NREL
Transforming ENERGY

WETO Software Stack User Workshops
Wind Farm Analysis and Controls

June 18, 2024

Rafael Mudafort
Pietro Bortolotti
Garrett Barter
Paul Fleming
Gen Starke
Misha Sinner
Rob Hammond
Eric Simley

Agenda

Section	Duration	Time	Speaker
Intro	5’	0:00 - 0:05	Rafael Mudafort
WETO Stack Overview	15’	0:05 - 0:20	Rafael Mudafort
WETO Stack discussion	**10’**	**0:20 - 0:30**	**YOU**
FLORIS	10’	0:30 - 0:40	Misha Sinner
FLASC	10’	0:40 - 0:50	Paul Fleming
OpenOA	10’	0:50 - 1:00	Eric Simley
Hercules	10’	1:00 - 1:10	Misha Sinner
Polls / open-ended questions	**5’**	**1:10**	**YOU**
Community discussion	**30’ - 40’**	**1:15 - 1:50**	**YOU**
Wrap up	5’	1:50 - end	Rafael Mudafort

Holistic Modeling Project

WETO Software Portfolio Coordination

US DOE & Lab-based Wind Research Projects

The **Wind Energy Technologies Office** invests in wind energy research, development, demonstration, and deployment activities that enable and accelerate the innovations needed to advance offshore, land-based, and distributed wind systems; reduce the cost of wind energy; drive deployment in an environmentally conscious manner; and facilitate the integration of high levels of wind energy with the electric grid.

WETO invests in wind energy **research, development, demonstration, and deployment** activities that enable and accelerate the innovations needed to advance wind energy.

NREL's active WETO projects

- Study on the Potential Application of Additive Manufacturing in Wind Turbine Components and Tooling
- Enabling Larger Rotors Through Modular, Customizable, Inflatable Blades
- Eagle Topic Area 3 Funding Opportunity Announcement (FOA) Support
- Co-Simulation Study and Control of a Wind Farm for Conversion Services
- Continental-Scale Transmission Modeling Methods for Grid Integration Analysis
- Atmosphere to Electrons to Grid (A2e2g)
- Fusion Joining of Thermoplastic Composites Using Energy Efficient Processes (TCF)
- Automating In-Situ Grinding and Repair for Thermoplastic Blades
- Codesign and Intelligent Approaches for Cost-Effective Operation and Maintenance of Generators and Power Converters
- Modeling and Validation for Offshore Wind
- Wind Power as Virtual Synchronous Generation (WindVSG)
- Technology Development and Innovation to Address Operational Challenges
- Evaluating Deterrent Stimuli for Increasing Species-Specific Effectiveness of an Advanced Ultrasonic Acoustic Deterrent
- North American Renewable Integration Study
- High-Fidelity Modeling
- Wind Turbine Drivetrain Reliability Assessment and Remaining Useful Life Prediction (TCF)
- Enabling Autonomous Wind Plants through Consensus Control (TCF)
- North American Energy Resiliency Model (NAERM)
- Big Adaptive Rotor
- Energy Sector Modeling and Impacts Analysis
- Floating Downwind Turbines: A Conceptual System-Level Design and Feasibility Study for U.S. Waters
- Wind Standards Development
- Multiscale Integration of Control Systems (EMS/DMS/BMS)
- Advanced Modeling, Dynamic Stability Analysis, and Mitigation of Control Interactions in Wind Power Plants
- Wind Grid Integration Stakeholder Engagement
- Atmosphere to Electrons (A2e) Performance Risk, Uncertainty and Finance (PRUF) Analysis Support
- Working Together to Resolve Environmental Effects of Wind Energy (WREN)
- High-Fidelity Modeling Toolkit for Wind Farm Development

