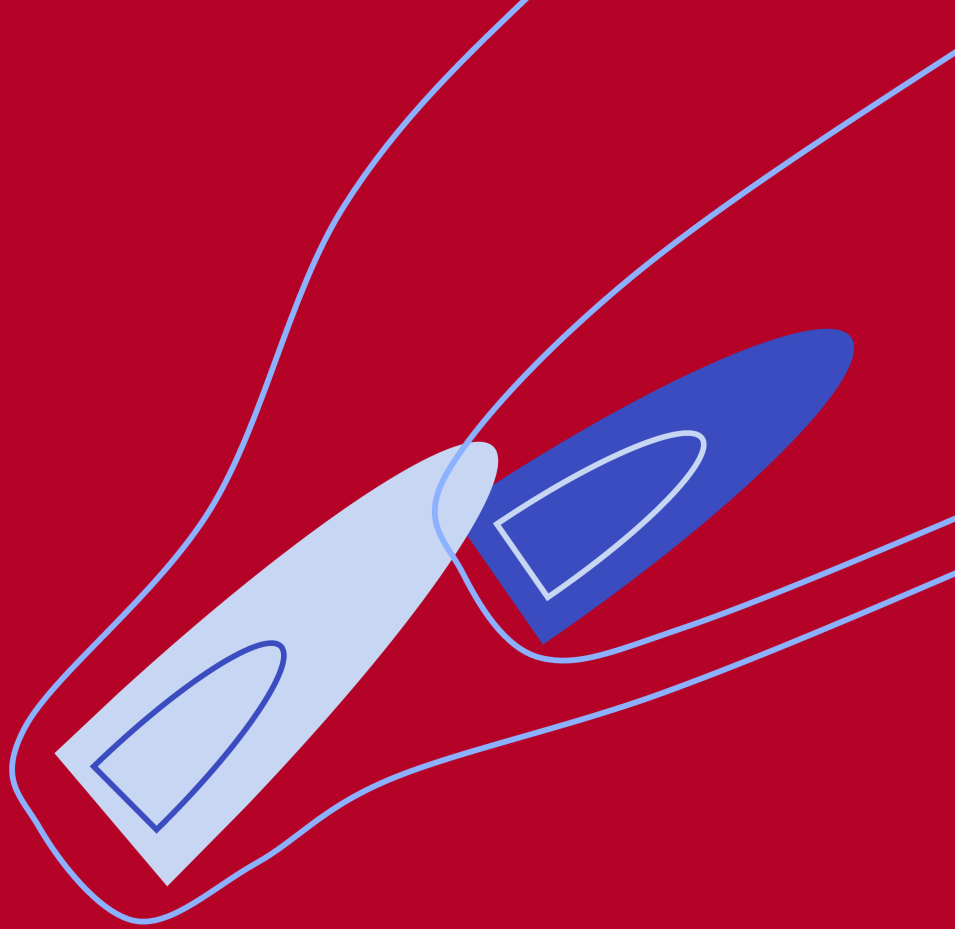
# FLORIS

Michael (Misha) Sinner

NAWEA/WindTech · October 2025



NREL







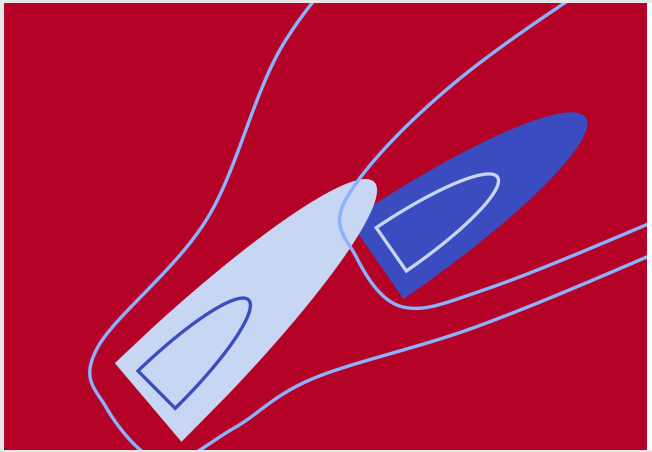

Wake models

Turbine models

Wind data

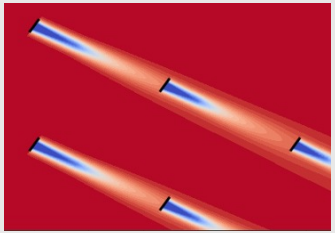
Design tools

# FLORIS



https://github.com/NREL/floris/

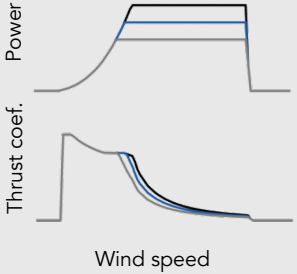
## Wake models



Flow velocity deficit models

- Jensen
- Gauss-Curl Hybrid
- Cumulative Curl
- TurbOPark
- Empirical Gaussian

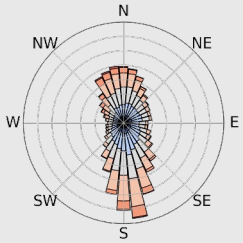
## Turbine models




Actuator disks with power, thrust coefficient curves

- Yaw misaligned
- Derating
- Peak shaving
- Active wake mixing
- Shut off

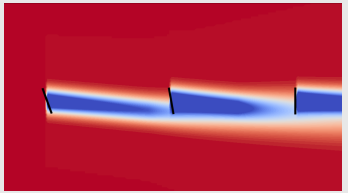
## Wind data




Vectorized input wind conditions

- Wind rose
- Time series
- Flow heterogeneity
- Data readers

## Design tools



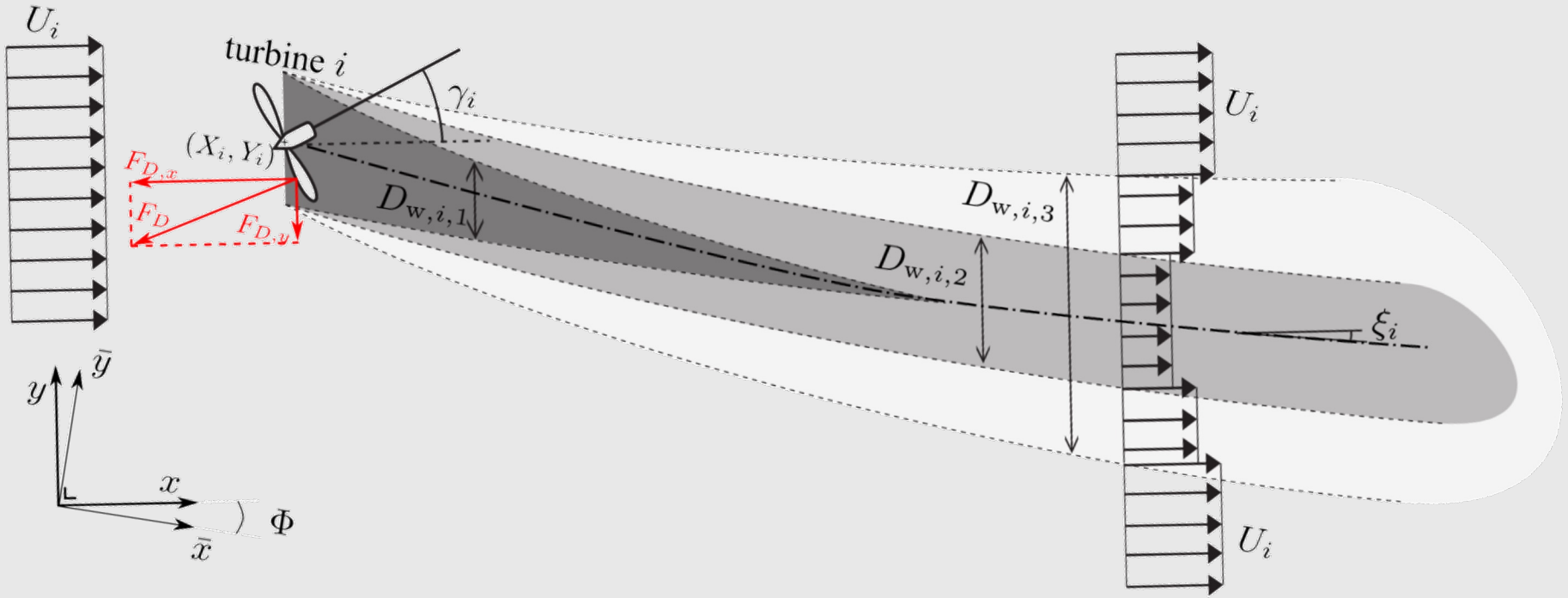
Optimization tools to help in the design and control of wind farms

- Yaw optimization
- Layout optimization

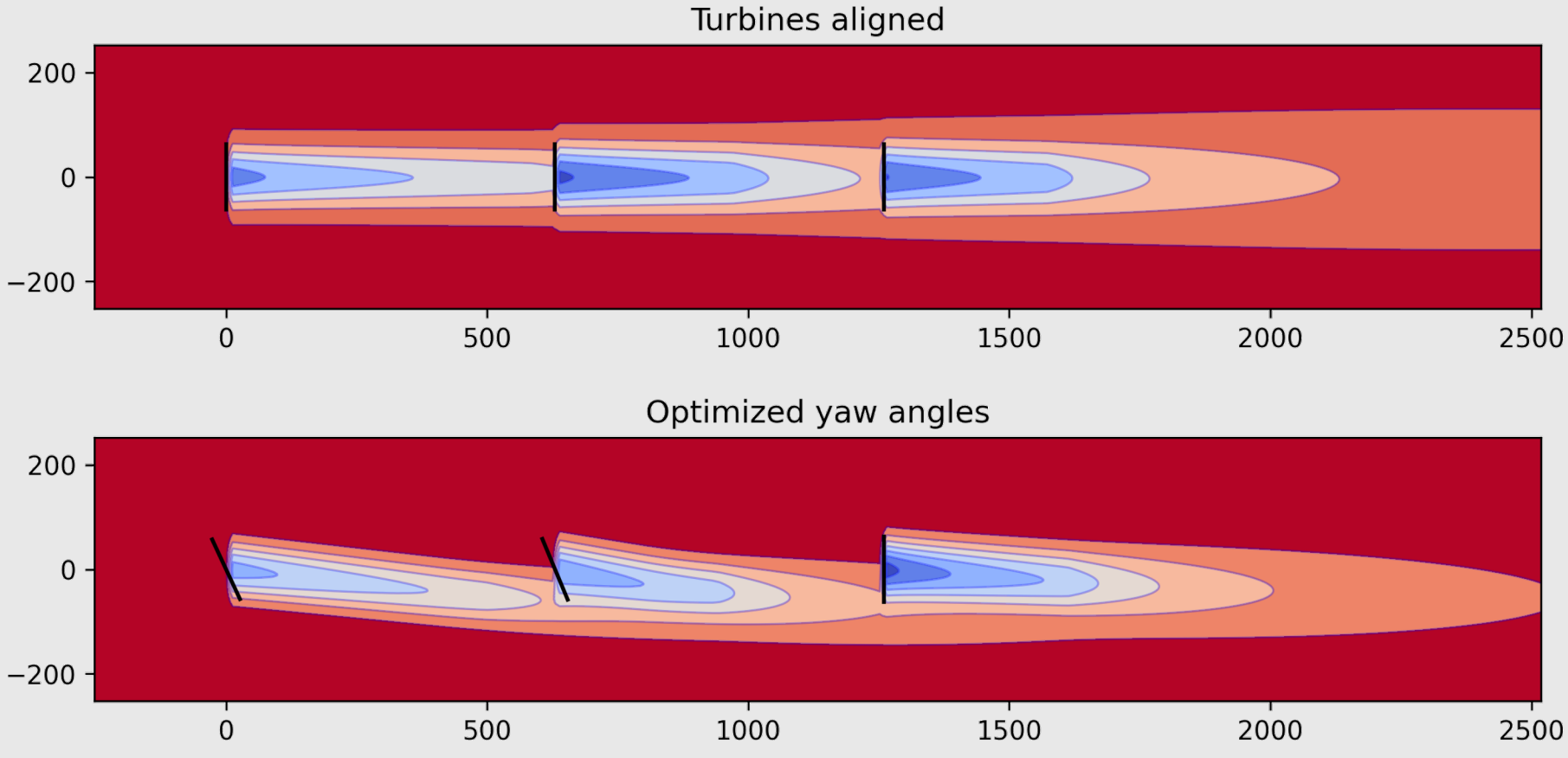
# FLORIS has many use cases

- Originally developed to simulate wake steering
- Controller development for experimental campaigns
- Tools added to perform layout design
- Integration into hybrid plant simulation and design tools
- Analysis techniques developed into standalone repositories (FLASC)

# What is wake steering?




Gebraad, P. M. O., *et al*. Wind plant power optimization through yaw control using a parametric model for wake effects—a CFD simulation study. *Wind Energy*, 2016.


