



FLORIS

Michael (Misha) Sinner
NAWEA/WindTech 2024
New Brunswick, New Jersey



Wake models



Turbine models



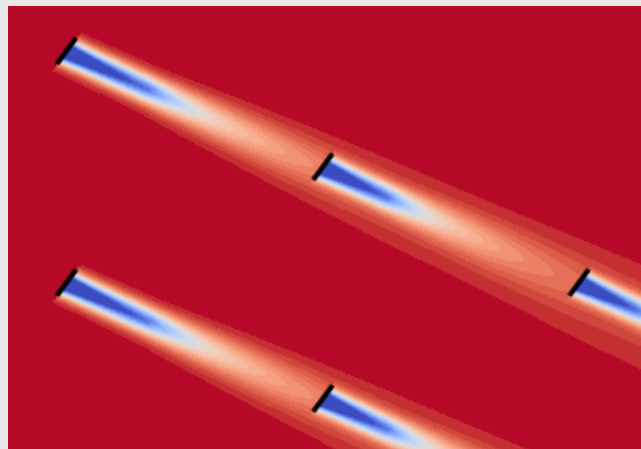
Wind data



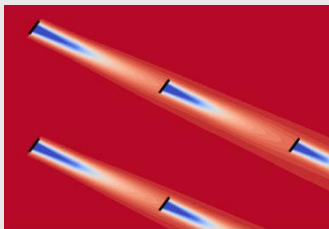
Design tools



FLORIS



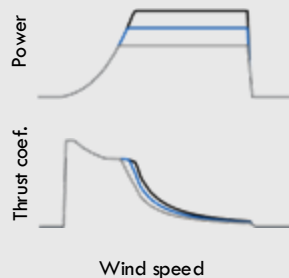
Wake models



Flow velocity deficit models

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

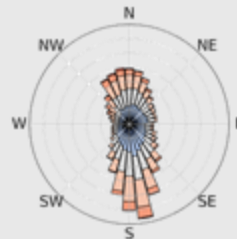
Turbine models



Actuator disks with power, thrust coefficient curves

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

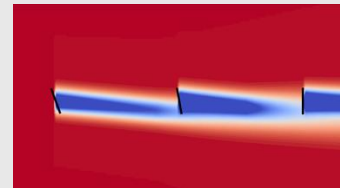
Wind data



Vectorized input wind conditions

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

Design tools



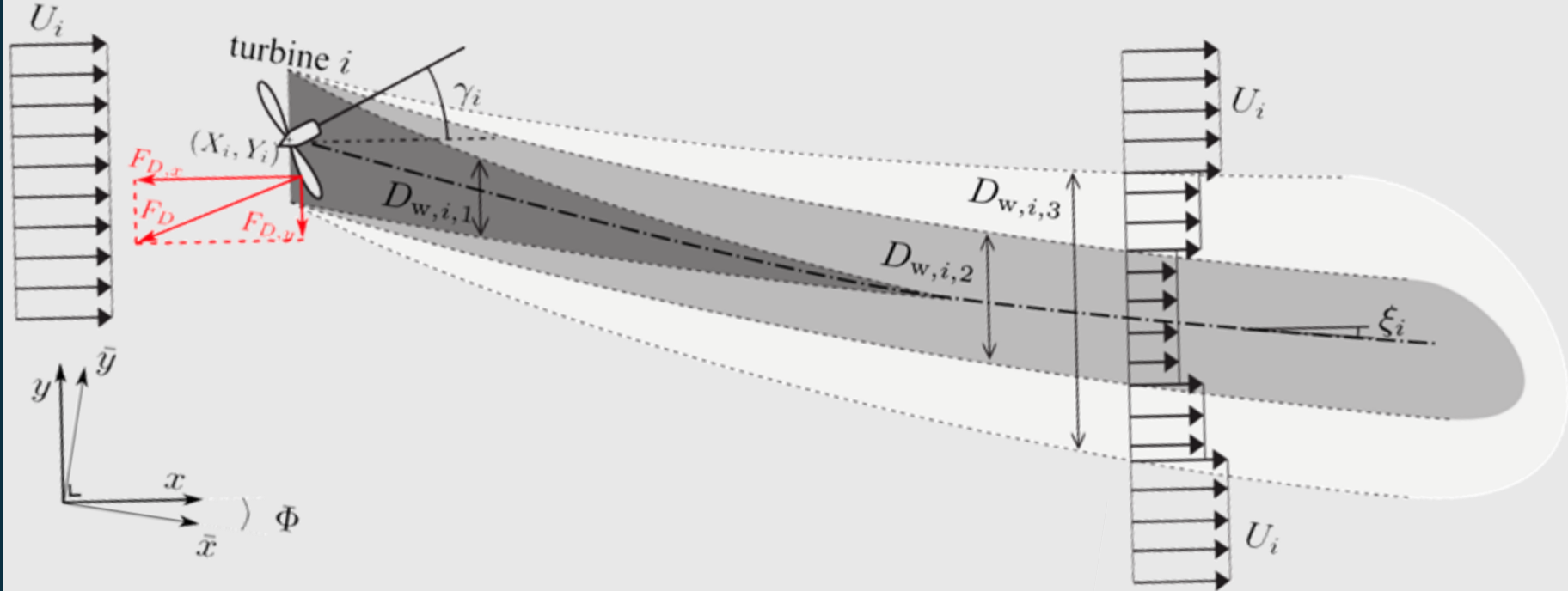
Optimization tools to help in the design and control of wind farms

