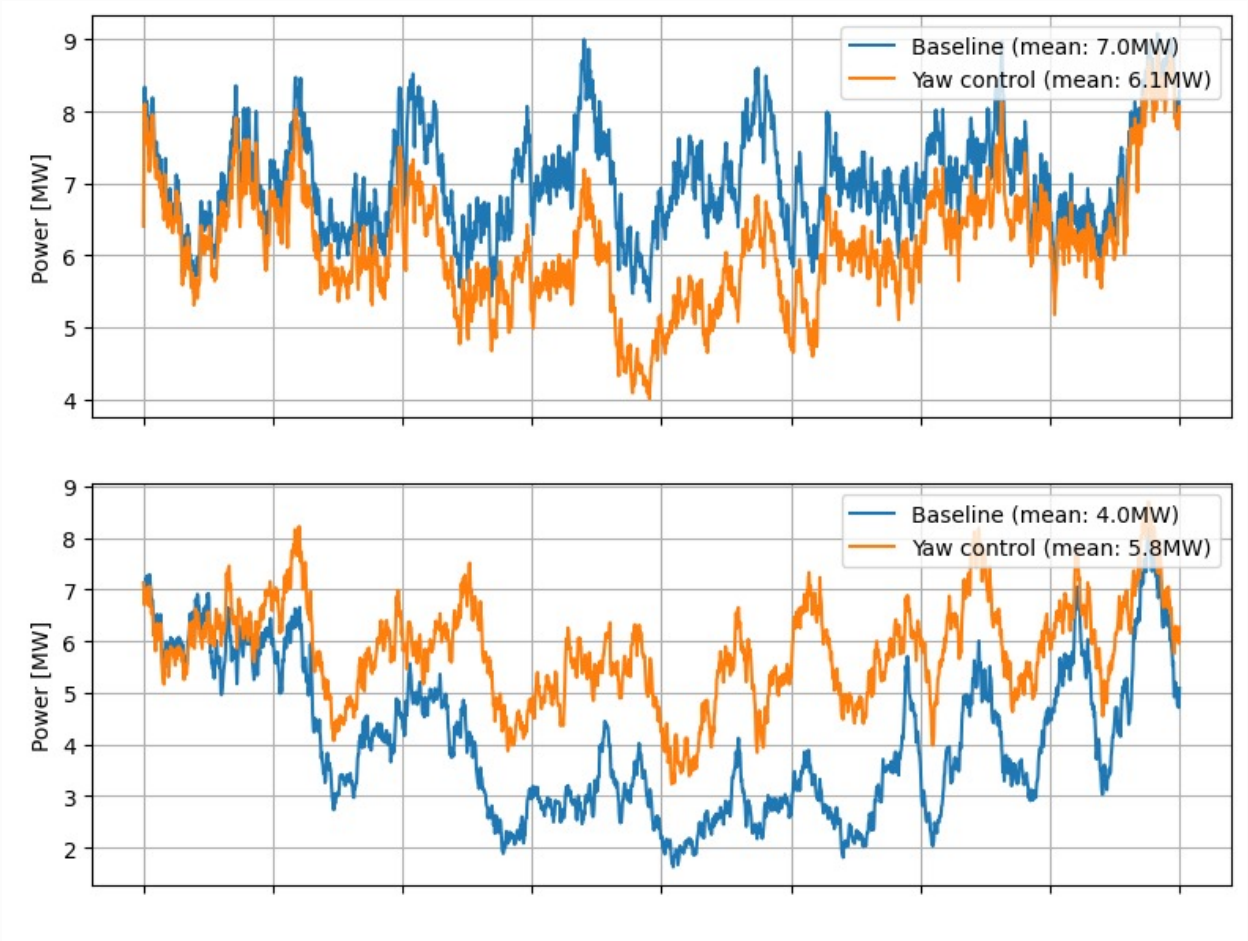
```
100% |████████████████████| 2000/2000 [01:19<00:00, 24.61it/s]
```