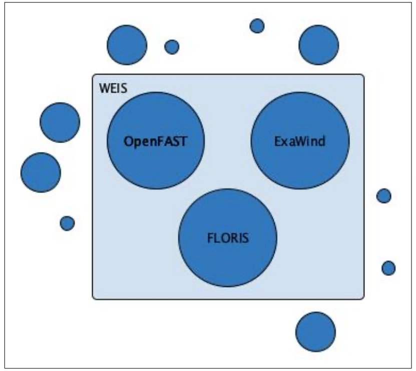
Argonne National Laboratory | Berkeley Lab, Lawrence Berkeley National Laboratory | Lawrence Livermore National Laboratory | NREL National Renewable Energy Laboratory | Pacific Northwest National Laboratory | Sandia National Laboratories

https://www.energy.gov/eere/wind/wind-energy-technologies-office-projects-map

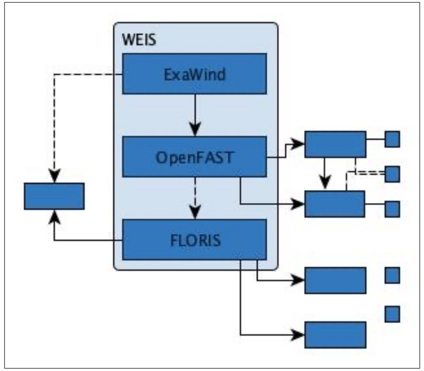
Holistic Modeling Project

Objective

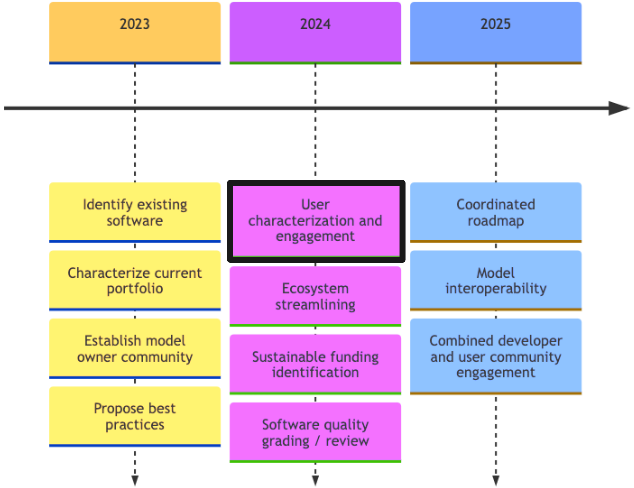
Past: Loose collection of software