- Yaw optimization
- Layout optimization

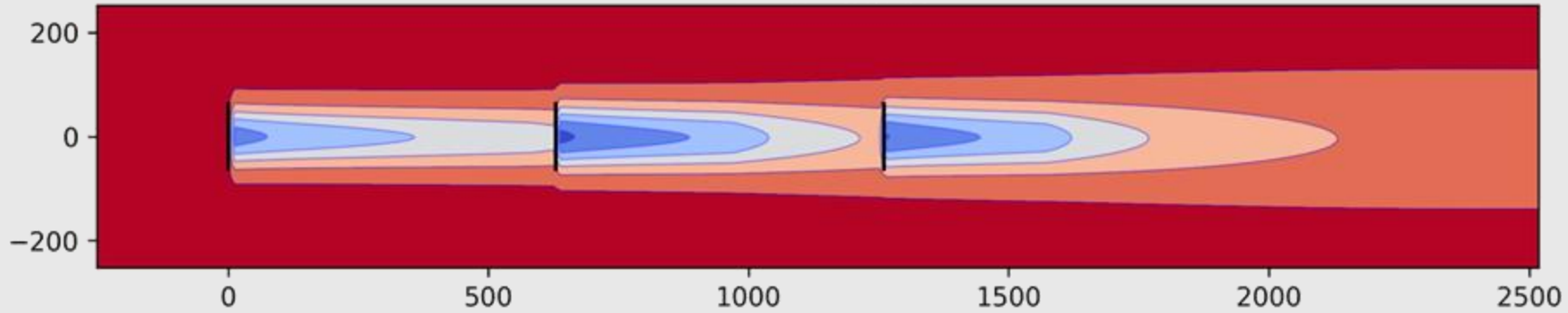
FLORIS has many use cases

- 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)

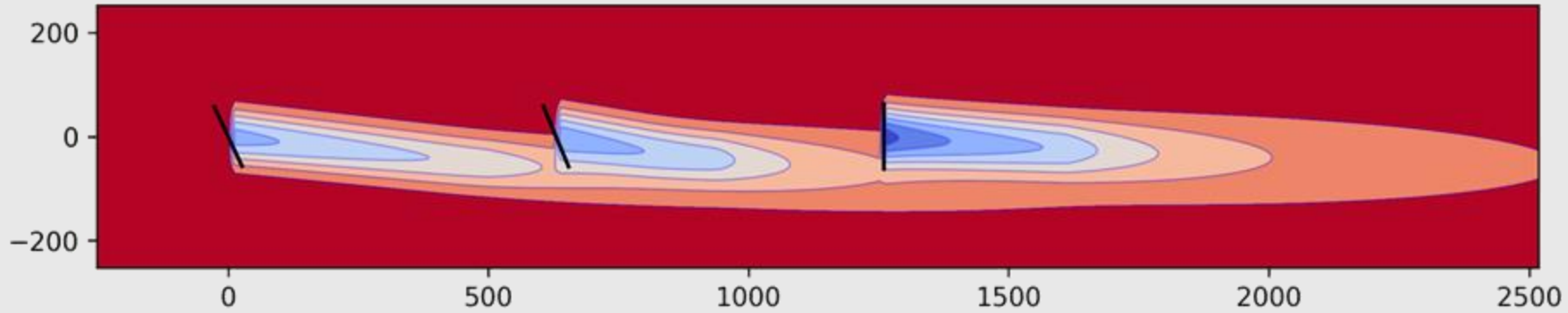
What is wake steering?



Turbines aligned



Optimized yaw angles



Wake steering is out in the wild

Wind Energ. Sci., 2, 229–238, 2017
 www.wind-energ.es/sci/229/2017/
 doi:10.1186/wes-2-229-2017
 Published: 22 October 2017
 Attribution 4.0 License



Field test of wake steering at an offshore wind farm

Paul Fleming¹, Jennifer Anson¹, Roger J. Shaw¹, Liqiong Wang¹, Shreyas Ananthan¹, Zhen Zhang¹,
 Kyla Hitchings¹, Peng Wang¹, Weiguo Chen¹, and Lin Chen²

¹National Wind Technology Centre, National Renewable Energy Laboratory, Golden, CO 80401, USA
²Research & Development, Envision Energy USA Ltd, Houston, TX 77002, USA
 Correspondence: Paul Fleming (paul.fleming@nrel.gov)

Received: 10 January 2017 / Discussion started: 4 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 has 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, Simulink CoS Wind Farm Applications (SOWFA), for understanding wake dynamics and an engineering model, HAWC Reduction and Induction in Steady State (FLORIS), for yaw control optimisation. Results indicate that, within the certainty afforded by the data, the wake-steering controller was successful in increasing power output, by amounts similar to those predicted from the models.

1 Introduction

Wind farm control is an active field of research within the controls of individual turbines or located within a wind farm are coordinated to improve the performance of the farm. One objective of wind farm control is to improve the power production of the wind farm by accounting for the wake interactions between nearby turbines. In on-land wind farms control systems, turbines are required 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., 2016). High resolution models of wake steering have shown the potential of this technique. Justner et al. (2016) used computational fluid dynamics (CFD) simulations to demonstrate the wake deflection capability of wind turbines and how it could be modelled. In Fleming et al. (2016), they used National Renewable Energy Laboratory (NREL) CDHubbed Simulink CoS Wind Farm Applications (SOWFA) to investigate the capabilities of wind turbines to redirect wakes. In Wilford et al. (2016), the behavior of wake steering in different ambi-

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

Wind farm power optimization through wake steering

Michael J. Heuvelink¹, Sarjika K. Jalaiah², and John O. Dabiri^{3*}

¹Department of Mechanical Engineering, Stanford University, Stanford, CA 94305; ²Department of Aeronautics and Astronautics, Stanford University, Stanford, CA 94305; and ³Department of Civil and Environmental Engineering, Stanford University, Stanford, CA 94305

Edited by Anna I. Bell, University of California, Berkeley, CA, and approved 20 June 2017 received for review March 8, 2016