Turbines aligned
200
0
−200
0
500
1000
1500
2000
2500
Optimized yaw angles
200
0
−200
0
500
1000
1500
2000
2500

# Wake steering is now out in the wild

Wind Energy. Sci., 2, 229–239, 2017
www.wind-energ-sci.net/2/229/2017/
doi:10.5194/wes-2-229-2017


## Field test of wake steering at an offshore wind farm

**Paul Fleming[1], Jennifer Annoni[2], Jigar J. Shah[2], Linpeng Wang[3], Shreyas Ananthan[2], Zhijun Zhang[3], Kyle Hutchings[1], Peng Wang[3], Weiguo Chen[3], and Lin Chen[3]**

[1]National Wind Technology Center, National Renewable Energy Laboratory, Golden, CO 80401, USA
[2]Research & Development, Envision Energy USA Ltd, Houston, TX 77002, USA
[3]Research & Development, Envision Energy Ltd, Shanghai, 200051, China

*Correspondence to:* Paul Fleming (paul.fleming@nrel.gov)

Received: 16 January 2017 – Discussion started: 6 February 2017
Revised: 4 April 2017 – Accepted: 7 April 2017 – Published: 8 May 2017

**Abstract.** In this paper, a field test of wake-steering control is presented. The field test is the result of a collaboration between the National Renewable Energy Laboratory (NREL) and Envision Energy, a smart energy management company and turbine manufacturer. In the campaign, an array of turbines within an operating commercial offshore wind farm in China have the normal yaw controller modified to implement wake steering according to a yaw control strategy. The strategy was designed using NREL wind farm models, including a computational fluid dynamics model, Simulator fOr Wind Farm Applications (SOWFA), for understanding wake dynamics and an engineering model, FLOw Redirection and Induction in Steady State (FLORIS), for yaw control optimization. Results indicate that, within the certainty afforded by the data, the wake-steering controller was successful in increasing power capture, by amounts similar to those predicted from the models.

### 1 Introduction

Wind farm control is an active field of research in which the controls of individual turbines co-located within a wind farm are coordinated to improve the overall performance of the farm. One objective of wind farm control is improving the power production of wind farms by accounting for the wake interactions between nearby turbines.

In one wind farm control concept, turbines are yawed to introduce a deflection of the wake away from downstream turbines. This method has been referred to as "controlling the wind" (Wagenaar et al., 2012) and "yaw-based wake steering" (Fleming et al., 2014b). High-fidelity simulations of wake steering have shown the potential of this technique. Jimenez et al. (2010) used computational fluid dynamics (CFD) simulations to demonstrate the wake deflection capability of wind turbines and provided a model of this deflection. In Fleming et al. (2014b), they used National Renewable Energy Laboratory (NREL)'s CFD-based Simulator fOr Wind Farm Applications (SOWFA) to investigate the capabilities of wind turbines to redirect wakes. In Vollmer et al. (2016), the behavior of wake steering in different atmospheric conditions was investigated, also using CFD. Finally, in Fleming et al. (2014a), simulations of two-turbine wind farms, again using SOWFA, were used to show that through wake steering, the net power of the two turbines is increased when the upstream turbine applies an intentional yaw misalignment.

Based on high-fidelity simulations, there appears to be good opportunities for improved power performance of wind farms with significant wake losses. Recent efforts have focused on the design of lower-fidelity, controller-oriented models, and controllers based on these models that use wake steering to actively improve power. In Gebraad et al. (2014), the FLOw Redirection and Induction in Steady State (FLORIS) model is described and used to determine optimal yaw settings for a model six-turbine wind farm. Set points for a particular wind speed and direction are determined by optimizing the yaw angles of the turbines using FLORIS, and these set points are used in SOWFA simulations. The results from SOWFA agree with the predictions from FLORIS, and total power capture is increased by 13 %. This work is carried further, and FLORIS is used to assess the overall improvement from control, first for one speed and over a wind

Published by Copernicus Publications on behalf of the European Academy of Wind Energy e.V.

PNAS

## Wind farm power optimization through wake steering

**Michael F. Howland[a], Sanjiva K. Lele[a,b], and John O. Dabiri[a,c,1]**

[a]Department of Mechanical Engineering, Stanford University, Stanford, CA 94305; [b]Department of Astronautics and Aeronautics, Stanford University, Stanford, CA 94305; and [c]Department of Civil and Environmental Engineering, Stanford University, Stanford, CA 94305

Edited by Alexis T. Bell, University of California, Berkeley, CA, and approved May 20, 2019 (received for review March 4, 2019)

**Global power production increasingly relies on wind farms to supply low-carbon energy. The recent Intergovernmental Panel on Climate Change (IPCC) Special Report predicted that renewable energy production must leap from 20% of the global energy mix in 2018 to 67% by 2050 to keep global temperatures from rising 1.5 °C above preindustrial levels. This increase requires reliable, low-cost energy production. However, wind turbines are often placed in close proximity within wind farms due to land and transmission line constraints, which results in wind farm efficiency degradation of up to 40% for wind directions aligned with columns of turbines. To increase wind farm power production, we developed a wake steering control scheme. This approach maximizes the power of a wind farm through yaw misalignment that deflects wakes away from downstream turbines. Optimization was performed with site-specific analytic gradient ascent relying on historical operational data. The protocol was tested in an operational wind farm in Alberta, Canada, resulting in statistically significant ($P < 0.05$) power increases of 7–13% for wind speeds near the site average and wind directions which occur during less than 10% of nocturnal operation and 28–47% for low wind speeds in the same wind directions. Wake steering also decreased the variability in the power production of the wind farm by up to 72%. Although the resulting gains in annual energy production were insignificant at this farm, these statistically significant wake steering results demonstrate the potential to increase the efficiency and predictability of power production through the reduction of wake losses.**

wind energy | turbulence | data science

The Intergovernmental Panel on Climate Change (IPCC) Special Report 15 on global warming (1) found that current rates of emissions will result in a temperature rise from preindustrial levels of 1.5 °C by 2040. Meanwhile, recent studies have predicted that the Paris Climate Agreement (2) will fail to keep warming below the stated goal of 2 °C (3, 4). The Special Report 15 found that coal-based electricity generation must decrease from contemporary rates of 40% of global energy production to 1–7%. As a result, renewable energy should compensate for this transition, increasing from 20% of energy generation in 2018 to 67% by 2050 (1). Wind and solar will likely comprise the bulk of these capacity additions due to their decreasing cost of electricity (5). While recent studies (5) have shown onshore wind energy to be economically favorable compared with coal and combined-cycle natural gas, such estimates are specific to sites with robust, reliable wind resource. To reach the goals of the Paris Climate Agreement, wind farms must significantly increase in number and density as well as extend to sites with less certain wind resource (6). As a result, methods to increase wind farm efficiency remain paramount to reducing carbon emissions.

While the major reason for decreased wind farm efficiency is variability in wind speed, aerodynamic losses within large arrays of turbines are also a key issue in wind farm operation (7). Due to the process of energy extraction from the atmospheric boundary layer, wind turbines necessarily produce a reduced momentum wake region immediately downstream (8). This wake will lower the power production of downstream turbines in the array. Wake power losses within a wind farm are a function of the incident wind speed and direction.

Wake losses occur when the wind speed is below the rated value (9) and turbines are at least partially aligned to the angle of the incoming wind. The mean wind speeds at the majority of wind farms are well below the rated value (10). Wind directions in the turbulent atmospheric boundary layer are inherently variable and will vary with the time of day, season, and other geophysical parameters (11). Wind farm layouts are designed to extract the maximum profit given historically observed wind direction and speed distributions, which typically results in larger streamwise turbine spacing in the most common wind directions. However, for offset wind directions, wind turbines are more closely spaced (12). In worst-case scenarios of wind turbine spacing and inflow directions in contemporary wind farms, there is an over 60% loss of efficiency when the wind shifts to a direction aligned with the columns of the turbines (13).

To minimize the aerodynamic losses between turbines under prevailing wind conditions, the optimal streamwise spacing has been found to be 10–15$D$, where $D$ is the turbine diameter (14–16). Modern turbines are increasing in size, with offshore turbines now above 200 m in rotor diameter (17). The corresponding spacing of turbines multiple kilometers apart significantly increases the cost of transmission lines and land use (18). As a result, wind farm designers are left with a complex multiobjective optimization problem which typically results in operational turbine spacing of 6 – 10$D$ (18). At this spacing, significant aerodynamic wake losses persist in modern wind farms when the flow is aligned with columns of turbines and the wind speed is below the rated value (13).

While the influence of wake losses on wind farm efficiency can be large for some inflow directions, the cumulative impact on the annual energy production of smaller wind farms is generally

**Significance**

**Wake effects within wind farms can significantly decrease the power production and increase the cost of electricity. Herein, we designed a wake steering control scheme to increase the power production of wind farms. The wake steering method was tested in an array of six utility-scale turbines where it increased the power production for wind speeds near the site annual average between 7% and 13% and decreased variability by up to 72%, for selected wind directions at night. These improvements can contribute to the increasing ability of wind farms to provide reliable, low-cost, and efficient base energy load.**

Author contributions: M.F.H. and J.O.D. designed research; M.F.H. performed research; M.F.H., S.K.L., and J.O.D. contributed new analytic tools; M.F.H., S.K.L., and J.O.D. analyzed data; and M.F.H., S.K.L., and J.O.D. wrote the paper.

The authors declare no conflict of interest.

This article is a PNAS Direct Submission.



Data deposition: The data from this paper have been deposited in the Stanford Digital Repository. https://purl.stanford.edu/mh827pg7681.

[1]To whom correspondence may be addressed. Email: jodabi@stanford.edu.

This article contains supporting information online at www.pnas.org/lookup/suppl/doi:10.1073/pnas.1903680116/-/DCSupplemental.

Published online July 1, 2019.





NAWEA WindTech 2019
Journal of Physics: Conference Series **1452** (2020) 012012 doi:10.1088/1742-6596/1452/1/012012

IOP Publishing

## Field Validation of Wake Steering Control with Wind Direction Variability

**Eric Simley, Paul Fleming, Jennifer King**
National Wind Technology Center, National Renewable Energy Laboratory, Golden, CO, 80401, USA
E-mail: eric.simley@nrel.gov

**Abstract.** Wake steering is a wind farm control strategy wherein upstream turbines are misaligned with the wind direction to redirect their wakes away from downstream turbines, increasing overall wind plant power. Wake steering is often analyzed assuming steady mean wind directions across the wind farm. However, in practice, the wind-direction varies considerably over time because of large-scale weather phenomena. Wind direction variability causes the increase

Wind Energ. Sci., 4, 273–285, 2019
https://doi.org/10.5194/wes-4-273-2019


## Initial results from a field campaign of wake steering applied at a commercial wind farm – Part 1

**Paul Fleming[1], Jennifer King[1], Katherine Dykes[1], Eric Simley[1], Jason Roadman[1], Andrew Scholbrock[1], Patrick Murphy[1,*], Julie K. Lundquist[2,3], Patrick Moriarty[1], Katherine Fleming[1], Jeroen van Dam[1], Christopher Bay[1], Rafael Mudafort[1], Hector Lopez[2], Jason Skopek[2], Michael Scott[2], Brady Ryan[2], Charles Guernsey[2], and Dan Brake[2]**

[1]National Wind Technology Center, National Renewable Energy Laboratory, Golden, CO 80401, USA
[2]NextEra Energy Resources, 700 Universe Blvd, Juno Beach, FL 33408
[3]Dept. Atmospheric and Oceanic Sciences, University of Colorado Boulder, Boulder, CO 80303, USA