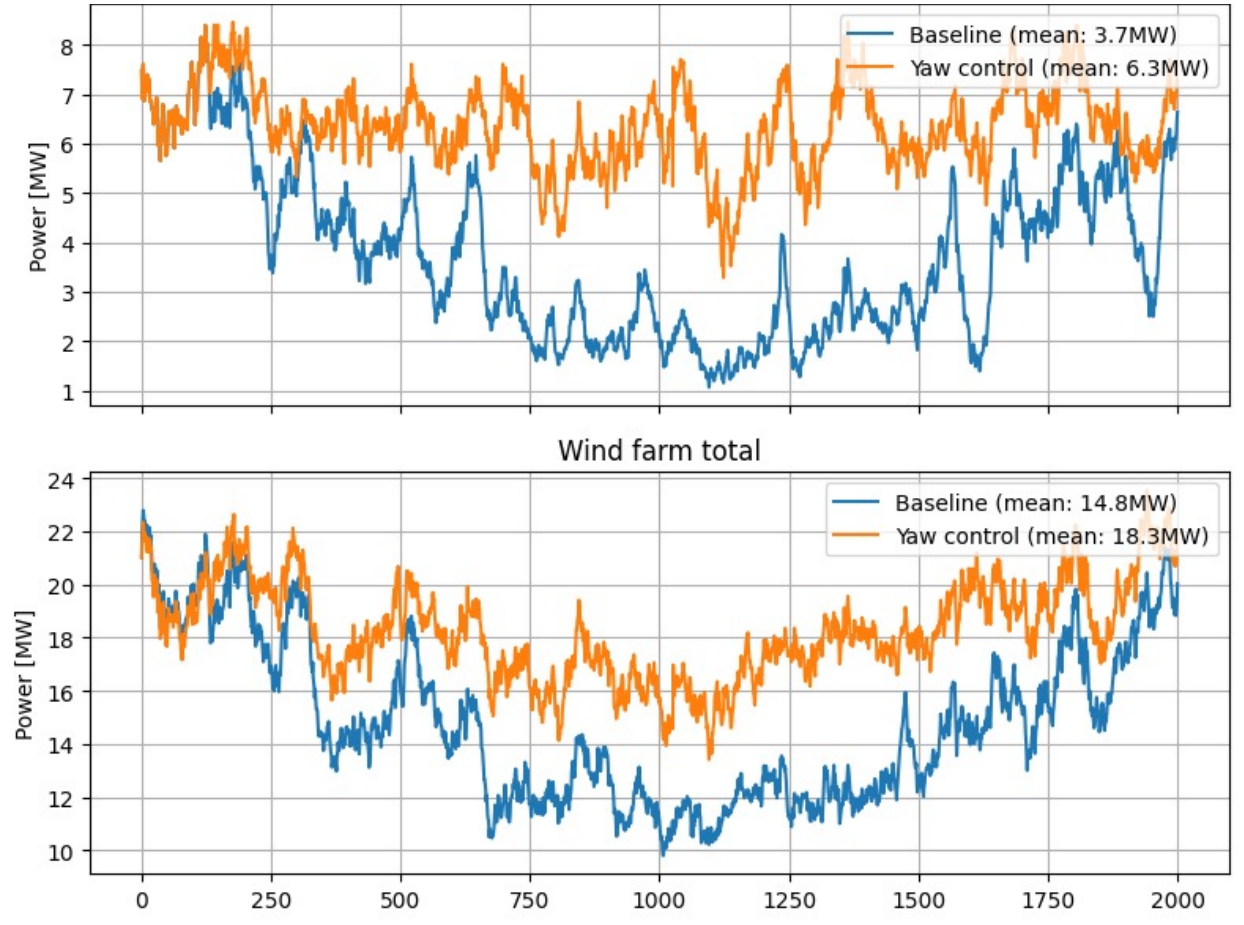
[16]:
```
wts = PyWakeWindTurbines(x=wt_x, y=wt_y, windTurbine=DTU10MW())
fs_baseline = DefaultDWMFlowSimulation(windTurbines=wts, particleMotionModel=HillVortexParticle
                          d_particle=.1, n_particles=100, ti=TI, ws=U,
                          step_handlers=[wind_direction_changer])
fs_baseline.run(2000, verbose=1)
```

```
100% |████████████████████| 2000/2000 [01:20<00:00, 24.11it/s]
```

[17]:
```
power_baseline = wts.sensors.to_xarray(dataset=True).power
```

[18]:
```
axes = plt.subplots(4,1, figsize=(8,12), sharex=True)[1]
for wt, ax in zip(power_yaw_control.wt, axes):
    for p,n in [(power_baseline,'Baseline'),(power_yaw_control,'Yaw control')]:
        p = p.sel(wt=wt)/1e6
        p.plot(ax=ax, label=f'{n} (mean: {p.mean().item():.1f}MW)')
    setup_plot(ax=ax,ylabel='Power [MW]')
    ax.legend(loc=1)
for p,n in [(power_baseline,'Baseline'),(power_yaw_control,'Yaw control')]:
    p = p.sum('wt')/1e6
    p.plot(ax=axes[3], label=f'{n} (mean: {p.mean().item():.1f}MW)')
setup_plot(ax=axes[3],ylabel='Power [MW]', title='Wind farm total')
axes[3].legend(loc=1)

plt.tight_layout()
```





Issues and simplifications

Simplifications:

- Power and ct follows the cosine law
  - $ct_x = ct_{normal}(U \cos\theta_{yaw}) \cos\theta_{yaw}{}^2$
  - $Power = Power(U \cos\theta_{yaw})$
- Wind turbine yaws instantaneous
- Mean Wind direction known from input
- Smooth change of mean wind direction from 260 to 280 deg

[ ]:

Previous Next

Implementation details and choises

Mapping time to position in turbulence field

When requestion turbulence at a point (x, y, z, t) in the wind farm, this point is be mapped to the corresponding turbulence field coordinate (x_t, y_t, z_T) utilizing Taylor's frozen turbulence hypothesis

Typically this is done by $(x_t, y_t, z_t) = (x + U \cdot t, y, z)$ where U is the turbulence field transport speed.