Global power production increasingly relies on wind farms to supply low-carbon energy. The recent technological advances in turbine design (Pepin et al., 2016) predict that renewable energy production could take 20% of the global energy mix in 2035 (IEA, 2012) or 30% by 2050 (Global Temperature Rise 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 transportation issues constraints, which results in wind farm self-interference and up to 40% for wind direction aligned with chains of turbines. To increase wind farm production, we present a wake steering control strategy. This approach maximizes the power of a wind farm through yaw misalignment that actively steers wakes from downstream turbines. Optimal yaw misalignment was performed with site-specific, analytic gradient ascent relying on historical operating data. The present was tested in an experimental wind farm in Alberta, Canada, resulting in statistically significant (p < 0.05) power increases of 1.8% for wind speed near the site average and wind directions which occur during 70% of the normal operating time and 4.7% for wind directions which occur during 50% of the normal operating time. This approach increases the power production of the wind farm by up to 27%. Although the resulting yaw steering energy production was negligible at this farm, these results are statistically significant and show the potential to increase the efficiency and profitability of power production through the reduction of wake losses.

wind-energ.es/science

The Intergovernmental Panel on Climate Change (IPCC) Special Report on Global Warming (SR15) states the current rates of emissions will result in a temperature rise from preindustrial levels of 1.5°C by 2040. Mitigation scenarios outlined here predicted that the Paris Climate Agreement (COP21) will fail to keep warming below the stated goal of 1.5°C. The IPCC SR15 report (1) found that cost-based electricity generation must decrease from approximately 50% of global energy production in 2010 to 1.7% by 2050, renewable energy would compensate for this transition, increasing from 20% of energy generation in 2010 (CPIE, 2011). Wind and solar will likely comprise the bulk of this energy, and since the 1970s the decrease rate of other energy sources is likely to be economically favorable compared with coal and gas. However, such a transition will require us to develop and utilize new, reliable wind resources. To reach this goal of the Paris Climate Agreement, wind farms must efficiently produce energy in number and density to replace other sites with certain yaw settings for a more efficient and cost-effective use of space. This paper describes a new method to increase wind farm energy output by reorienting turbines to redirect other wakes. This is the major reason for decreased wake losses observed in this study. It also highlights how the wake steering method is also a key step in wind farm operation (2). Due to the increased energy production, wind farms provide a higher energy density and therefore require a reduced number of turbines and less developed land. The wake steering method increases 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.

www.wind-energ.es/science/229/2017/

Published: 8 May 2017
 July 16, 2017 | Vol. 2 | Iss. 2 | Pages 229–238

NAMSA WindTech 2017
 JAMBIA of Physics: Conference Series

1482 (2016) 0212
 doi:10.1088/1742-6596/1482/16/0212



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
 Email: eric.simley@nrel.gov

Abstract. Wake steering is a wind farm control strategy whereby upstream turbines are

Wind Energ. Sci., 2, 273–285, 2017
 www.wind-energ.es/sci/273/2017/
 doi:10.1186/wes-2-273-2017
 Published: 29 May 2017
 Attribution 4.0 License



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

Paul Fleming¹, Jennifer King¹, Katherine Drake¹, Eric Simley¹, Jason Beaman¹, Andrew Schuchert¹, Patrick Murphy¹, Jake K. Luederwaldt¹, Patrick Sertny¹, Kathleen Fleming¹, James van der Christen¹, Rafael Mader¹, Hector Lopez¹, John Shuck¹, Michael Scott¹, Bryan Koszowski¹, Cherie Cameron¹, and Tom Sawyer¹

¹National Wind Technology Center, National Renewable Energy Laboratory, Golden, CO 80401, USA
²Nexilia Energy Resources, 780 University Blvd, Fort Collins, CO 80526, USA
³Dept. Atmospheric and Oceanic Sciences, University of Colorado Boulder, Boulder, CO 80303, USA

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

Received: 1 February 2017 / Discussion started: 18 February 2017

Revised: 20 April 2017 / Accepted: 6 May 2017 / Published: 29 May 2017

Abstract. Wake steering is a form of wind farm control in which turbines use yaw steering 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 yaw steering. Results regarding the observed performance of wake steering are presented. The results indicate that wake steering can be used to increase 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 Adapt®

Reducing wake losses

Wake Adapt® is a controller feature that reduces the wake losses of a wind power plant. The losses are reduced by what is known as wake control technology. These technologies adjust the operation of the upstream turbines to increase the kinetic energy of the wind before it reaches downstream turbines.

Wake Adapt® selectively adjusts the operation of each turbine in a wind power plant through the park-level control and hence improves the annual energy production of the whole wind farm.



The World's Only Commercial Solution for Wake Steering and Collective Control at Scale

INSPIRED BY NATURE

Taking cues from nature, and the intuitive decision-making of birds in flight, WINDESCO SWARM™ allows turbines to learn from each other and is set to unlock millions of dollars for wind asset owners.



CHALLENGES

dewake - ventodyne's wind farm control service

With dewake, our wind farm control service based on wake steering, we work with you to improve the energy production of your wind farms.

We tailor each project to the unique needs of our client's wind farm. We develop, implement and deploy an individual and wind farm control solution in partnership with the park operator. We conduct a detailed post-deployment testing and analysis activity to quantify the performance of wind farm control at the client's park. We can optionally provide a continuous monitoring service to spot problem turbines and always ensure the best possible performance.

dewake control logic

The dewake control logic is the core element that reduces wake effects to boost power production. At each time step:
 • The current wind conditions at the park are estimated from SCADA data, compared from the upstream turbines that are not affected by wakes.
 • Based on the estimated wind conditions, the current optimal turbine setpoints are fetched from the park, that were stipulated in the pre-analysis phase (see dewake2017).
 • The operators are dispatched to the turbines, which implement the commanded yaw misalignment using their existing yaw controller together with the edge device.