**Correspondence:** Paul Fleming (paul.fleming@nrel.gov)

Received: 1 February 2019 – Discussion started: 18 February 2019
Revised: 26 April 2019 – Accepted: 6 May 2019 – Published: 20 May 2019

**Abstract.** Wake steering is a form of wind farm control in which turbines use yaw offsets to affect wakes in order to yield an increase in total energy production. In this first phase of a study of wake steering at a commercial wind farm, two turbines implement a schedule of offsets. Results exploring the observed performance of wake

Wind Energ. Sci., 6, 1521–1531, 2021
https://doi.org/10.5194/wes-6-1521-2021


## Experimental results of wake steering using fixed angles

**Paul Fleming[1], Michael Sinner[1], Tom Young[1], Marine Lannic[1], Jennifer King[1], Eric Simley[1], and Bart Doekemeijer[2]**

[1]National Wind Technology Center, National Renewable Energy Laboratory, Golden, CO, 80401, USA
[2]RES Group, Beaufort Court, Egg Farm Lane, Kings Langley, Hertfordshire, WD4 8LR, UK

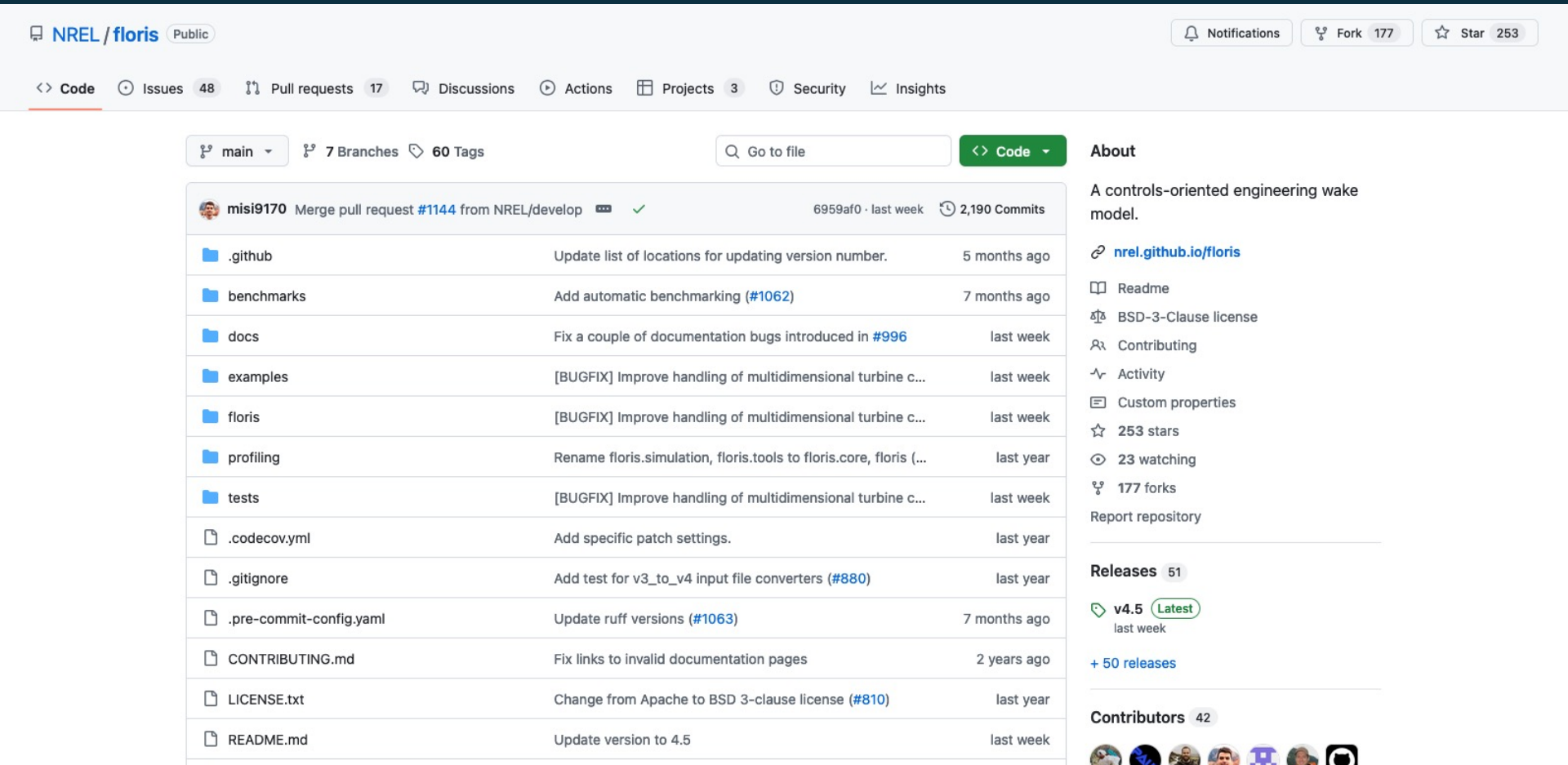
**Correspondence:** Paul Fleming (paul.fleming@nrel.gov)

Received: 16 April 2021 – Discussion started: 19 May 2021
Revised: 1 September 2021 – Accepted: 11 October 2021 – Published: 7 December 2021

**Abstract.** In this article, the authors present a test of wake steering at a commercial wind farm. A single fixed yaw offset, rather than an optimized offset schedule, is alternately applied to an upstream wind turbine, and the effect on downstream turbines is analyzed. This experimental design allows for comparison with engineering wake models independent of the controller's ability to track a varying offset and correctly measure wind direction. Additionally, by applying the same offset in beneficial and detrimental conditions, we are able to collect important data for assessing second-order wake model predictions. Results of the article from collected data show good agreement with the FLOw Redirection and Induction in Steady State (FLORIS) engineering model and offer support for the asymmetry of wake steering predicted by newer models, such as the Gauss–curl hybrid

# FLORIS software

# FLORIS is available on github.com



NREL / floris Public

Notifications | Fork 177 | Star 253

Code | Issues 48 | Pull requests 17 | Discussions | Actions | Projects 3 | Security | Insights

main | 7 Branches | 60 Tags | Go to file | Code

misi9170 Merge pull request #1144 from NREL/develop — 6959af0 · last week — 2,190 Commits

| | | |
|---|---|---|
| .github | Update list of locations for updating version number. | 5 months ago |
| benchmarks | Add automatic benchmarking (#1062) | 7 months ago |
| docs | Fix a couple of documentation bugs introduced in #996 | last week |
| examples | [BUGFIX] Improve handling of multidimensional turbine c... | last week |
| floris | [BUGFIX] Improve handling of multidimensional turbine c... | last week |
| profiling | Rename floris.simulation, floris.tools to floris.core, floris (... | last year |
| tests | [BUGFIX] Improve handling of multidimensional turbine c... | last week |
| .codecov.yml | Add specific patch settings. | last year |
| .gitignore | Add test for v3_to_v4 input file converters (#880) | last year |
| .pre-commit-config.yaml | Update ruff versions (#1063) | 7 months ago |
| CONTRIBUTING.md | Fix links to invalid documentation pages | 2 years ago |
| LICENSE.txt | Change from Apache to BSD 3-clause license (#810) | last year |
| README.md | Update version to 4.5 | last week |

About

A controls-oriented engineering wake model.

nrel.github.io/floris

Readme
BSD-3-Clause license
Contributing
Activity
Custom properties
253 stars
23 watching
177 forks
Report repository

Releases 51

v4.5 Latest
last week

+ 50 releases

Contributors 42

## ... and can be readily cloned

```
> git clone https://github.com/NREL/floris
> pip install -e floris
```

## or installed directly from PyPI

```
> pip install floris
```

`python` Open-source high-level programing language

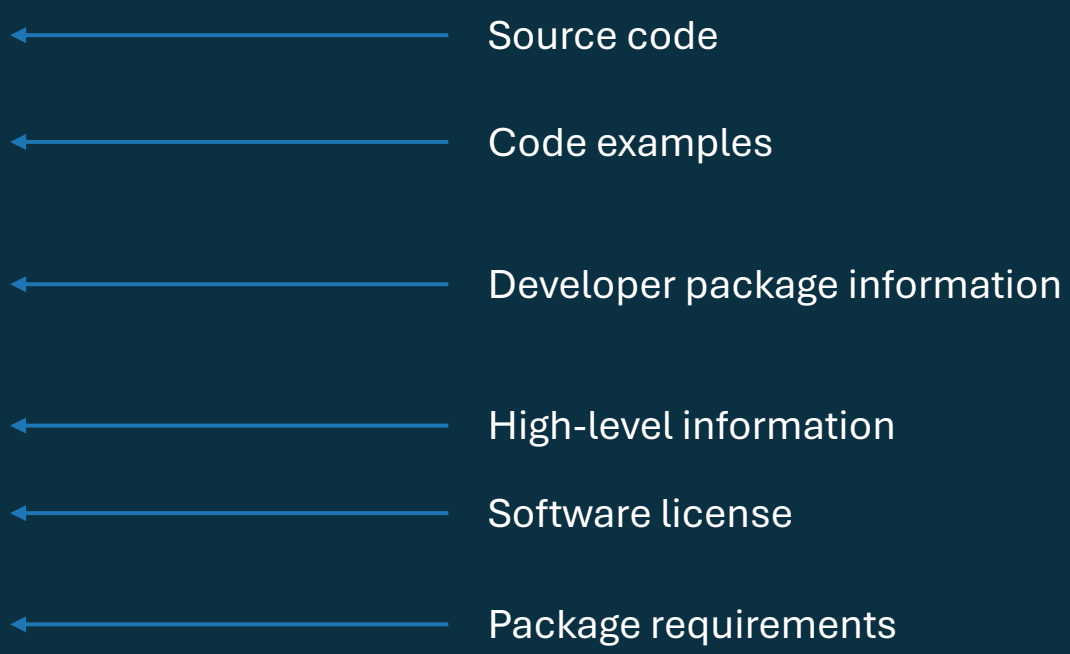
`conda` Software environment manager

`pip` Software package manager

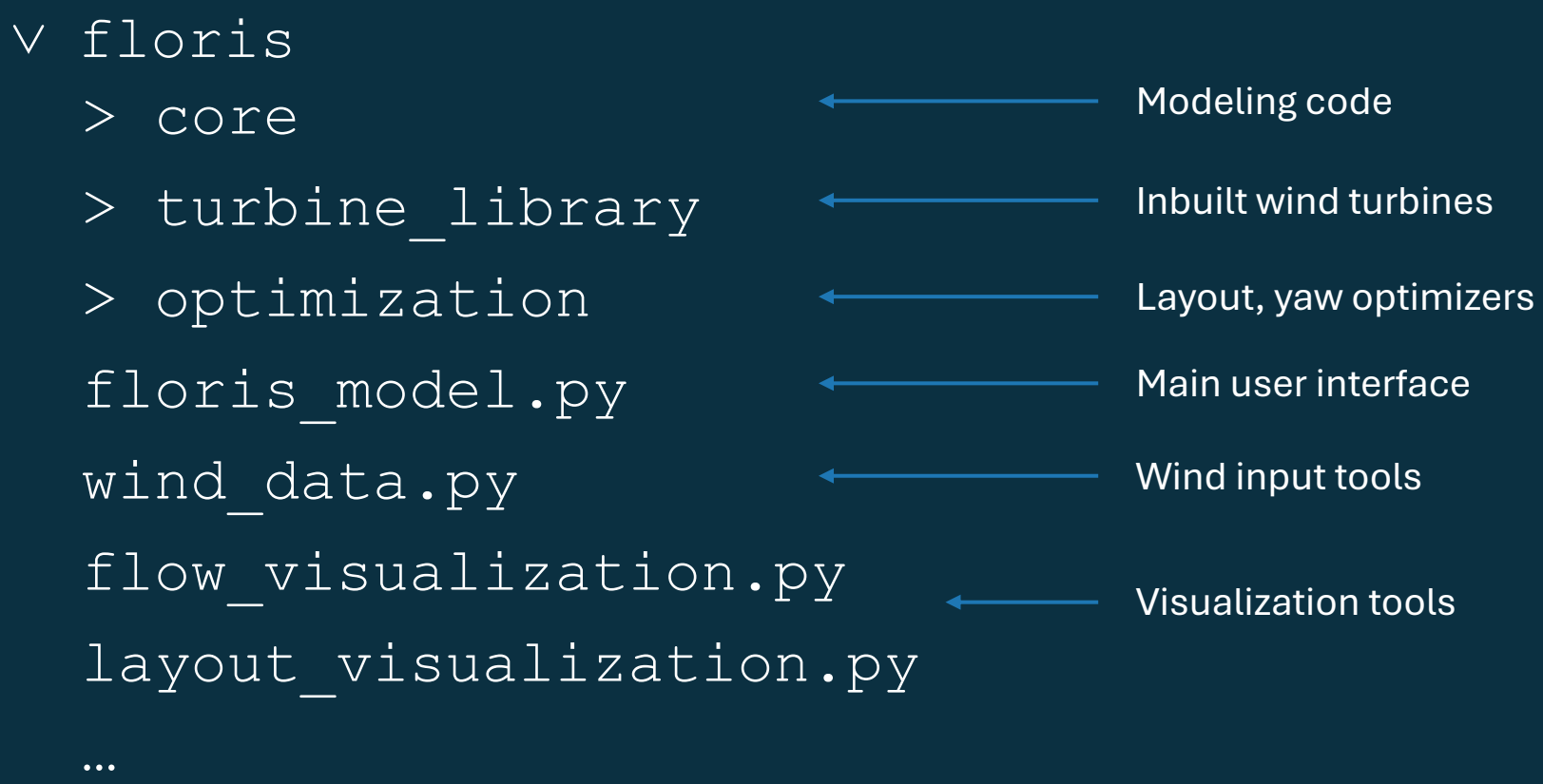
`git` Text-based software version control