Moreover, the turbulence field may have an additional offset:

[1]:
```
def get_turbulence_point(x,y,z,t):
    xyz_t = np.array([x + turb_field_speed*t,y,z]) + turb_field_offset
```

This method, however, is problematic if the turbulence field transport speed changes, as a fixed point in the wind farm will jump dUt in the turbulence field.

To overcome this, $U \cdot t$ is changed to $\int_t U(t)dt$. In Dynamiks this is implemented by updating the offset in each time step to include the turbulence advection:

[2]:
```
t_last = 0
offset = (2500, 100, 20) # offset in x,y,z
def get_turbulence_point(x,y,z,t):
    if t>t_last:
        offset[0] += (t-t_last)*U
        t_last = t
    return np.array([x,y,z]) + offset
```

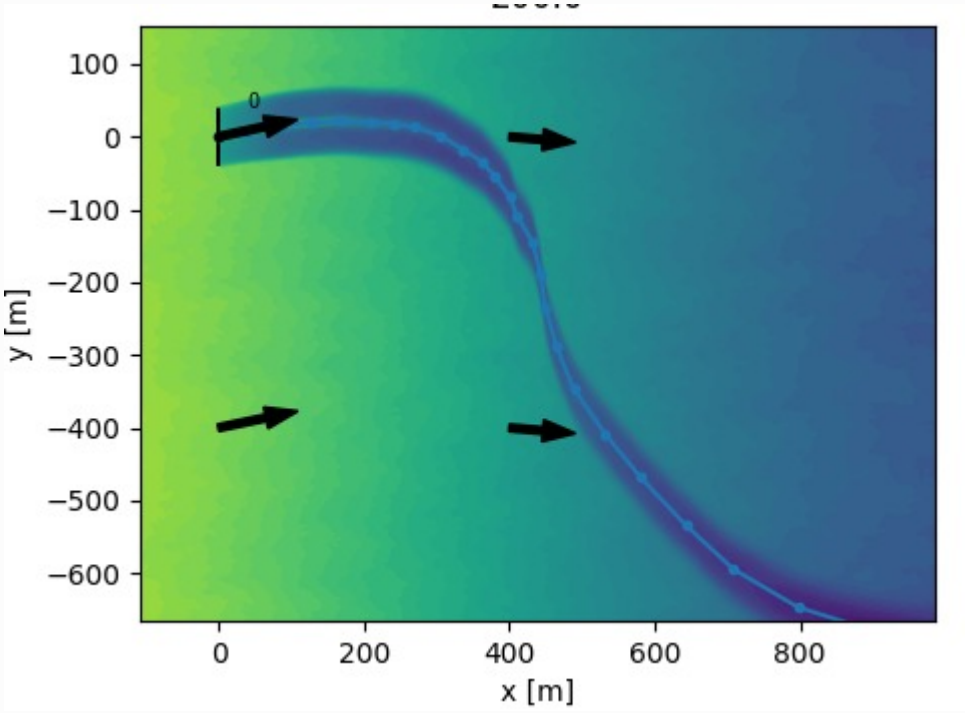
Deficit interpolation

The deficit profiles are defined in the (y-z) plane and advection along the x-axis. This results in questionable profile shape if distance between particles in the (y-z) plane is large compared to

the distance in along the x-axis.