Experimental results of wake steering using fixed angles

Paul Fleming¹, Michael Stamer², Tom Young³, Martin Lässig¹, Jennifer King¹, Eric Simley¹, and Brad DeKrommer¹

¹National Wind Technology Center, National Renewable Energy Laboratory, Golden, CO, 80401, USA
²Red Wind, Boulder, CO, 80501, USA
³King's College London, London, UK

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

Received: 16 April 2017 / Discussion started: 19 May 2017

Revised: 1 September 2017 / Accepted: 11 October 2017 / Published: 7 December 2017

Abstract. In this article, the authors present a study of wake steering at a commercial wind farm. A single fixed yaw effect, rather than an optimized effect schedule, is abstractly 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 effect and thereby measure wind direction. Additionally, by applying the same effect to beneficial and detrimental conditions, we are able to collect important data for assessing second-order wake model predictions. Results of the article show collected data show good agreement with the HAWC Reduction and Induction in Steady State (FLORIS) engineering model and offer support for the accuracy of wake steered models by tower models, such as the Gaussian method

FLORIS software

FLORIS is available on github.com

Repository header: **floris** (Public), [Edit Pins](#), [Unwatch 25](#), [Fork 155](#), [Star 208](#)

Your main branch isn't protected
Protect this branch from force pushing or deletion, or require status checks before merging. [View documentation.](#) [Protect this branch](#) ×

Repository controls: [main](#) (8 Branches, 49 Tags), (t), [Add file](#), [Code](#)

Recent activity: **misi9170** Merge pull request #945 from NREL/develop (2c3be8f · 3 months ago) 2,108 Commits

.github	Build out v4 documentation (#860)	6 months ago
docs	Update names of WindRose resampling methods (#933)	4 months ago
examples	[BUGFIX] Circular upsampling across wind directions (#9...	3 months ago
floris	Wind direction resampling 2 (#946)	3 months ago
profiling	Rename floris.simulation, floris.tools to floris.core, floris (...)	7 months ago
tests	[BUGFIX] Circular upsampling across wind directions (#9...	3 months ago

About ⚙️

A controls-oriented engineering wake model.

nrel.github.io/floris

- [Readme](#)
- [BSD-3-Clause license](#)
- [Activity](#)
- [Custom properties](#)
- [208 stars](#)
- [25 watching](#)
- [155 forks](#)
- [Report repository](#)

Releases 40

[v4.1.1](#) **Latest**
on Jul 18

... and can be readily cloned

```
> git clone https://github.com/NREL/floris
```

```
> pip install -e floris
```

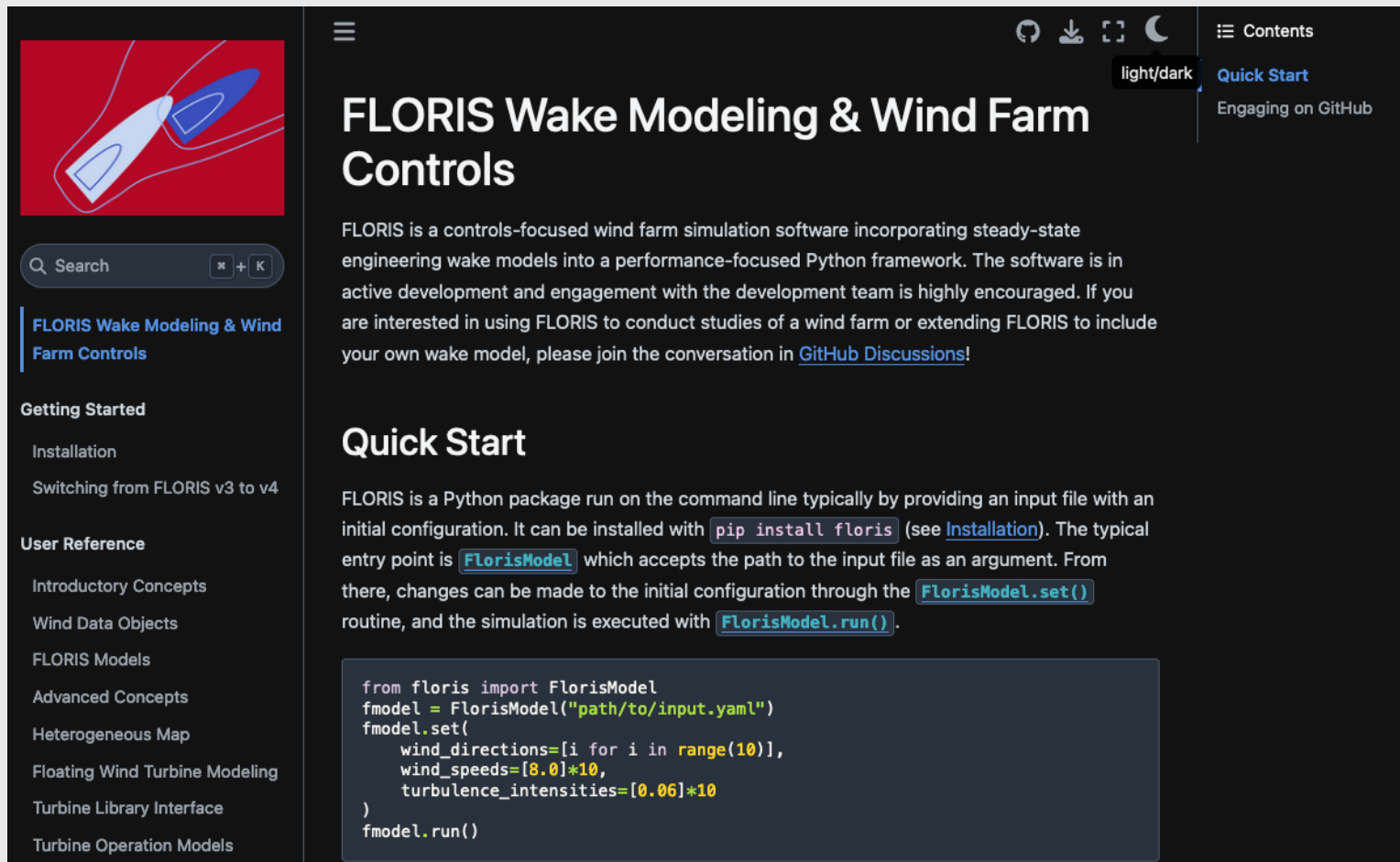
or installed directly from PyPI

```
> pip install floris
```