`github` Online location for code sharing a cooperative programing

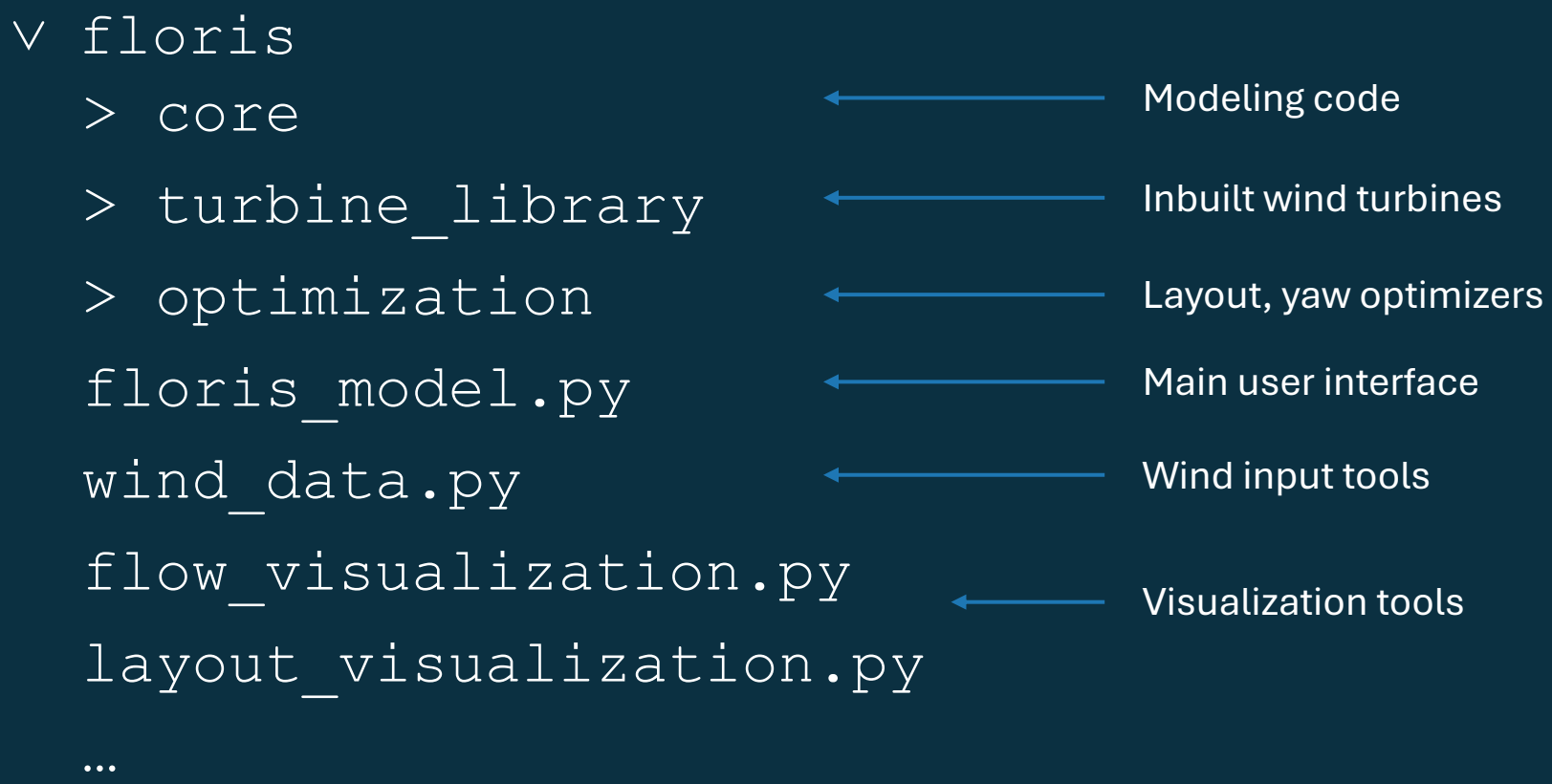
# FLORIS follows a typical python package structure



# FLORIS follows a typical python package structure

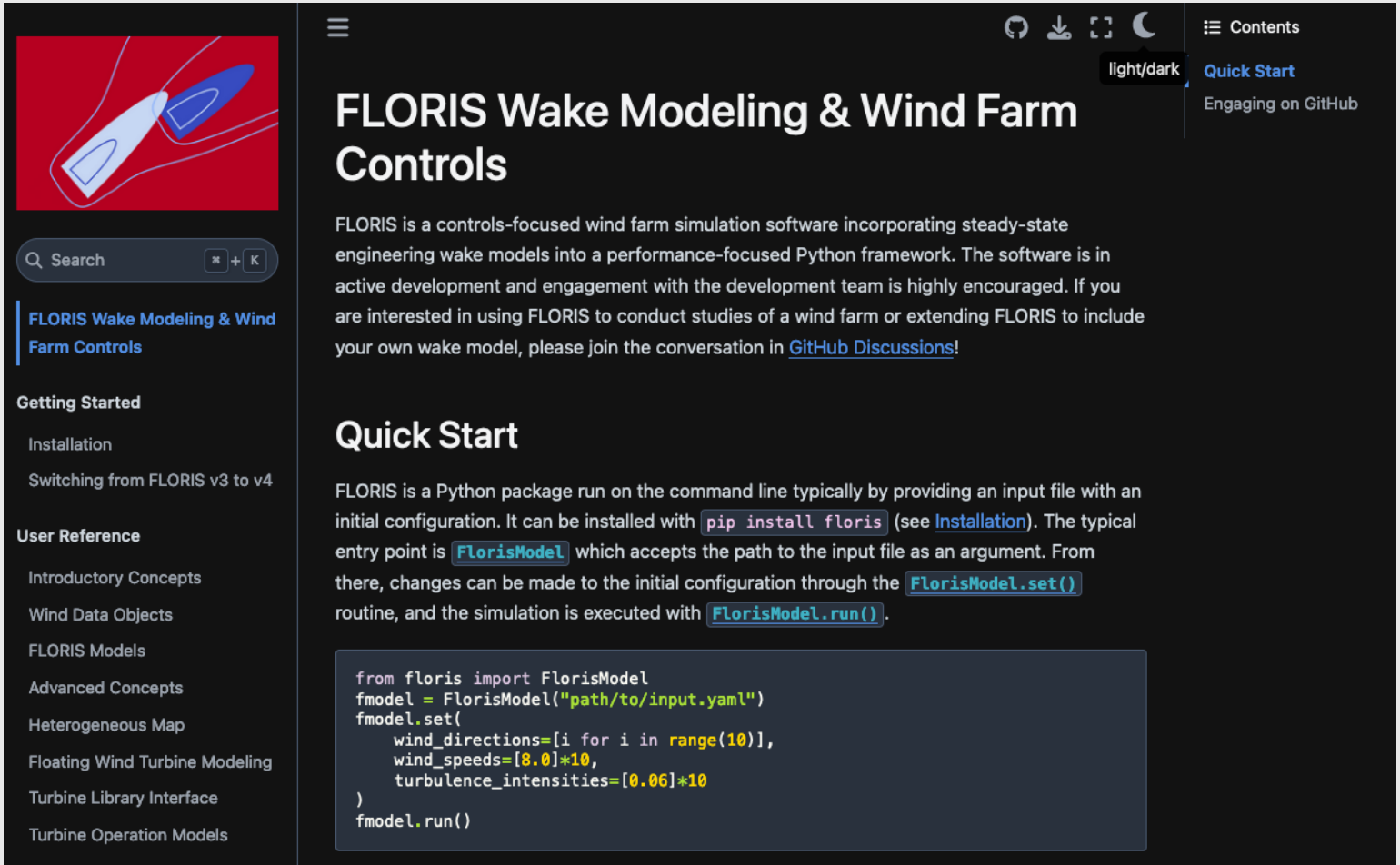


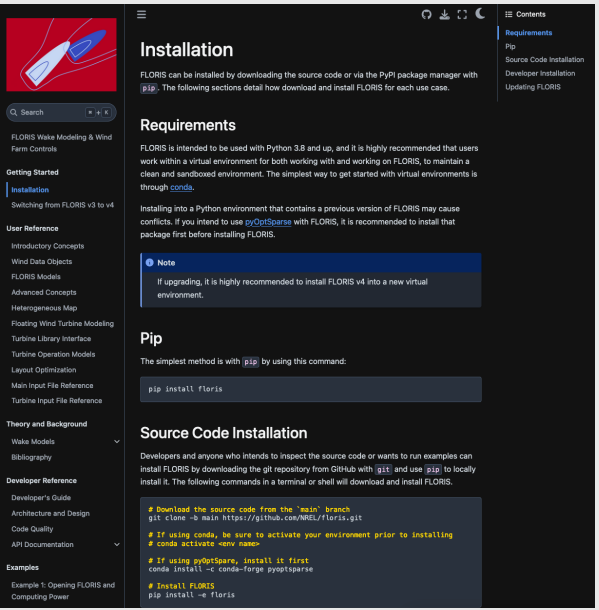
# FLORIS follows a typical python package structure