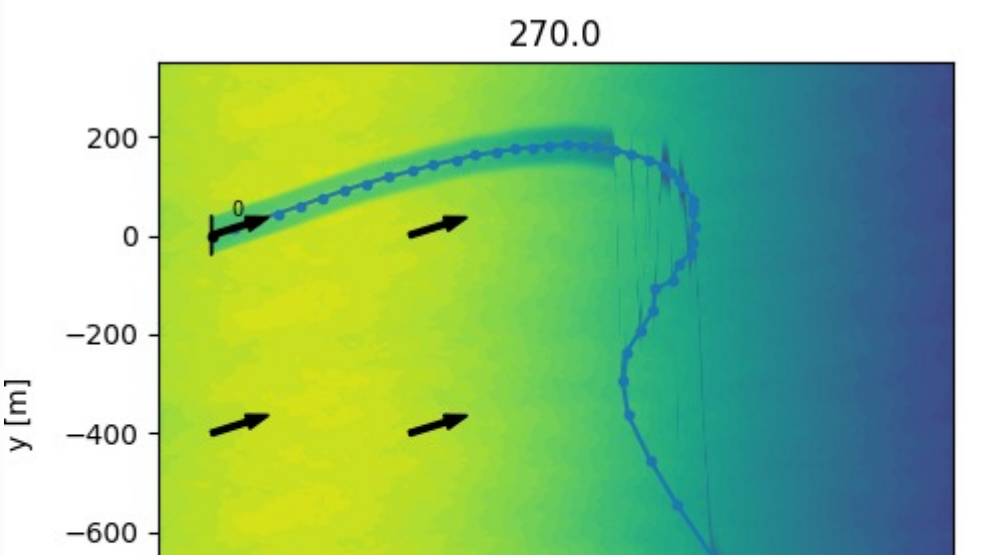
Possible solutions

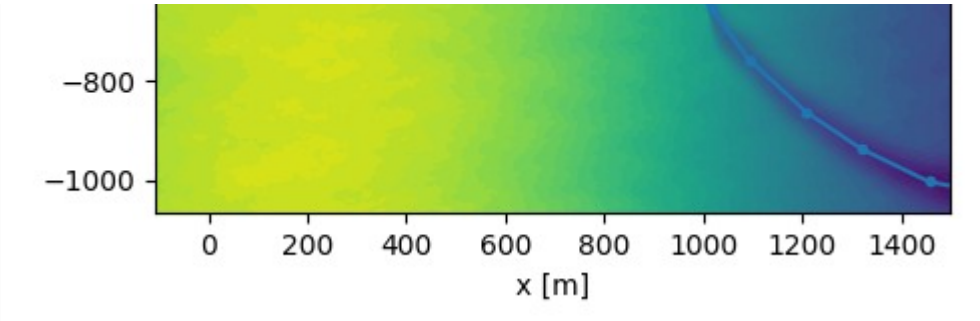
- Define the deficit profile perpendicular to the actual particle direction.
  - if the wake deficit rotates, then it should also change u deficit to v deficit. Does this make sense?
- Accept the narrow shape as it is a consequence of the meandering formulation



Overtaken particles

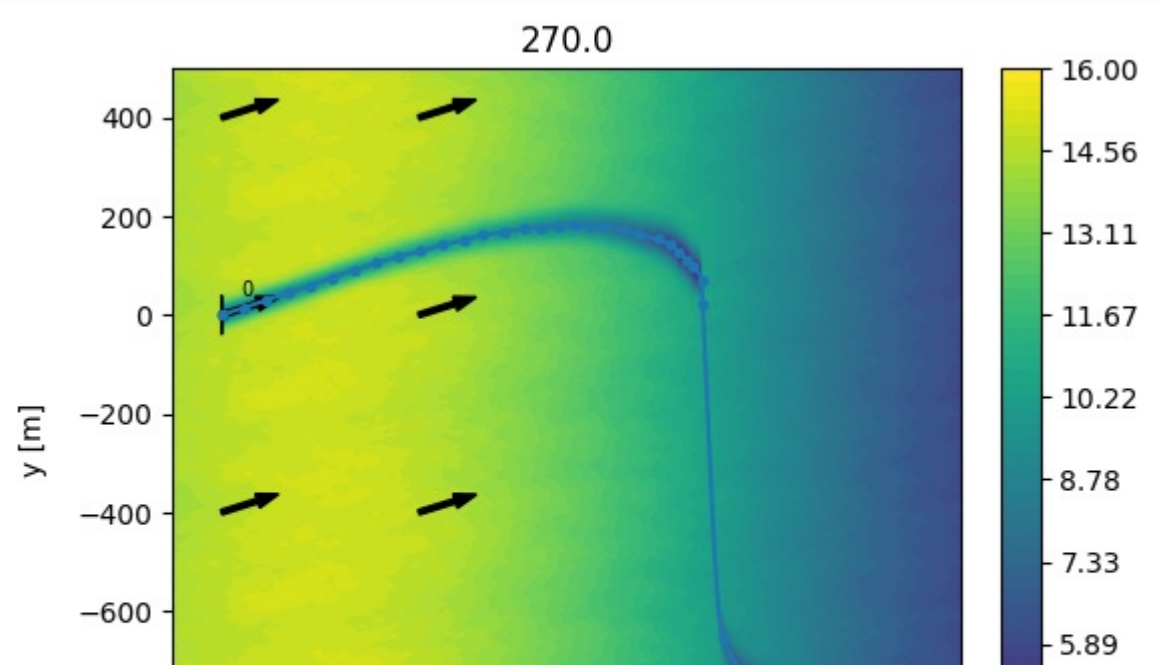
When a particle overtake another particle, the downstream positions, x, of the particles are not monotonically increasing. This results in wrong interpolation and Dynamiks are not able to interpolate the particle path using pchip