Future: Cohesive software stack



Project Timeline

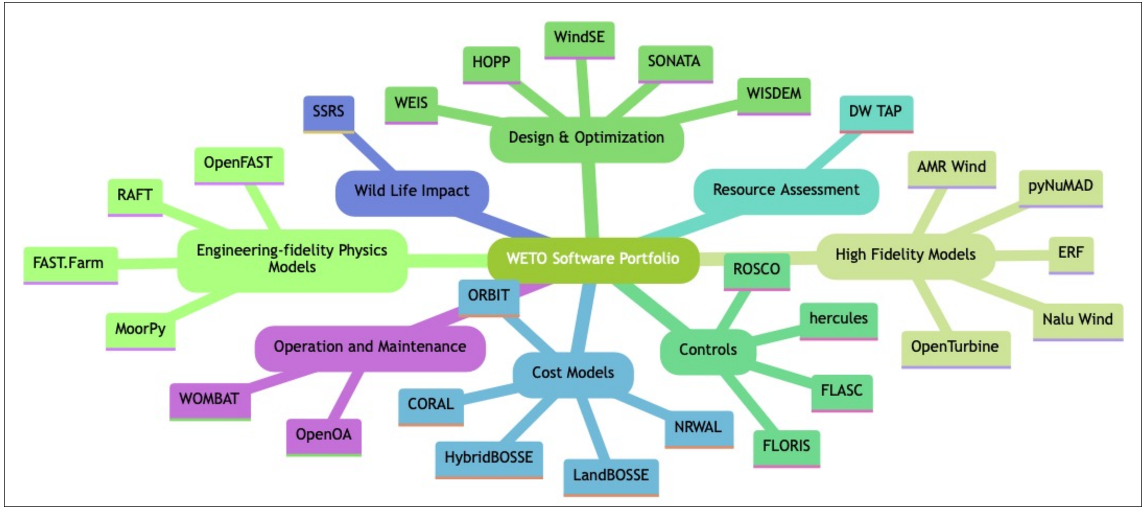


WETO Software Stack

Overview

WETO Software Stack

Grouped by what it does



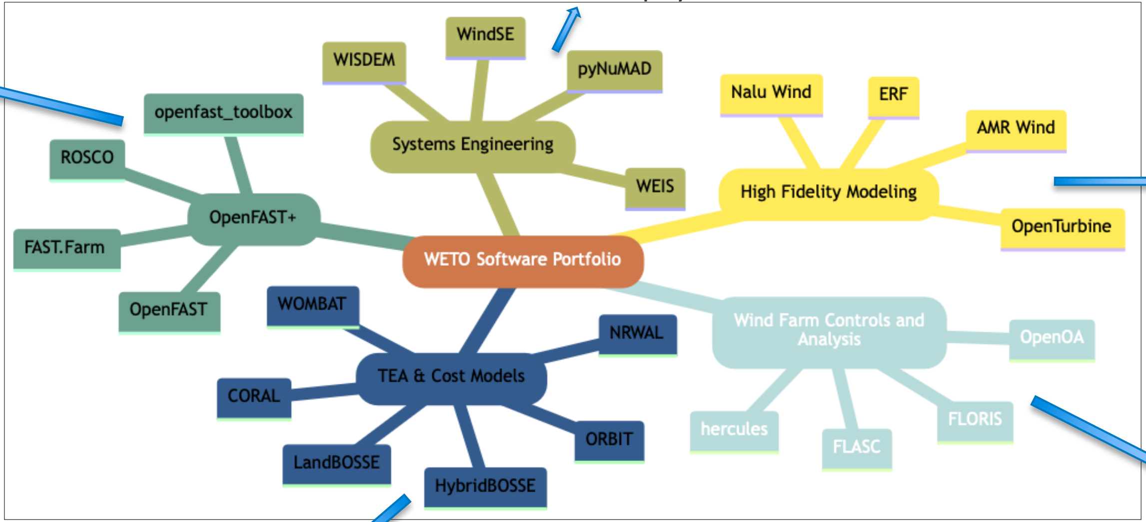
https://nrel.github.io/WETOStack/portfolio_analysis/software_list.html

WETO Software Stack

Grouped by how its used

System-level optimizations with multi-physics and multi-fidelities

Wind turbine and wind farm mid-fidelity physics



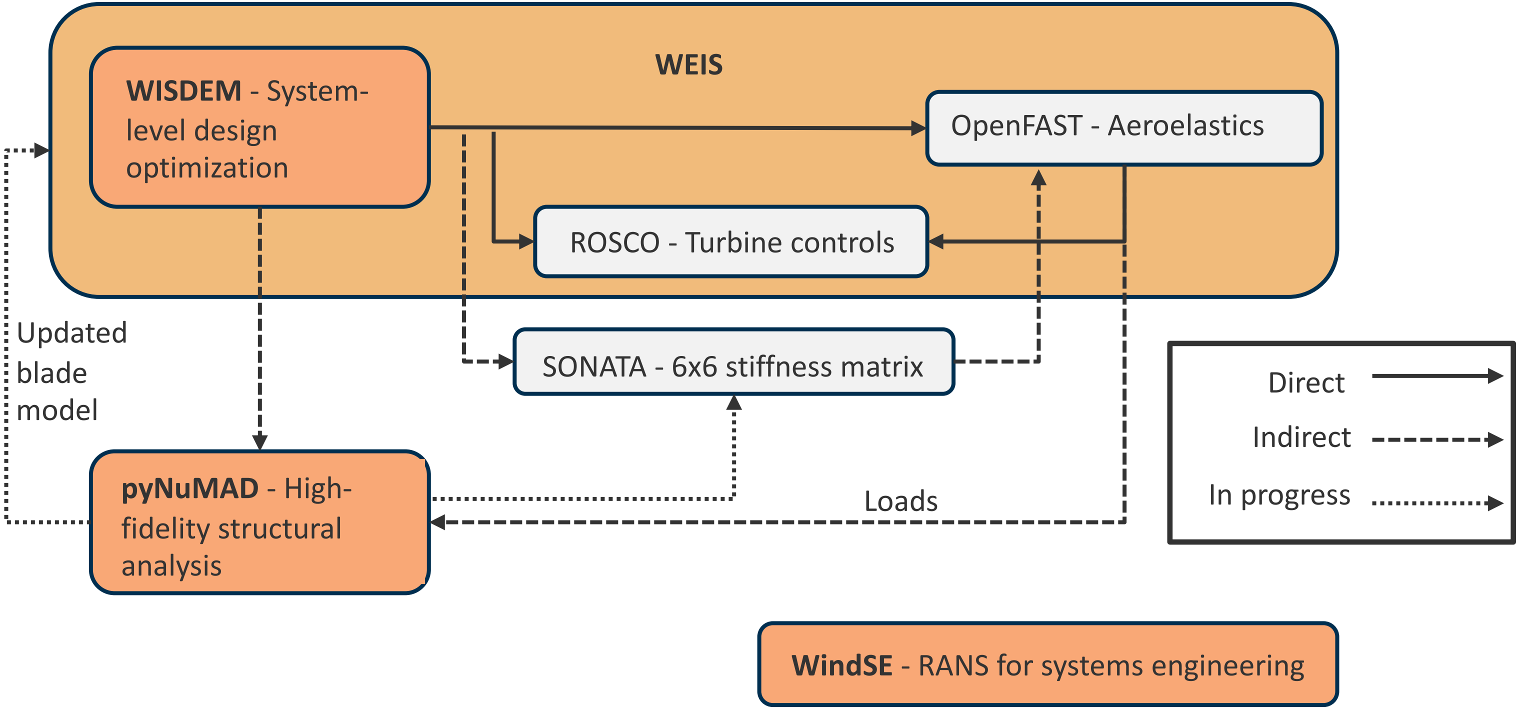
Detailed aerodynamics and structural dynamics