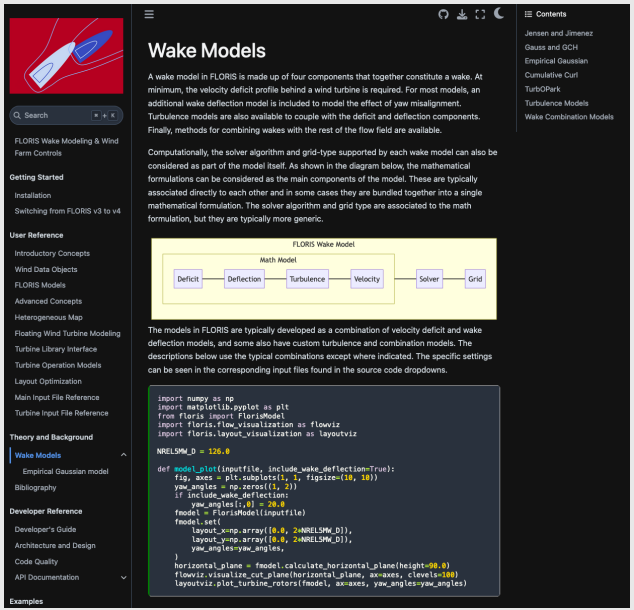


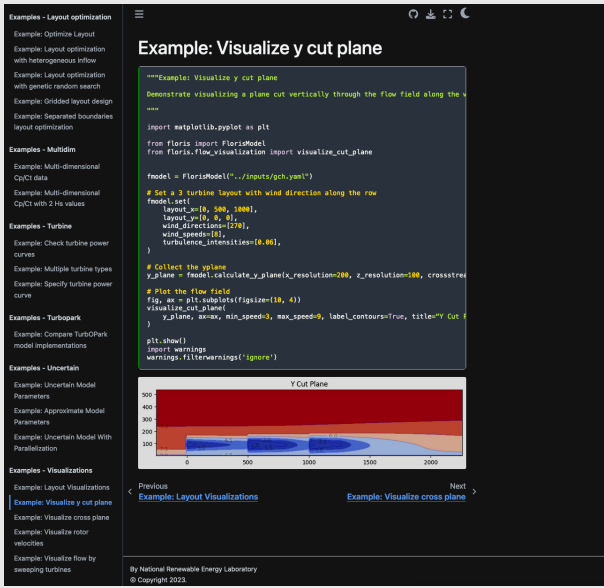
# FLORIS documentation

# Documentation at nrel.github.io/floris/




Search
FLORIS Wake Modeling & Wind Farm Controls
Getting Started
Installation
Switching from FLORIS v3 to v4
User Reference
Introductory Concepts
Wind Data Objects
FLORIS Models
Advanced Concepts
Heterogeneous Map
Floating Wind Turbine Modeling
Turbine Library Interface
Turbine Operation Models
Layout Optimization
Main Input File Reference
Turbine Input File Reference
Theory and Background
Wake Models
Bibliography
Developer Reference
Developer's Guide
Architecture and Design
Code Quality
API Documentation
Examples
Example 1: Opening FLORIS and Computing Power
Contents
Requirements
Pip
Source Code Installation
Developer Installation
Updating FLORIS
Installation
FLORIS can be installed by downloading the source code or via the PyPI package manager with pip. The following sections detail how download and install FLORIS for each use case.
Requirements
FLORIS is intended to be used with Python 3.8 and up, and it is highly recommended that users work within a virtual environment for both working with and working on FLORIS, to maintain a clean and sandboxed environment. The simplest way to get started with virtual environments is through conda.
Installing into a Python environment that contains a previous version of FLORIS may cause conflicts. If you intend to use pyOptSparse with FLORIS, it is recommended to install that package first before installing FLORIS.
Note
If upgrading, it is highly recommended to install FLORIS v4 into a new virtual environment.
Pip
The simplest method is with pip by using this command:
pip install floris
Source Code Installation
Developers and anyone who intends to inspect the source code or wants to run examples can install FLORIS by downloading the git repository from GitHub with git and use pip to locally install it. The following commands in a terminal or shell will download and install FLORIS.
# Download the source code from the `main` branch
git clone -b main https://github.com/NREL/floris.git
# If using conda, be sure to activate your environment prior to installing
# conda activate <env name>
# If using pyOptSpare, install it first
conda install -c conda-forge pyoptsparse
# Install FLORIS
pip install -e floris

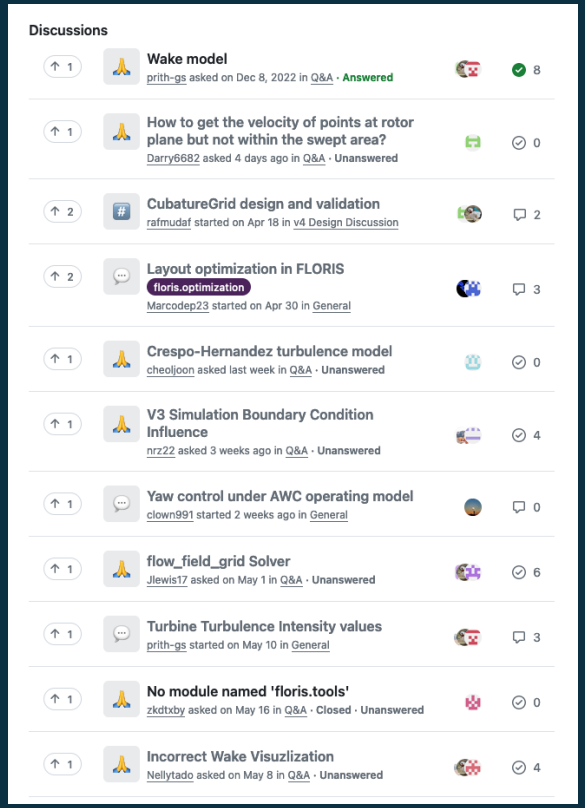

Search
FLORIS Wake Modeling & Wind Farm Controls
Getting Started
Installation
Switching from FLORIS v3 to v4
User Reference
Introductory Concepts
Wind Data Objects
FLORIS Models
Advanced Concepts
Heterogeneous Map
Floating Wind Turbine Modeling
Turbine Library Interface
Turbine Operation Models
Layout Optimization
Main Input File Reference
Turbine Input File Reference
Theory and Background
Wake Models
Empirical Gaussian model
Bibliography
Developer Reference
Developer's Guide
Architecture and Design
Code Quality
API Documentation
Examples
Contents
Jensen and Jimenez
Gauss and GCH
Empirical Gaussian
Cumulative Curl
TurboPark
Turbulence Models
Wake Combination Models
Wake Models
A wake model in FLORIS is made up of four components that together constitute a wake. At minimum, the velocity deficit profile behind a wind turbine is required. For most models, an additional wake deflection model is included to model the effect of yaw misalignment. Turbulence models are also available to couple with the deficit and deflection components. Finally, methods for combining wakes with the rest of the flow field are available.
Computationally, the solver algorithm and grid-type supported by each wake model can also be considered as part of the model itself. As shown in the diagram below, the mathematical formulations can be considered as the main components of the model. These are typically associated directly to each other and in some cases they are bundled together into a single mathematical formulation. The solver algorithm and grid type are associated to the math formulation, but they are typically more generic.
FLORIS Wake Model
Math Model
Deficit
Deflection
Turbulence
Velocity
Solver
Grid
The models in FLORIS are typically developed as a combination of velocity deficit and wake deflection models, and some also have custom turbulence and combination models. The descriptions below use the typical combinations except where indicated. The specific settings can be seen in the corresponding input files found in the source code dropdowns.
import numpy as np
import matplotlib.pyplot as plt
from floris import FlorisModel
import floris.flow_visualization as flowviz
import floris.layout_visualization as layoutviz
NREL5MW_D = 126.0
def model_plot(inputfile, include_wake_deflection=True):
fig, axes = plt.subplots(1, 1, figsize=(10, 10))
yaw_angles = np.zeros((1, 2))
if include_wake_deflection:
yaw_angles[:,0] = 20.0
fmodel = FlorisModel(inputfile)
fmodel.set(
layout_x=np.array([0.0, 2*NREL5MW_D]),
layout_y=np.array([0.0, 2*NREL5MW_D]),
yaw_angles=yaw_angles,
)
horizontal_plane = fmodel.calculate_horizontal_plane(height=90.0)
flowviz.visualize_cut_plane(horizontal_plane, ax=axes, clevels=100)
layoutviz.plot_turbine_rotors(fmodel, ax=axes, yaw_angles=yaw_angles)


Examples – Layout optimization
Example: Optimize Layout
Example: Layout optimization with heterogeneous inflow
Example: Layout optimization with genetic random search
Example: Gridded layout design
Example: Separated boundaries layout optimization
Examples – Multidim
Example: Multi-dimensional Cp/Ct data
Example: Multi-dimensional Cp/Ct with 2 Hs values
Examples – Turbine
Example: Check turbine power curves
Example: Multiple turbine types
Example: Specify turbine power curve
Examples – Turbopark
Example: Compare TurbOPark model implementations
Examples – Uncertain
Example: Uncertain Model Parameters
Example: Approximate Model Parameters
Example: Uncertain Model With Parallelization
Examples – Visualizations
Example: Layout Visualizations
Example: Visualize y cut plane
Example: Visualize cross plane
Example: Visualize rotor velocities
Example: Visualize flow by sweeping turbines
Example: Visualize y cut plane
"""Example: Visualize y cut plane
Demonstrate visualizing a plane cut vertically through the flow field along the
"""
import matplotlib.pyplot as plt
from floris import FlorisModel
from floris.flow_visualization import visualize_cut_plane
fmodel = FlorisModel("../inputs/gch.yaml")
# Set a 3 turbine layout with wind direction along the row
fmodel.set(
layout_x=[0, 500, 1000],
layout_y=[0, 0, 0],
wind_directions=[270],
wind_speeds=[8],
turbulence_intensities=[0.06],
)
# Collect the yplane
y_plane = fmodel.calculate_y_plane(x_resolution=200, z_resolution=100, crossstre
# Plot the flow field
fig, ax = plt.subplots(figsize=(10, 4))
visualize_cut_plane(
y_plane, ax=ax, min_speed=3, max_speed=9, label_contours=True, title="Y Cut
)
plt.show()
import warnings
warnings.filterwarnings('ignore')
Y Cut Plane
500
400
300
200
100
0
500
1000
1500
2000
Previous
Example: Layout Visualizations
Next
Example: Visualize cross plane
By National Renewable Energy Laboratory
© Copyright 2023.

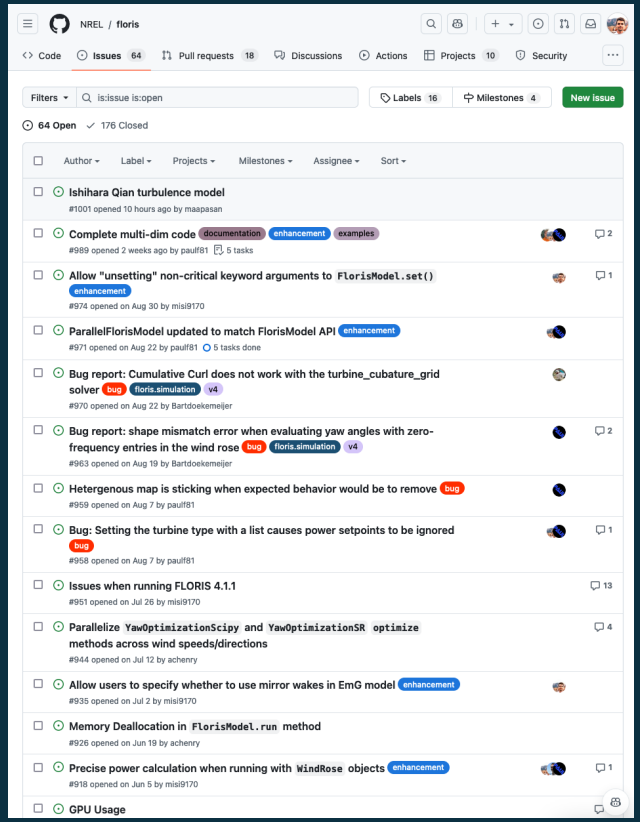
# Open-source software community

# We support and encourage interaction on github

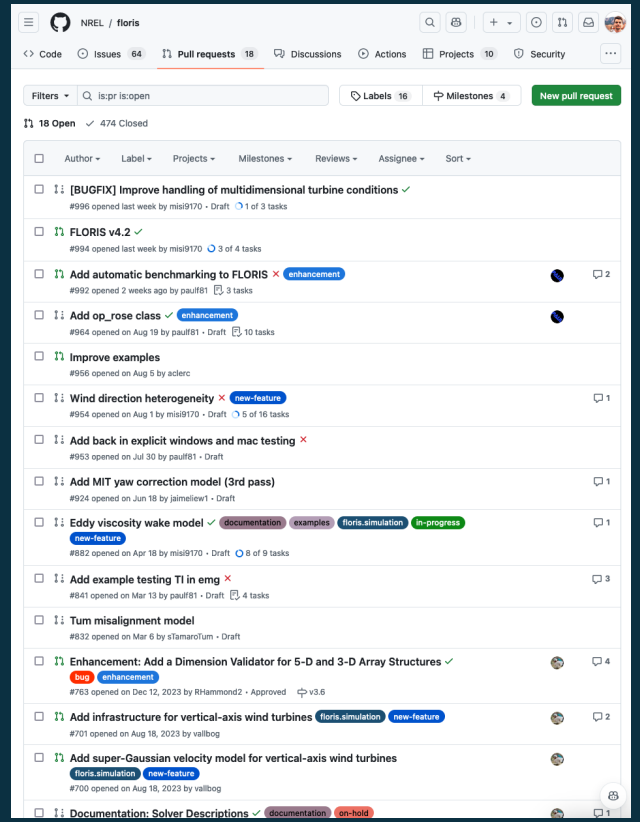
## Discussion forum




## Issues




## Pull requests



# Basic FLORIS usage

```python
import numpy as np
from floris import FlorisModel, TimeSeries

# Load the Floris model
fmodel = FlorisModel("inputs/gch.yaml")

# Set up inflow wind conditions
time_series = TimeSeries(
    wind_directions=270 + 30 * np.random.randn(100),
    wind_speeds=8 + 2 * np.random.randn(100),
    turbulence_intensities=0.06 + 0.02 * np.random.randn(100),
)

# Set the wind conditions for the model
fmodel.set(wind_data=time_series)

# Run the calculations
fmodel.run()

# Extract turbine and farm powers
turbine_powers = fmodel.get_turbine_powers() / 1000.0
farm_power = fmodel.get_farm_power() / 1000.0

print(turbine_powers.shape)
print(farm_power.shape)

# # Output:
# (100, 3)
# (100,)
```