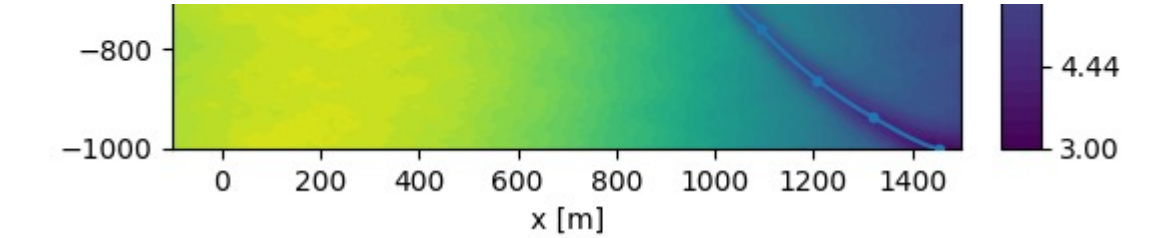




Current solution is to disregard particles that have been overtaken

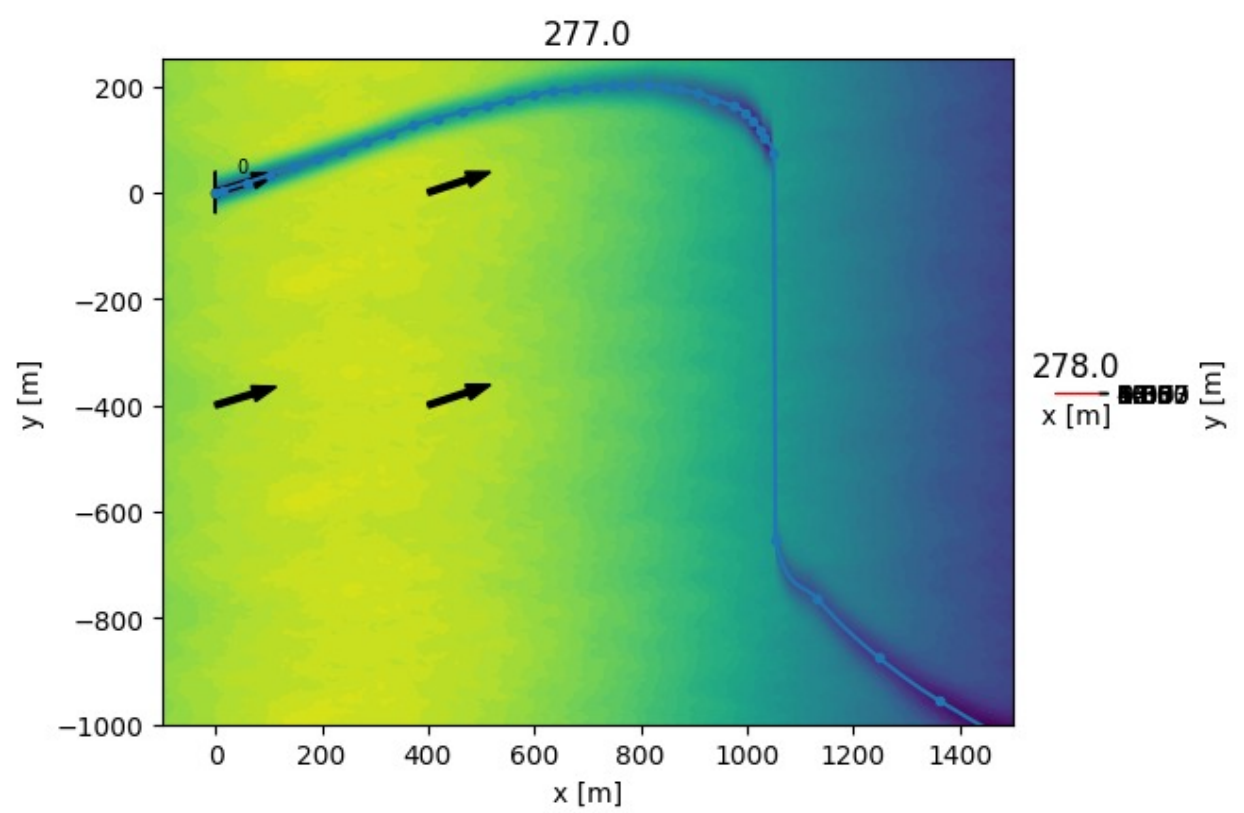
[3]:
```
from numpy import newaxis as na
from dynamiks.dwm.particle_motion_models import ParticleMotionModel, XSpeed
from dynamiks.utils.test_utils import DefaultDWMFlowSimulation, DemoSite
from dynamiks.visualizers import ParticleVisualizer
import numpy as np
from dynamiks.views import XYView
from dynamiks.visualizers._visualizers import WindDirectionVisualizer
import matplotlib.pyplot as plt


site = DemoSite(ws=10, ti=0.01)
site.turbulenceField.uvw[:2, ] += np.cos(np.linspace(0, 2 * np.pi, 1024)[:, na, na]) * 5  # ado
fs = DefaultDWMFlowSimulation(
    site=site,
    particleMotionModel=ParticleMotionModel(x_speed=XSpeed.Rotor),
    d_particle=.5,
    n_particles=100)
wd_x, wd_y = [v.flatten() for v in np.meshgrid([-400, 0, 400], [-400, 0, 400])]

fs.run(270)
view = XYView(z=70, x=np.linspace(-100, 1500, 500), y=np.linspace(-1000, 500, 500),
                    adaptive=True,
                    visualizers=[WindDirectionVisualizer(wd_x, wd_y, 70, 5),
                                 ParticleVisualizer(deficit_profile=False)])
fs.show(view=view)
```