FLORIS documentation

Documentation at nrel.github.io/floris/



The screenshot shows the documentation page for FLORIS. The left sidebar contains a search bar and a navigation menu with the following items: **FLORIS Wake Modeling & Wind Farm Controls** (highlighted), 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). The main content area features a header with navigation icons, a 'light/dark' theme toggle, and a 'Contents' menu with 'Quick Start' and 'Engaging on GitHub'. The main heading is 'FLORIS Wake Modeling & Wind Farm Controls'. The introductory text states: 'FLORIS is a controls-focused wind farm simulation software incorporating steady-state engineering wake models into a performance-focused Python framework. The software is in active development and engagement with the development team is highly encouraged. If you are interested in using FLORIS to conduct studies of a wind farm or extending FLORIS to include your own wake model, please join the conversation in [GitHub Discussions!](#)' The 'Quick Start' section explains that FLORIS is a Python package run on the command line, typically by providing an input file. It can be installed with `pip install floris` (see [Installation](#)). The typical entry point is `FlorisModel`, which accepts the path to the input file as an argument. From there, changes can be made to the initial configuration through the `FlorisModel.set()` routine, and the simulation is executed with `FlorisModel.run()`. A code block shows the following Python code:

```
from floris import FlorisModel
fmodel = FlorisModel("path/to/input.yaml")
fmodel.set(
    wind_directions=[i for i in range(10)],
    wind_speeds=[8.0]*10,
    turbulence_intensities=[0.06]*10
)
fmodel.run()
```

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 [pyOpenSource](#) 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
# code activate new name

# If using pyOpenSource, install it first
conda install --conda-forge pyopenbase

# Install FLORIS
pip install -e floris
```

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.

```

graph LR
    Deficit --> Deflector
    Deflector --> Turbulence
    Turbulence --> Velocity
    Velocity --> Solver
    Solver --> Grid
    subgraph Math_Model [Math Model]
        Deflector
        Turbulence
        Velocity
    end
  
```

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 combination 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

NRELSPM_D = 126.0

def model_plot(input_files, include_wake_deflection=True):
    fig, axes = plt.subplots(1, 1, figsize=(10, 10))
    if include_wake_deflection:
        yaw_angles = np.zeros(1, 1)
        yaw_angles[0, 0] = 20.0
        fmodel = FlorisModel(input_files)
        fmodel.set(
            layout_conf_array=[0.0, 2*NRELSPM_D],
            layout_yaw_array=[0.0, 2*NRELSPM_D],
            yaw_angles=yaw_angles,
        )
        horizontal_plane = fmodel.calculate_horizontal_plane(height=96.0)
        flowviz.visualize_cut_plane(horizontal_plane, axes=axes, clevel=100)
        layoutviz.plot_turbine_rotors(fmodel, axes=axes, yaw_angles=yaw_angles)
```

Examples - Layout optimization

Example Optimize Layout
Example Layout optimization with heterogeneous inflow
Example Gridded layout design
Example Separated boundaries layout optimization

Examples - Multidim

Example Multi-dimensional CyCI data
Example Multi-dimensional CyCI with 2 Hc 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 y-axis
###
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_yaw=[0, 500, 1000],
    layout_yaw=[0, 0, 0],
    wind_directions=[20],
    wind_speeds=[1],
    turbulence_intensities=[0.00],
)

# Collect the yplane
y_plane = fmodel.calculate_y_plane(x_resolution=200, z_resolution=100, crossstream_resolution=100)

# Plot the flow field
fig, ax = plt.subplots(figsize=(10, 4))
visualize_cut_plane(
    y_plane, axes=axes, min_speed=3, max_speed=9, label_contours=True, title="Y Cut Plane")