Input file contains wake model parameters and specifies turbine to use

Wind data objects (`TimeSeries`, `WindRose`, `WindTIRose`, etc) conveniently package inflow conditions

Set inflow conditions, farm layout, control setpoints, etc. (replaces `reinitialize()`)

Execute solve, takes no inputs (replaces `calculate_wake()`)

Extract outputs after solve

# `FlorisModel` is the main user interface

```
fmodel = FlorisModel("inputs/gch.yaml")
```

```
class FlorisModel():
    def __init__():
        ...
    def set():
        ...
    def run():
        ...
    def get_turbine_powers():
```

Class definition

“Constructor”

Set up the FLORIS solve

Run the FLORIS wake solver

Access outputs

# Symmetry between `FlorisModels`

`FlorisModel`

Basic class for running FLORIS calculations

`UncertainFlorisModel`

Class for running calculations under uncertainty

`ParFlorisModel`

Class for running calculations with parallel computing

```yaml
name: GCH
description: Three turbines using Gauss Curl Hybrid model
floris_version: v4

logging:
  console:
    enable: true
    level: WARNING
  file:
    enable: false
    level: WARNING

solver:
  type: turbine_grid
  turbine_grid_points: 3

farm:
  layout_x:
  - 0.0
  - 630.0
  layout_y:
  - 0.0
  - 0.0
  turbine_type:
  - nrel_5MW
  - iea_10MW

flow_field:
  air_density: 1.225
  reference_wind_height: 90.0 # Since multiple defined turbines, must s
  turbulence_intensities:
  - 0.06
  wind_directions:
  - 270.0
  wind_shear: 0.12
  wind_speeds:
  - 8.0
  wind_veer: 0.0

wake:
  model_strings:
    combination_model: sosfs
    deflection_model: gauss
    turbulence_model: crespo_hernandez
    velocity_model: gauss

  enable_secondary_steering: false
  enable_yaw_added_recovery: false
  enable_transverse_velocities: false
  enable_active_wake_mixing: false
```

Documentation

Logging

Grid points

Farm details

Inflow details

Wake model selection

Deflection parameters

Deficit parameters

Turbulence parameters

```yaml
  wake_deflection_parameters:
    gauss:
      ad: 0.0
      alpha: 0.58
      bd: 0.0
      beta: 0.077
      dm: 1.0
      ka: 0.38
      kb: 0.004
    jimenez:
      ad: 0.0
      bd: 0.0
      kd: 0.05

  wake_velocity_parameters:
    cc:
      a_s: 0.179367259
      b_s: 0.0118889215
      c_s1: 0.0563691592
      c_s2: 0.13290157
      a_f: 3.11
      b_f: -0.68
      c_f: 2.41
      alpha_mod: 1.0
    gauss:
      alpha: 0.58
      beta: 0.077
      ka: 0.38
      kb: 0.004
    jensen:
      we: 0.05

  wake_turbulence_parameters:
    crespo_hernandez:
      initial: 0.1
      constant: 0.5
      ai: 0.8
      downstream: -0.32
```

Anything can be set dynamically, too!

```yaml
# Data based on:
# https://github.com/IEAWindTask37/IEA-15-240-RWT/blob/master/
# IEA-15-240-RWT_tabular.xlsx
# Note: Small power variations above rated removed.
# Generator efficiency of 100% used.
turbine_type: 'iea_15MW'
hub_height: 150.0
rotor_diameter: 242.24
TSR: 8.0
operation_model: cosine-loss
power_thrust_table:
  ref_air_density: 1.225
  ref_tilt: 6.0
  cosine_loss_exponent_yaw: 1.88
  cosine_loss_exponent_tilt: 1.88
  helix_a: 1.809
  helix_power_b: 4.828e-03
  helix_power_c: 4.017e-11
  helix_thrust_b: 1.390e-03
  helix_thrust_c: 5.084e-04
  power:
    - 0.000000
    - 0.000000
    - 42.733312
    - 292.585981
    - 607.966543
    - 981.097693
    - 1401.98084
    - 1858.67086
    - 2337.575997
    - 2824.097302
    - 3303.06456
    - 3759.432328
    - 4178.637714
    - 4547.19121
    - 4855.342682
    - 5091.537139
    - 5248.453137
    - 5320.793207
    - 5335.345498
    - 5437.90563
    - 5631.253025
    - 5920.980626
    - 6315.115602
    - 6824.470067
    - 7462.846389
    - 8238.359448
    - 9167.96703
```

Documentation

Physical characteristics

Operation model

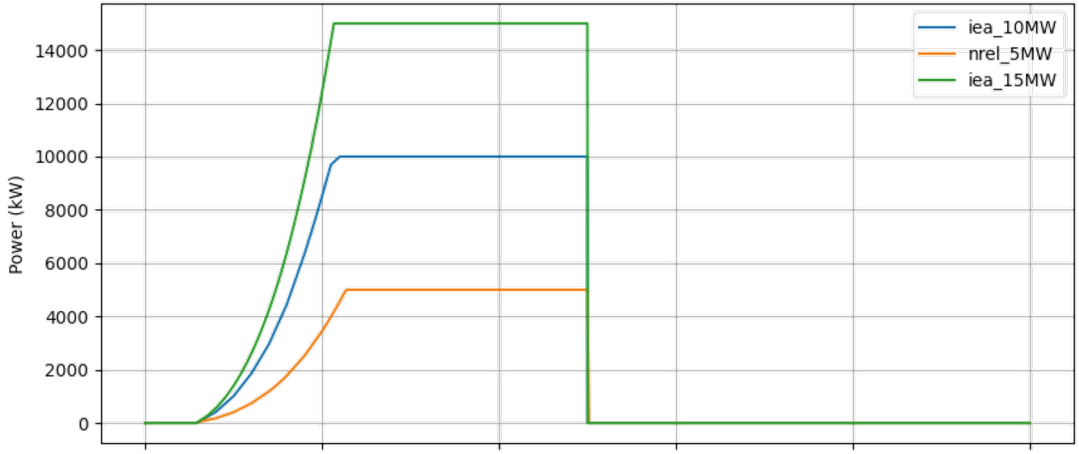
Power/thrust curve metadata

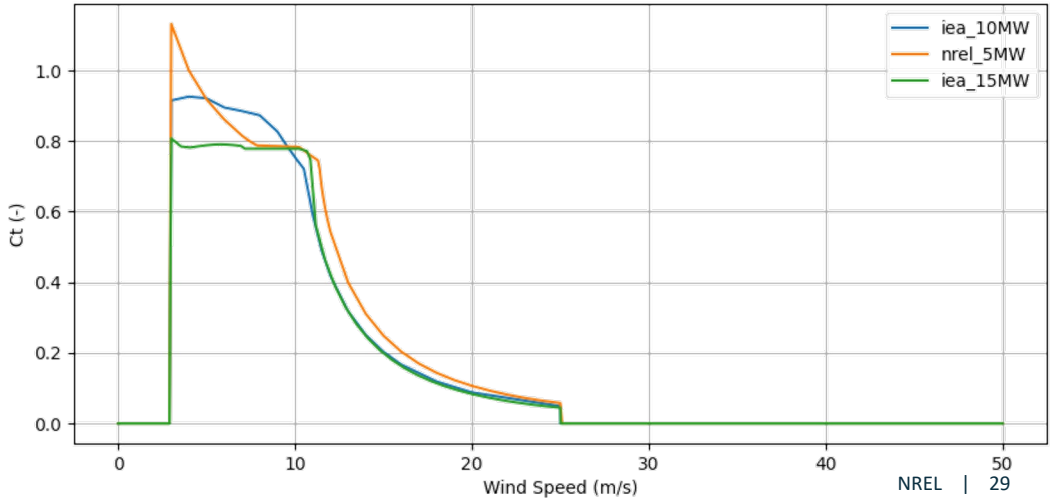
Power/thrust curve definition

# Wind turbine models

# Wind turbines are modeled as actuator disks

- Basic geometric parameters (hub height, rotor diameter)
- Power (kW) and thrust coefficient (-) curves



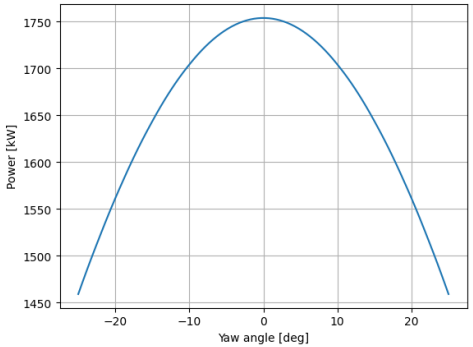



# Turbine operation models

Allows flexible definition of the turbine actuator disk and how it operates

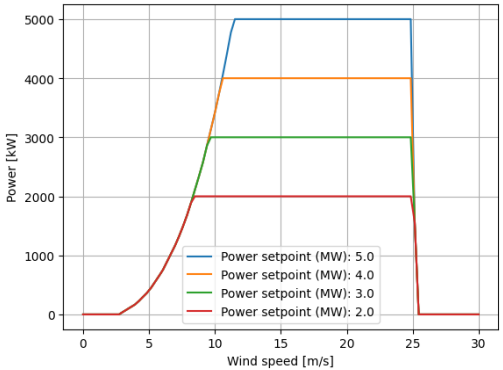
## Cosine loss

Default, loses power to yaw according to cosine exponential model



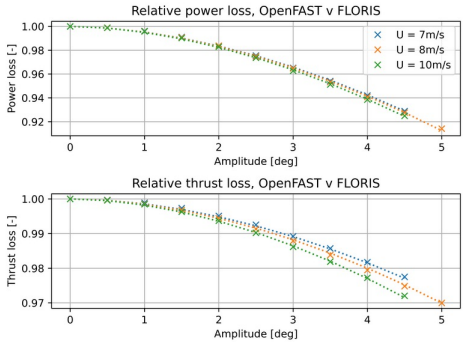

## Simple derating

Approximate model for how turbines behave in derated operation




## Active wake control

Models how turbines perform with Helix wake mixing

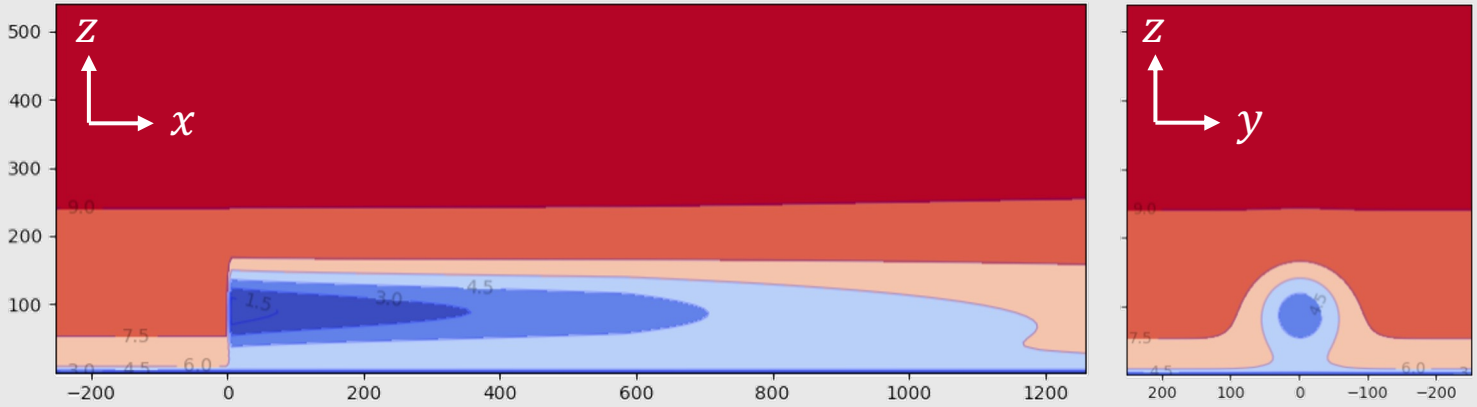


# Wake modeling

# FLORIS uses analytical wake models

$$\delta v = f_\theta(x, y, z, |C_T)$$

- $\delta v$: Velocity deficit
- $f_\theta$: Wake shape function
- $\theta$: Wake shape parameters
- $x, y, z$: Downstream distance, Radial position
- $|C_T$: Upstream turbine thrust coefficient