This solution has the issue that the wake jumps when a particle is overtaken

[4]:
```
fs.run(277)
fs.show(view=XYView(z=70, x=np.linspace(-100, 1500, 500), y=np.linspace(-1000, 250, 500),
                    visualizers=[WindDirectionVisualizer(wd_x, wd_y, 70, 5),
                                 ParticleVisualizer(deficit_profile=False)]))
fs.step()
fs.show(view=XYView(z=70, x=np.linspace(-100, 1500, 500), y=np.linspace(-1000, 250, 500),
                    flowVisualizer=False,
                    visualizers=[ParticleVisualizer(deficit_profile=False, color='r')], clear=F
```



Linear or pchip particle path

[5]:
```
from dynamiks.utils.test_utils import DefaultDWMFlowSimulation, tfp
import matplotlib.pyplot as plt
import numpy as np
from scipy.interpolate._cubic import pchip_interpolate
from dynamiks.sites.turbulence_fields import MannTurbulenceField
from dynamiks.sites._site import TurbulenceFieldSite
from py_wake.utils.plotting import setup_plot
U = 10
turbfield = MannTurbulenceField.from_netcdf(
```

```
    tfp + "mann_turb/Hipersim_mann_l29.4_ae1.0000_g3.9_h0_1024x128x32_9.600x9.60x9.60_s0001.nc"
)
turbfield.scale_TI(TI=.1, U=U)
site = TurbulenceFieldSite(ws=U, turbulenceField=turbfield, turbulence_offset=(-2500, -100, 20)


def get_pxyz(d_particle):
    n_particles = int(1000 / (d_particle * 80)) + 2
    dt = d_particle / U

    fs = DefaultDWMFlowSimulation(x=[0], y=[0], site=site, d_particle=d_particle, n_particles=r
    fs.run(100, verbose=1)
    # fs.show(XYView(visualizers=[ParticleVisualizer()]))
    x = np.linspace(0, 1000, n_particles * 2)
    return fs.particle_position_xip[:, 0, fs.get_active_particles_idx(0, x)]
```

[6]:
```
d_particle_fine = 0.05
pxyz_fine = get_pxyz(.05)
```

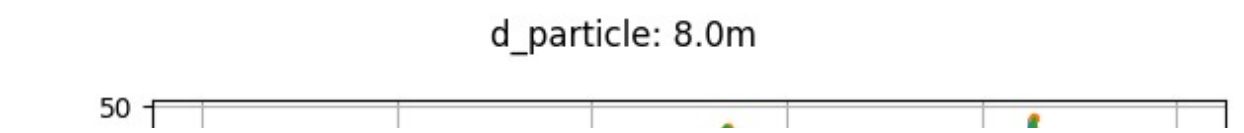
100% 20000/20000 [00:26<00:00, 740.04it/s]

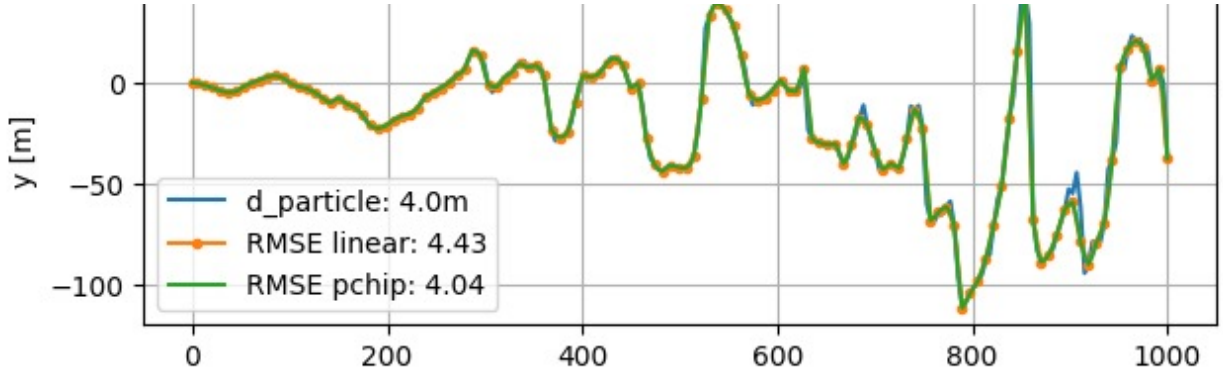
[7]:
```
for d_p in [.1, .5, 1]:
    fig, axes = plt.subplots(2, 1)
    fig.suptitle(f"d_particle: {d_p*80}m")
    axes[0].plot(pxyz_fine[0], pxyz_fine[1], label=f'd_particle: {d_particle_fine*80}m')
    axes[1].plot(pxyz_fine[0], pxyz_fine[2], label=f'd_particle: {d_particle_fine*80}m')
    pxyz = get_pxyz(d_p)

    def rmse(err):
        return np.sqrt(np.mean(err**2))

    def linear_RMSE(i):
        return rmse(pxyz_fine[i] - np.interp(pxyz_fine[0], pxyz[0], pxyz[i]))

    def pchip_RMSE(i):
        return rmse(pxyz_fine[i] - pchip_interpolate(pxyz[0], pxyz[i], pxyz_fine[0]))

    axes[0].plot(pxyz[0], pxyz[1], '.-', label=f'RMSE linear: {linear_RMSE(1):.2f}')
    axes[1].plot(pxyz[0], pxyz[2], '.-', label=f'RMSE linear: {linear_RMSE(2):.2f}')


    axes[0].plot(pxyz_fine[0], pchip_interpolate(pxyz[0], pxyz[1], pxyz_fine[0]), '-',
                 label=f'RMSE pchip: {pchip_RMSE(1):.2f}')
    axes[1].plot(pxyz_fine[0], pchip_interpolate(pxyz[0], pxyz[2], pxyz_fine[0]), '-',
                 label=f'RMSE pchip: {pchip_RMSE(2):.2f}')
    setup_plot(ax=axes[0], ylabel='y [m]')
    setup_plot(ax=axes[1], ylabel='z [m]')
    plt.savefig(f'pchip {d_p}.png')
plt.show()
```