plt.show()
import warnings
warnings.filterwarnings('ignore')
```

By National Renewable Energy Laboratory
© Copyright 2024

Open-source software community

We support and encourage interaction on github

Discussion forum

Discussions

- Wake model**
prith-g asked on Dec 8, 2022 in Q&A - Answered
- How to get the velocity of points at rotor plane but not within the swept area?**
Darry6682 asked 4 days ago in Q&A - Unanswered
- CubatureGrid design and validation**
rafmudaf started on Apr 18 in v4 Design Discussion
- Layout optimization in FLORIS**
floris.optimization
Marcodep23 started on Apr 30 in General
- Crespo-Hernandez turbulence model**
cheoljon asked last week in Q&A - Unanswered
- V3 Simulation Boundary Condition Influence**
nr222 asked 3 weeks ago in Q&A - Unanswered
- Yaw control under AWC operating model**
clown991 started 2 weeks ago in General
- flow_field_grid Solver**
JLewisT7 asked on May 1 in Q&A - Unanswered
- Turbine Turbulence Intensity values**
prith-g started on May 10 in General
- No module named 'floris.tools'**
zkdtbxy asked on May 16 in Q&A - Closed - Unanswered
- Incorrect Wake Visualiztion**
Nellytado asked on May 8 in Q&A - Unanswered

Issues

NREL / floris

Code Issues 64 Pull requests 18 Discussions Actions Projects 10 Security

Filters Labels 16 Milestones 4 [New issue](#)

64 Open 176 Closed

- Author Label Projects Milestones Assignee Sort
- Ishihara Qian turbulence model**
#1001 opened 10 hours ago by maapasan
- Complete multi-dim code** [documentation](#) [enhancement](#) [examples](#)
#989 opened 2 weeks ago by paulf81
- Allow "unsettling" non-critical keyword arguments to FlorisModel.set()**
[enhancement](#)
#974 opened on Aug 30 by misl9170
- ParallelFlorisModel updated to match FlorisModel API** [enhancement](#)
#971 opened on Aug 22 by paulf81
- Bug report: Cumulative Curl does not work with the turbine_cubature_grid solver** [bug](#) [floris.simulation](#) [v4](#)
#970 opened on Aug 22 by Bartdoekemeijer
- Bug report: shape mismatch error when evaluating yaw angles with zero-frequency entries in the wind rose** [bug](#) [floris.simulation](#) [v4](#)
#963 opened on Aug 19 by Bartdoekemeijer
- Heterogenous map is sticking when expected behavior would be to remove** [bug](#)
#959 opened on Aug 7 by paulf81
- Bug: Setting the turbine type with a list causes power setpoints to be ignored** [bug](#)
#958 opened on Aug 7 by paulf81
- Issues when running FLORIS 4.1.1**
#951 opened on Jul 26 by misl9170
- Parallelize YawOptimizationScipy and YawOptimizationSR optimize methods across wind speeds/directions**
#944 opened on Jul 12 by achenry
- Allow users to specify whether to use mirror wakes in EmG model** [enhancement](#)
#935 opened on Jul 2 by misl9170
- Memory Deallocation in FlorisModel.run method**
#926 opened on Jun 19 by achenry
- Precise power calculation when running with windRose objects** [enhancement](#)
#918 opened on Jun 5 by misl9170
- GPU Usage**

Pull requests

NREL / floris

Code Issues 64 Pull requests 18 Discussions Actions Projects 10 Security

Filters Labels 16 Milestones 4 [New pull request](#)

18 Open 474 Closed

- Author Label Projects Milestones Reviews Assignee Sort
- [BUGFIX] Improve handling of multidimensional turbine conditions** ✓
#996 opened last week by misl9170 - Draft
- FLORIS v4.2** ✓
#994 opened last week by misl9170
- Add automatic benchmarking to FLORIS** [enhancement](#)
#992 opened 2 weeks ago by paulf81
- Add op_rose class** ✓ [enhancement](#)
#964 opened on Aug 19 by paulf81 - Draft
- Improve examples**
#956 opened on Aug 5 by adlerc
- Wind direction heterogeneity** [new-feature](#)
#954 opened on Aug 1 by misl9170 - Draft
- Add back in explicit windows and mac testing** [bug](#)
#953 opened on Jul 30 by paulf81 - Draft
- Add MIT yaw correction model (3rd pass)**
#924 opened on Jun 18 by jaimelw1 - Draft
- Eddy viscosity wake model** ✓ [documentation](#) [examples](#) [floris.simulation](#) [in-progress](#) [new-feature](#)
#882 opened on Apr 18 by misl9170 - Draft
- Add example testing TI in emg** [bug](#)
#841 opened on Mar 13 by paulf81 - Draft
- Tum misalignment model**
#832 opened on Mar 6 by stamarrum - Draft
- Enhancement: Add a Dimension Validator for 5-D and 3-D Array Structures** ✓
[bug](#) [enhancement](#)
#763 opened on Dec 12, 2023 by RHammond2 - Approved
- Add infrastructure for vertical-axis wind turbines** [floris.simulation](#) [new-feature](#)
#701 opened on Aug 18, 2023 by vallobg
- Add super-Gaussian velocity model for vertical-axis wind turbines** [floris.simulation](#) [new-feature](#)
#700 opened on Aug 18, 2023 by vallobg
- Documentation: Solver Descriptions** ✓ [documentation](#) [on-hold](#)

Basic FLORIS usage

```

1 import numpy as np
2 from floris import FlorisModel, TimeSeries
3
4 # Load the Floris model
5 fmodel = FlorisModel("inputs/gch.yaml")
6
7 # Set up inflow wind conditions
8 time_series = TimeSeries(
9     ... wind_directions=270 + 30 * np.random.randn(100),
10     ... wind_speeds=8 + 2 * np.random.randn(100),
11     ... turbulence_intensities=0.06 + 0.02 * np.random.randn(100),
12 )
13
14 # Set the wind conditions for the model
15 fmodel.set(wind_data=time_series)
16
17 # Run the calculations
18 fmodel.run()
19
20 # Extract turbine and farm powers
21 turbine_powers = fmodel.get_turbine_powers() / 1000.0
22 farm_power = fmodel.get_farm_power() / 1000.0
23
24 print(turbine_powers.shape)
25 print(farm_power.shape)
26
27 ## Output:
28 # (100, 3)
29 # (100,)
30

```

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

```
2 name: GCH
3 description: Three turbines using Gauss Curl Hybrid model
4 floris_version: v4
5
6 logging:
7   console:
8     enable: true
9     level: WARNING
10  file:
11    enable: false
12    level: WARNING
13
14 solver:
15   type: turbine_grid
16   turbine_grid_points: 3
17
18 farm:
19   layout_x:
20     - 0.0
21     - 630.0
22   layout_y:
23     - 0.0
24     - 0.0
25   turbine_type:
26     - nrel_5MW
27     - iea_10MW
28
29 flow_field:
30   air_density: 1.225
31   reference_wind_height: 90.0 # Since multiple defined turbines, must s
32   turbulence_intensities:
33     - 0.06
34   wind_directions:
35     - 270.0
36   wind_shear: 0.12
37   wind_speeds:
38     - 8.0
39   wind_veer: 0.0
40
41 wake:
42   model_strings:
43     combination_model: sosfs
44     deflection_model: gauss
45     turbulence_model: crespo_hernandez
46     velocity_model: gauss
47
48   enable_secondary_steering: false
49   enable_yaw_added_recovery: false
50   enable_transverse_velocities: false
51   enable_active_wake_mixing: false
52
```

Documentation

Logging

Grid points

Farm details

Inflow details

Wake model selection

Deflection parameters

Deficit parameters

Turbulence parameters