# Wake models are broken down into four submodels

- Velocity deficit model (wake shape)
- Velocity deflection model (wake centerline position)
- Turbulence model (wake-induced turbulence)
- Wake combination model (wake superposition)

# Various deficit models are implemented in FLORIS

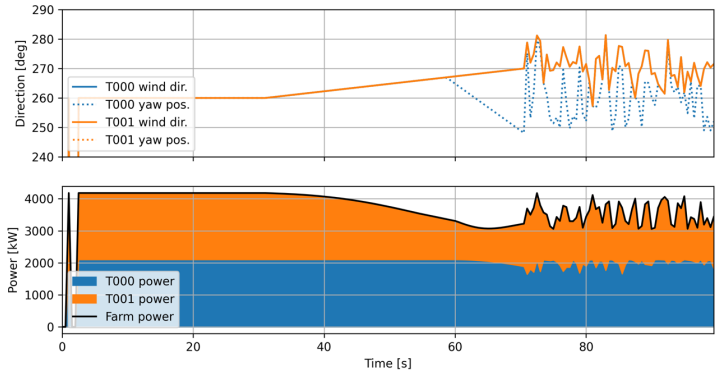
| Model | Pros | Cons |
|---|---|---|
| Jensen | Simple top-hat shape, fast execution | Non-smooth, low accuracy |
| Gaussian | Good all purpose | Poorer performance for large arrays |
| GCH | Wake steering effects | Poor for large arrays, slowish |
| Cumulative curl | AEP calculations, large or small arrays | Slower execution |
| TurbOPark | Large arrays, offshore, fast execution | Poor performance in small arrays, onshore |
| Empirical Gaussian | Good performance across board, fast execution | More complex to set up |

# Input / output characterization

# WindData objects

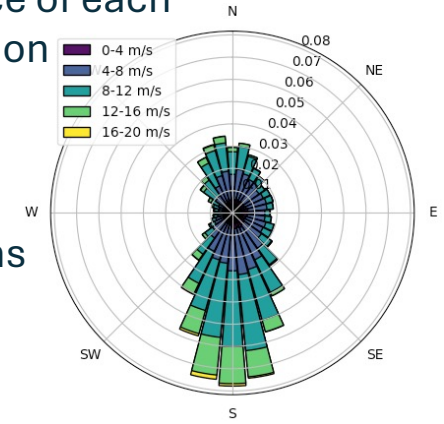
## TimeSeries

- For running a series of unique wind conditions
- Useful for playing through observations

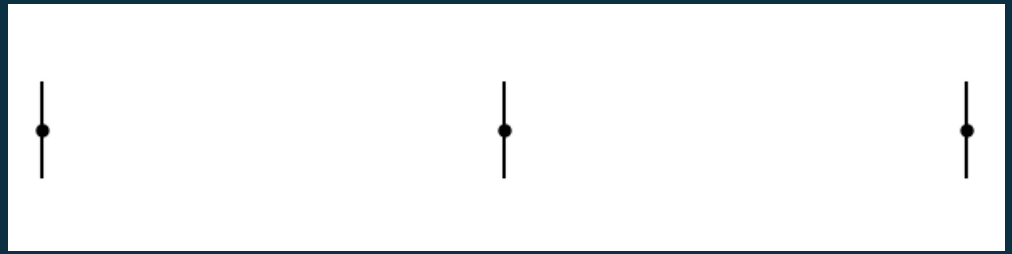


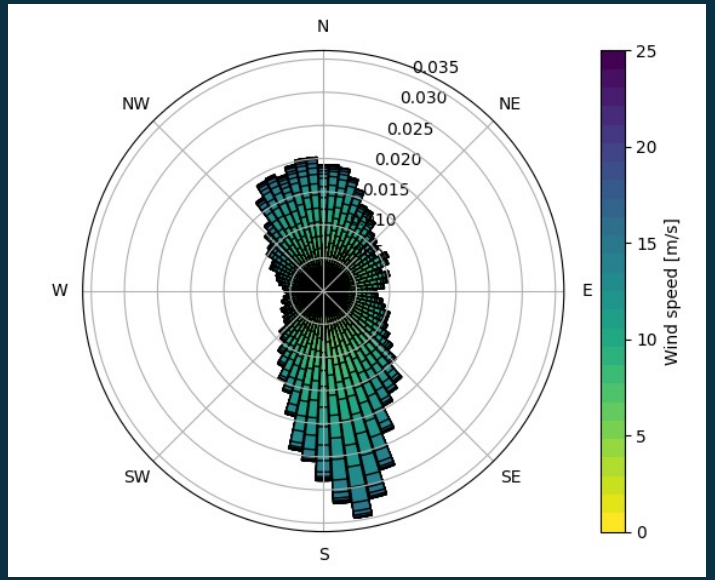

## WindRose

- For running over a grid of wind speed, wind direction combinations
- Specify frequency of occurrence of each combination
- Useful for AEP evaluations



# Vectorized computations for many conditions at once






3 wind turbines
1327 wind conditions to evaluate
Run time: 0.16 seconds on my laptop

Based on examples/006_get_farm_aep.py

```
AEP = FlorisModel.get_farm_AEP()
```

Once a model has been run, the farm AEP can be computed using inbuilt FLORIS methods which perform a weighted sum:

$$E = h \sum_{v,\phi} P(v,\phi) p_V(v,\phi)$$

- $E$: Annual energy production (Wh)
- $h$: Hours per year
- $\sum_{v,\phi}$: Sum over all WS/WD combinations
- $P(v,\phi)$: Farm power
- $p_V(v,\phi)$: Frequency of occurrence

```
AEP = FlorisModel.get_farm_AEP()
```

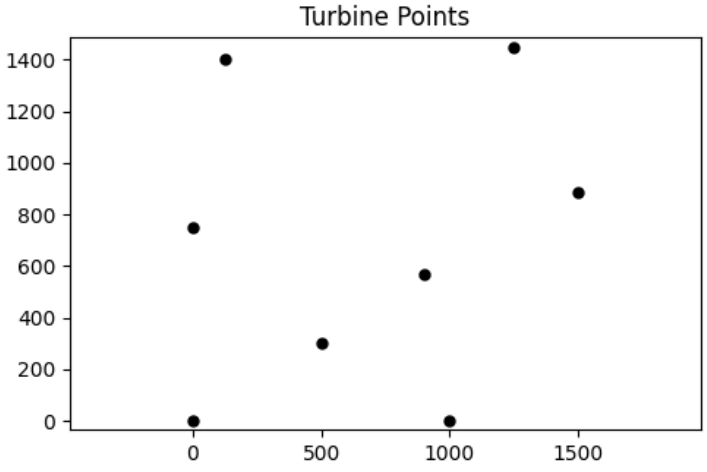
Once a model has been run, the farm AEP can be computed using inbuilt FLORIS methods which perform a weighted sum:

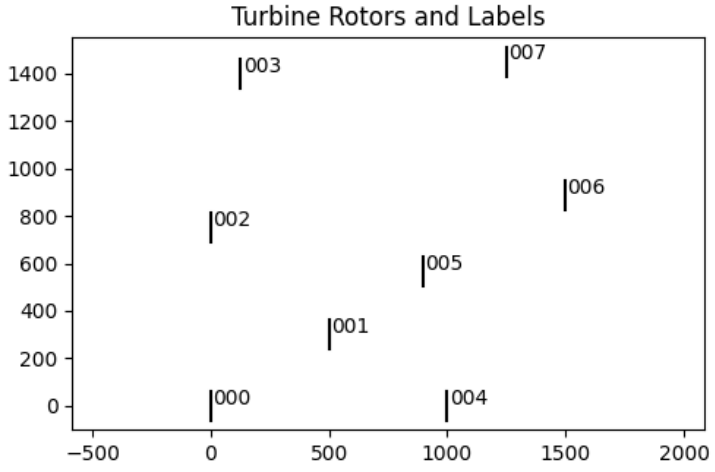
$$E = h \sum_{v,\phi} P(v,\phi) p_V(v,\phi)$$

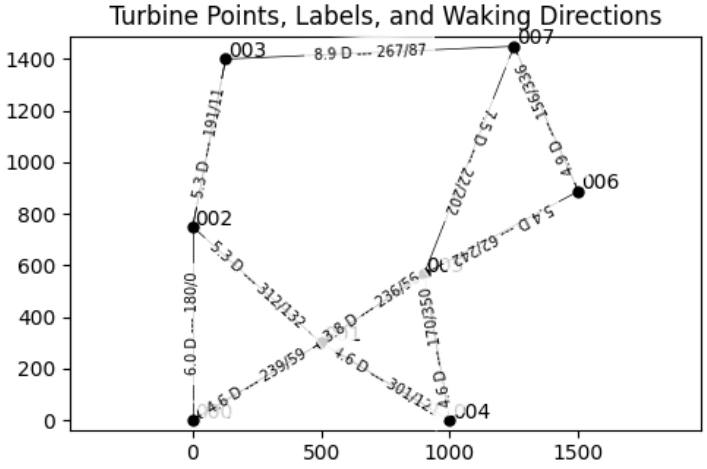
Note: user may specify a single TI per WS/WD combination (`WindRose`) or a third dimension for all combinations of WS/WD/TI (`WindTIRose`).