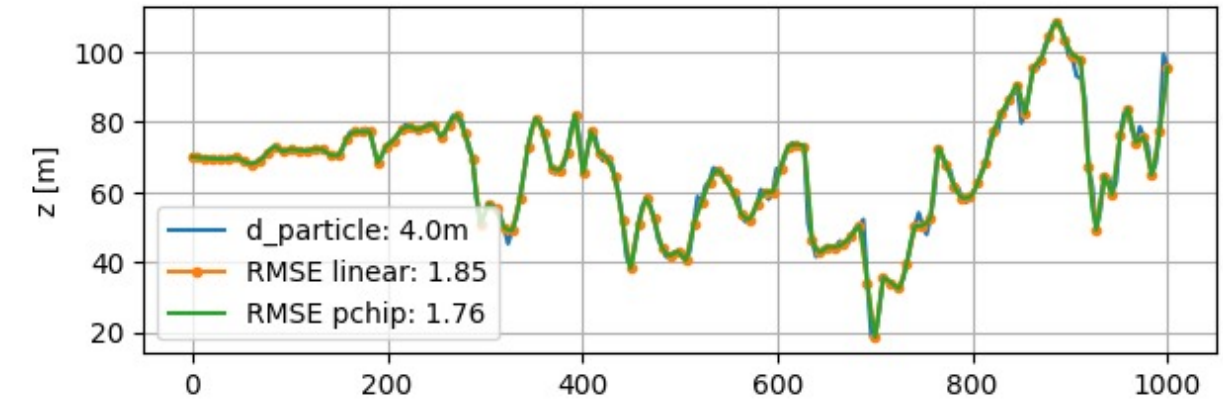
100% 10000/10000 [00:12<00:00, 732.90it/s]

100% 2000/2000 [00:02<00:00, 870.99it/s]

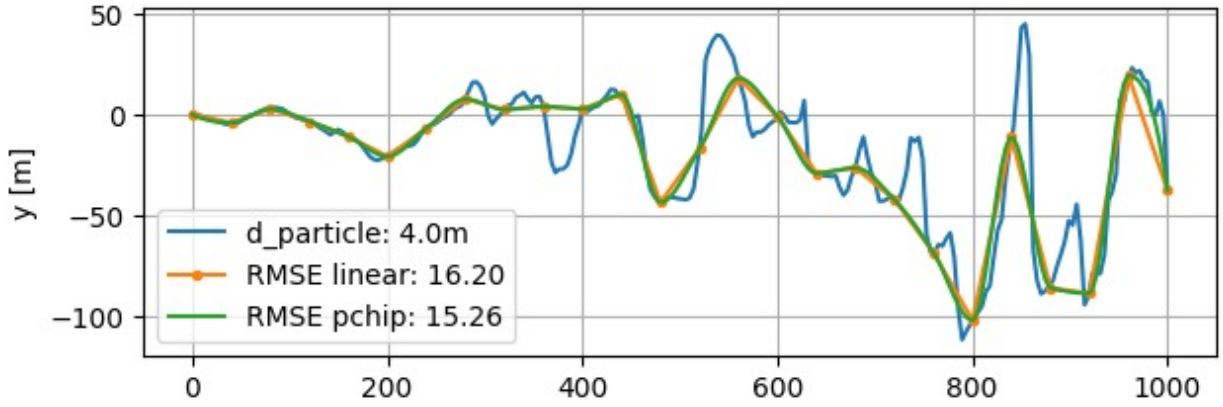
100% 1000/1000 [00:01<00:00, 833.89it/s]