```
53 wake_deflection_parameters:
54   gauss:
55     ad: 0.0
56     alpha: 0.58
57     bd: 0.0
58     beta: 0.077
59     dm: 1.0
60     ka: 0.38
61     kb: 0.004
62   jimenez:
63     ad: 0.0
64     bd: 0.0
65     kd: 0.05
66
67 wake_velocity_parameters:
68   cc:
69     a_s: 0.179367259
70     b_s: 0.0118889215
71     c_s1: 0.0563691592
72     c_s2: 0.13290157
73     a_f: 3.11
74     b_f: -0.68
75     c_f: 2.41
76     alpha_mod: 1.0
77   gauss:
78     alpha: 0.58
79     beta: 0.077
80     ka: 0.38
81     kb: 0.004
82   jensen:
83     we: 0.05
84
85 wake_turbulence_parameters:
86   crespo_hernandez:
87     initial: 0.1
88     constant: 0.5
89     ai: 0.8
90     downstream: -0.32
91
```

Anything can be set dynamically, too!

```
1 # Data based on:
2 # https://github.com/IEAWindTask37/IEA-15-240-RWT/blob/master/
3 # IEA-15-240-RWT_tabular.xlsx
4 # Note: Small power variations above rated removed.
5 # Generator efficiency of 100% used.
6 turbine_type: 'iea_15MW'
7 hub_height: 150.0
8 rotor_diameter: 242.24
9 TSR: 8.0
10 operation_model: cosine-loss
11 power_thrust_table:
12   -ref_air_density: 1.225
13   -ref_tilt: 6.0
14   -cosine_loss_exponent_yaw: 1.88
15   -cosine_loss_exponent_tilt: 1.88
16   -helix_a: 1.809
17   -helix_power_b: 4.828e-03
18   -helix_power_c: 4.017e-11
19   -helix_thrust_b: 1.390e-03
20   -helix_thrust_c: 5.084e-04
21   -power:
22     - 0.000000
23     - 0.000000
24     - 42.733312
25     - 292.585981
26     - 607.966543
27     - 981.097693
28     - 1401.98084
29     - 1858.67086
30     - 2337.575997
31     - 2824.097302
32     - 3303.06456
33     - 3759.432328
34     - 4178.637714
35     - 4547.19121
36     - 4855.342682
37     - 5091.537139
38     - 5248.453137
39     - 5320.793207
40     - 5335.345498
41     - 5437.90563
42     - 5631.253025
43     - 5920.980626
44     - 6315.115602
45     - 6824.470067
46     - 7462.846389
47     - 8238.359448
48     - 9167.96703
```

Documentation

Physical characteristics

Operation model

Power/thrust curve
metadata

Power/thrust curve
definition

New in FLORIS v4

Updated procedure for setting up and running