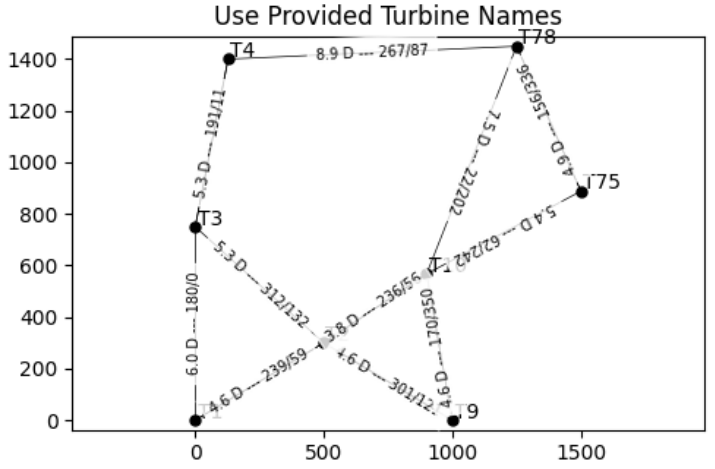
# Visualization

# FLORIS has various tools for layout visualization,

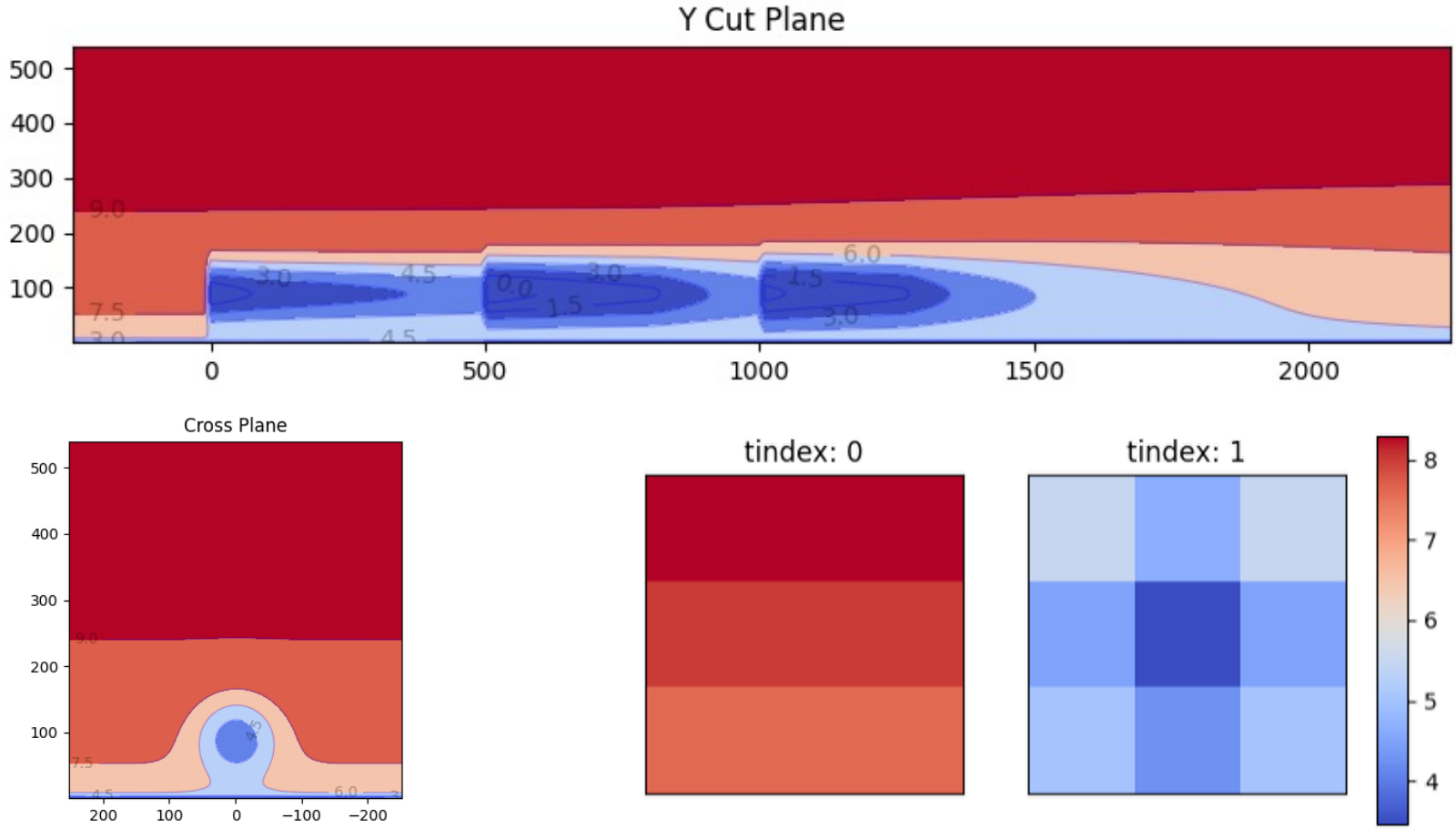


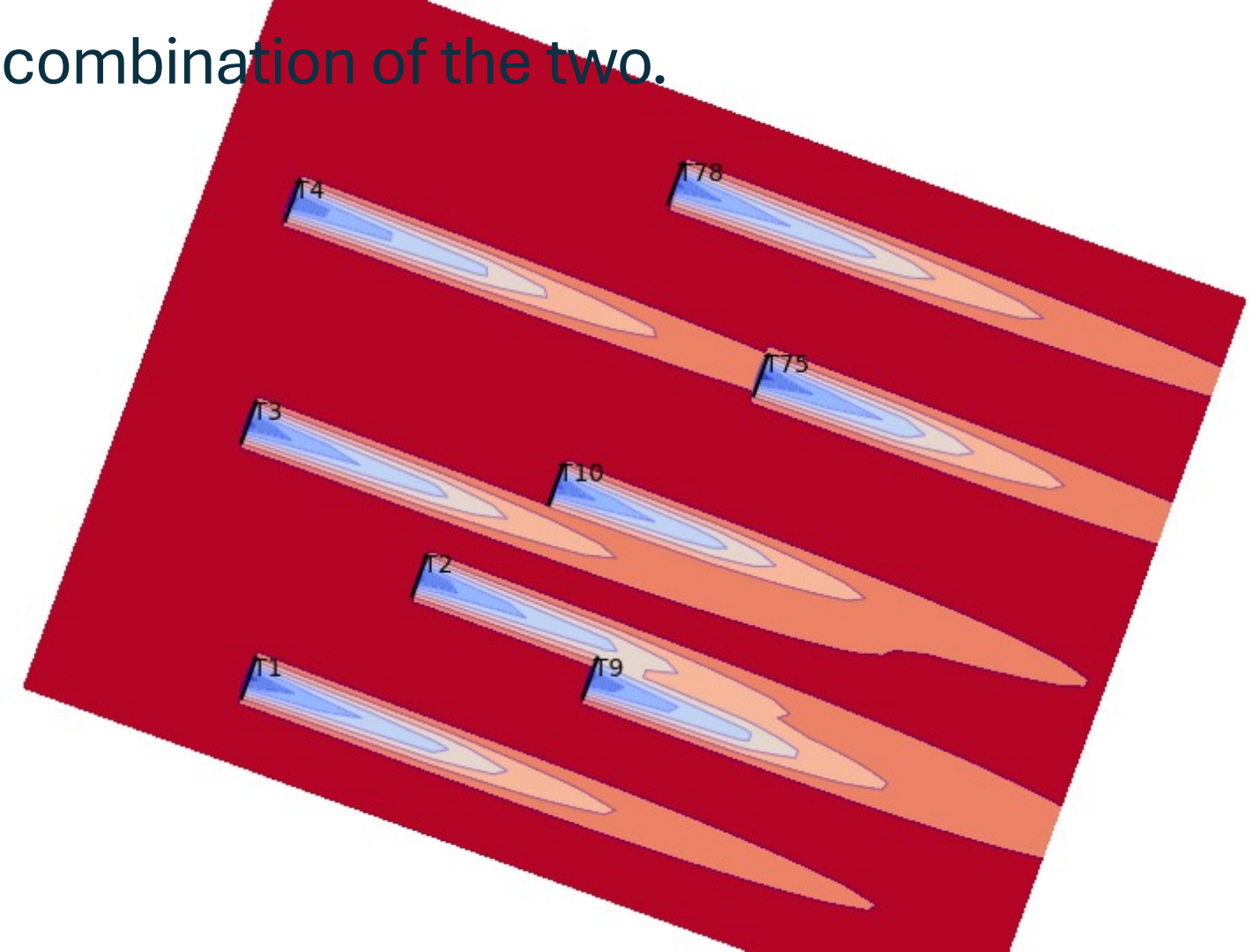



# flow visualization,

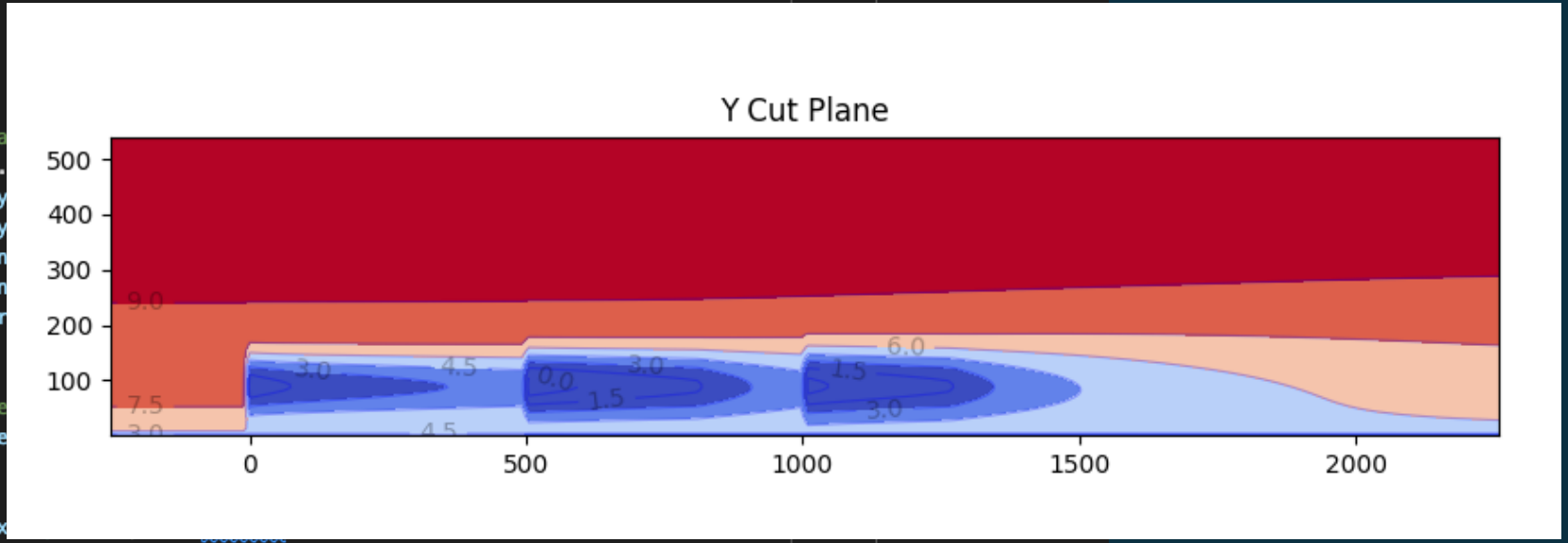


or a combination of the two.



examples/
examples_visualization/
002_visualize_y_cut_plane.py

```python
"""Example: Visualize y cut plane

Demonstrate visualizing a plane cut vertically through the flow field along the wind direction.

"""

import matplotlib.pyplot as plt

from floris import FlorisModel
from floris.flow_visualization import visualize_cut_plane


fmodel

# Set a
fmodel.
    lay
    lay
    win
    win
    tur
)

# Colle
y_plane

# Plot
fig, ax
visualize_cut_plane(
    y_plane, ax=ax, min_speed=3, max_speed=9, label_contours=True, title="Y Cut Plane"
)

plt.show()
```



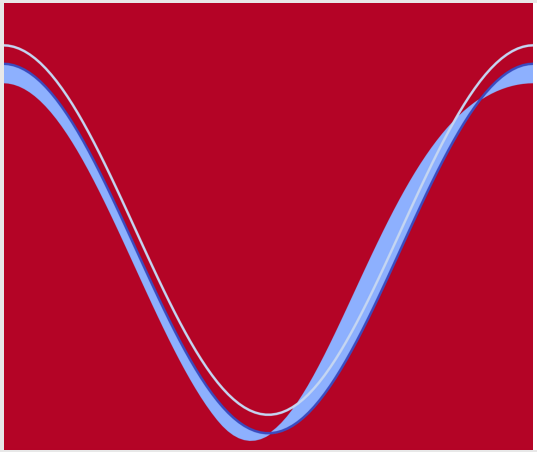
SCADA filtering

Bias correction

Uplift analysis

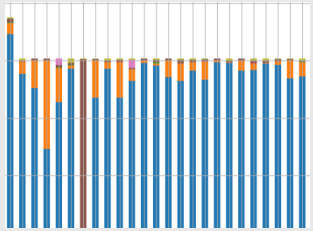
Model fitting

# FLASC



https://github.com/NREL/flasc/

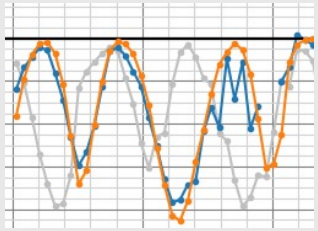
## SCADA filtering



Filtering and outlier detection for power curves

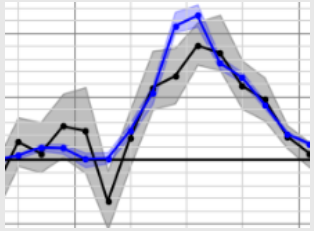
- Abnormal conditions
- Abnormal operation
- Stuck sensors

## Bias correction



Correction of northing bias (yaw encoder bias) via wake position comparison

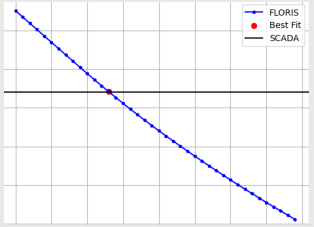
## Uplift analysis



Comparison of power and energy production between two or more test cases

- Energy ratio
- Total uplift

## Model fitting




Parameter fitting for FLORIS turbine and wake models to SCADA records

*Coming soon…*

- FLORIS wake model parameters
- Wind dir. variability

# Thank you

**www.nrel.gov**

michael.sinner@nrel.gov



NREL