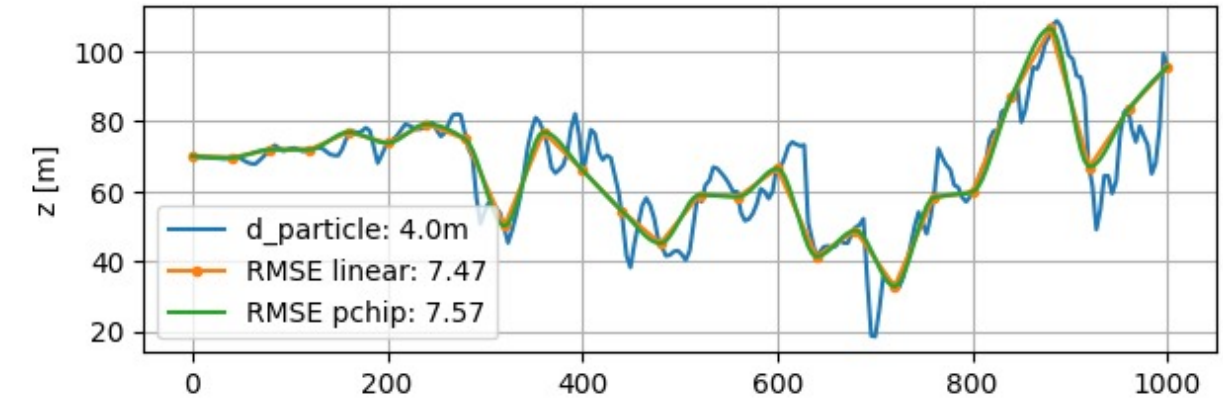




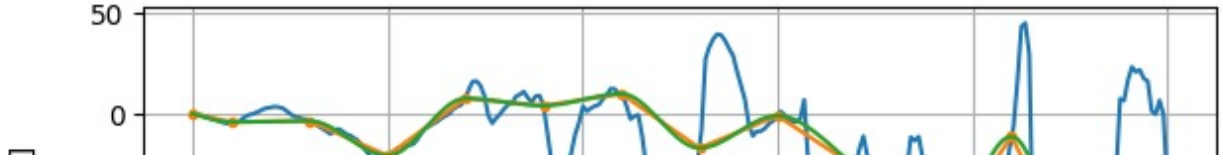


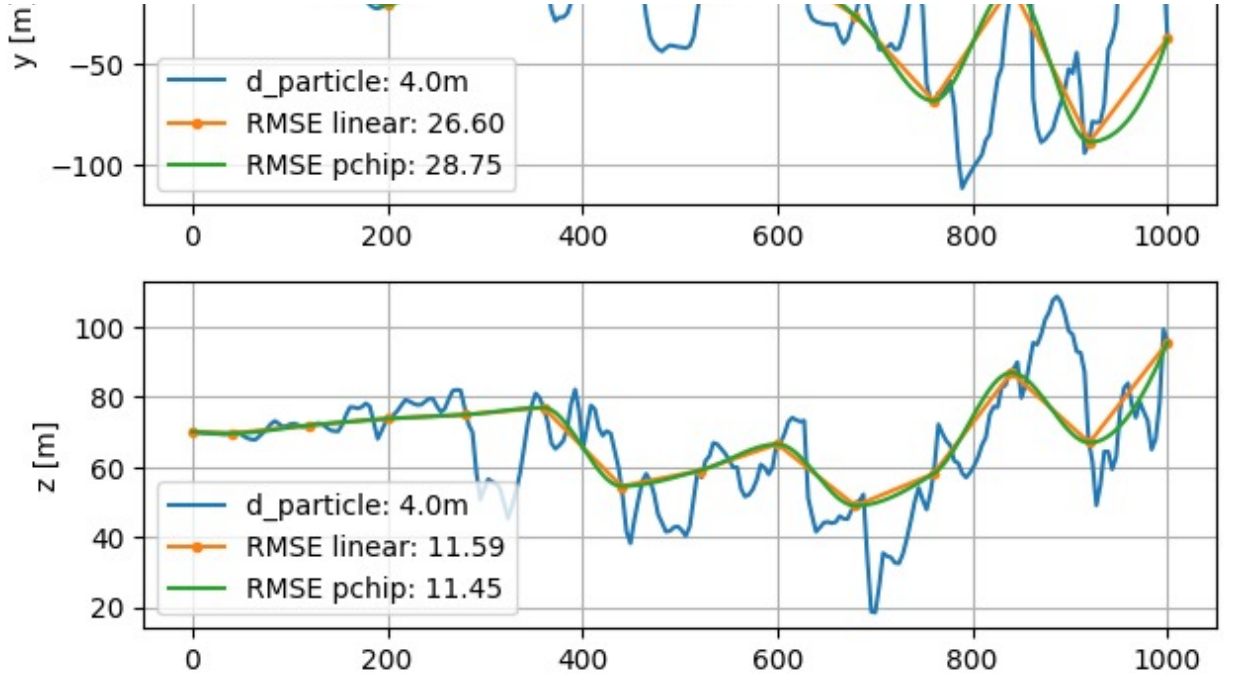
d_particle: 40.0m





d_particle: 80m





[ ]:

Previous