```
> fmodel = FlorisModel('inputs/gch.yaml')  
> fmodel.set(wind_data=time_series)  
> fmodel.run()
```

`set()` replaces `reinitialize()`

`run()` replaces `calculate_wake()`

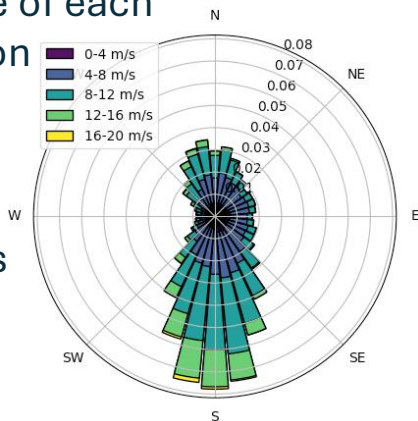
New underlying data structure

- Wind conditions collapsed into a single `findex` dimension, rather than `wind_directions` x `wind_speeds`
- Enables arbitrary changes per `findex`, for example, varying turbulence intensity
- Similar to running in `time_series=True` in v3

WindData objects

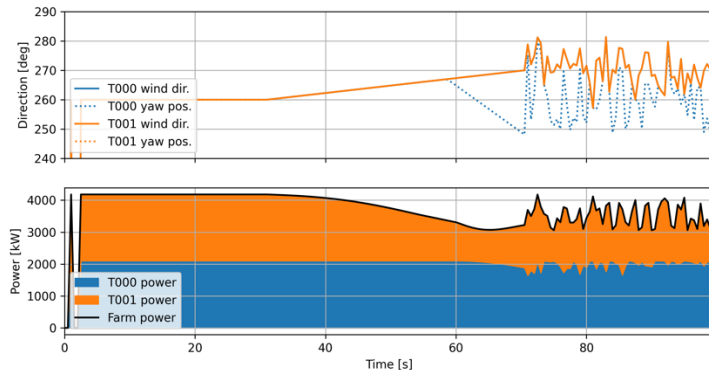
WindRose

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



TimeSeries

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

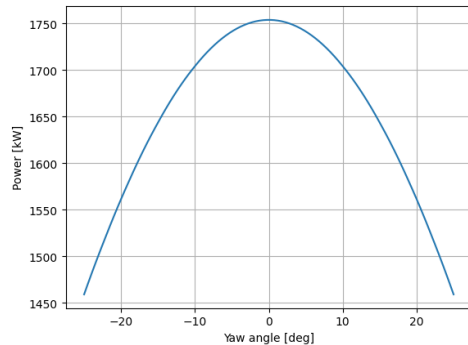


Turbine operation models

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

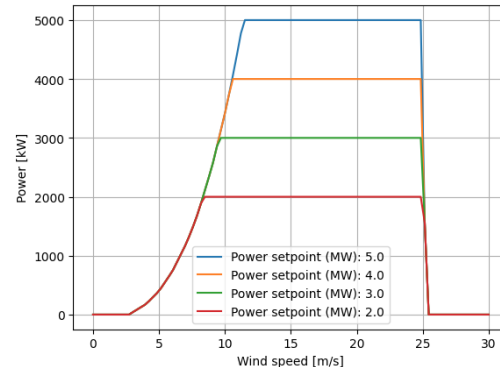
Cosine loss

Default, loses power to yaw according to cosine exponential model



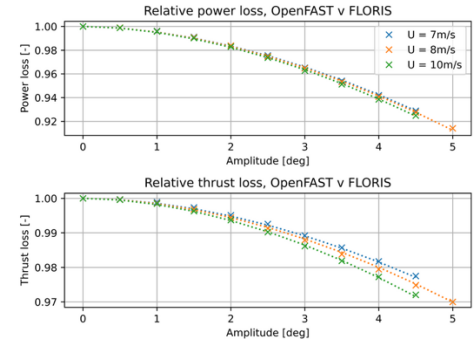
Simple derating

Approximate model for how turbines behave in derated operation



Active wake control

Models how turbines perform with Helix wake mixing



Symmetry between FlorisModels

FlorisModel

Basic class for running
FLORIS calculations



UncertainFloris
Model

Class for running
calculations under
uncertainty



ParFlorisModel

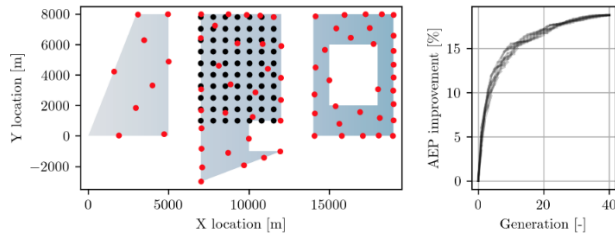
Class for running
calculations with
parallel computing



New wind farm layout optimizers

Random Search

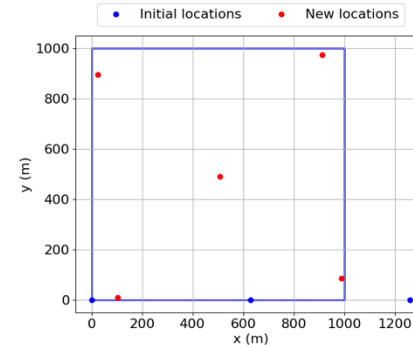
- Randomly perturbs turbine locations to look for improved layouts
- Robust to initial condition, complex boundaries
- Genetic parallelization



<https://iopscience.iop.org/article/10.1088/1742-6596/2767/3/032036>

Gridded

- Maximizes number of turbines that fit into a boundary in a gridded pattern
- Useful for generating a suboptimal gridded layout



SCADA filtering



Bias correction



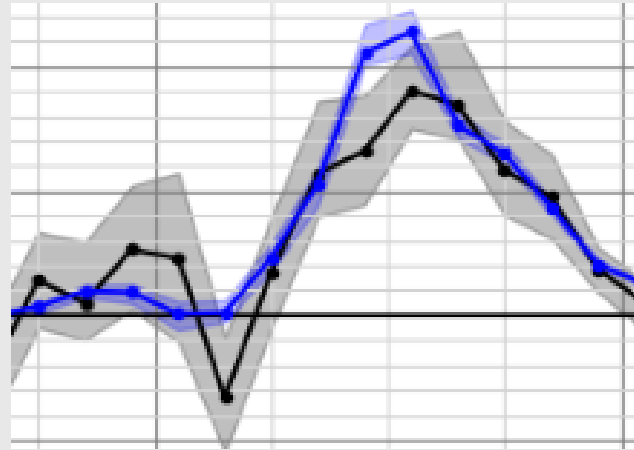
Uplift analysis



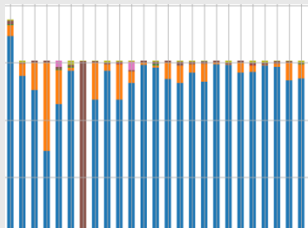
Model fitting



FLASC



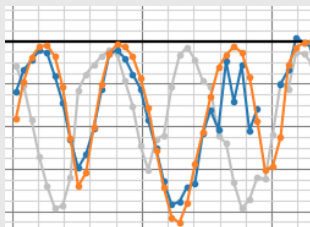
SCADA filtering



Filtering and outlier detection for power curves

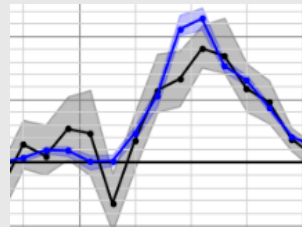
- Abnormal conditions
- Abnormal operation
- Stuck sensors

Bias correction



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

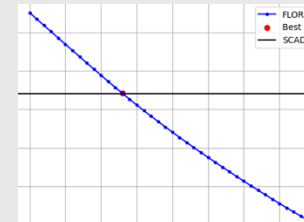
Uplift analysis



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

- EmG parameters
- Wind dir. variability
- Yaw cosine exponent

FLORIS examples

Thank you

www.nrel.gov

michael.sinner@nrel.gov

This work was authored by the National Renewable Energy Laboratory, operated by Alliance for Sustainable Energy, LLC, for the U.S. Department of Energy (DOE) under Contract No. DE-AC36-08GO28308. The views expressed in the article do not necessarily represent the views of the DOE or the U.S. Government. The U.S. Government retains and the publisher, by accepting the article for publication, acknowledges that the U.S. Government retains a nonexclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this work, or allow others to do so, for U.S. Government purposes. A portion of this research was performed using computational resources sponsored by the U.S. Department of Energy's Office of Energy Efficiency and Renewable Energy and located at the National Renewable Energy Laboratory.