Wind turbine and wind plant costs for installation and operation, cost forecasting

Wind farm energy production, energy loss identification, and loss reduction through farm-level controls

Systems Engineering

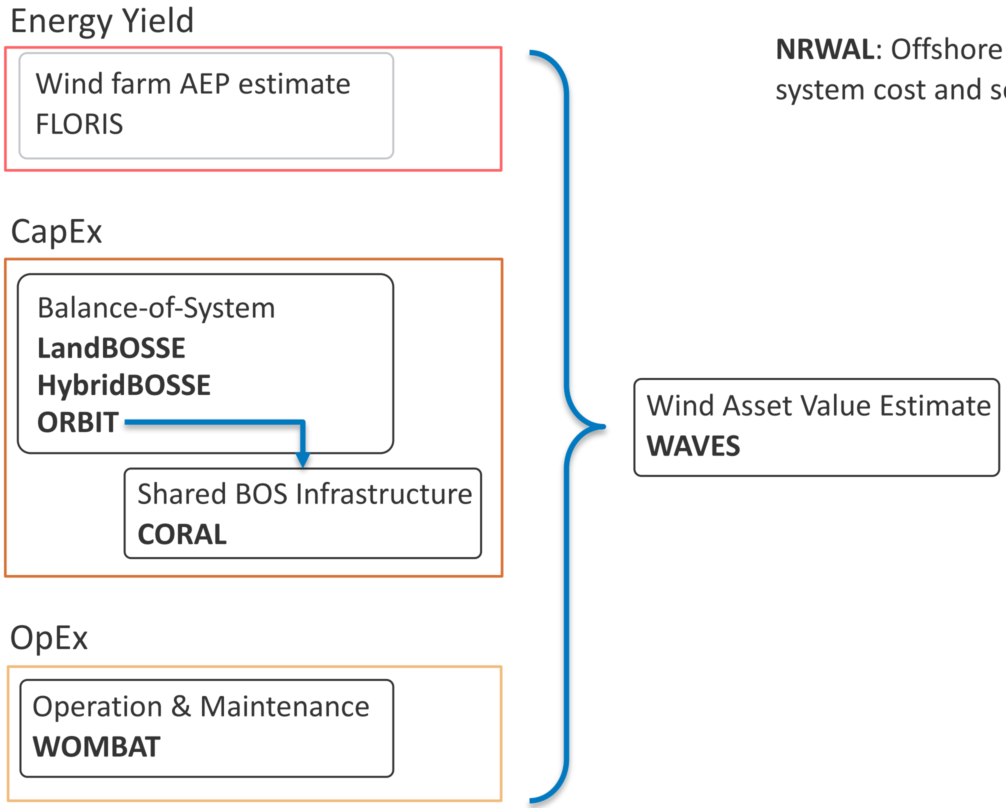
Workshop: June 10



Adapted from Big Adaptive Rotor (BAR) project

Technoeconomic Analysis / Cost Modeling

Workshop: June 12

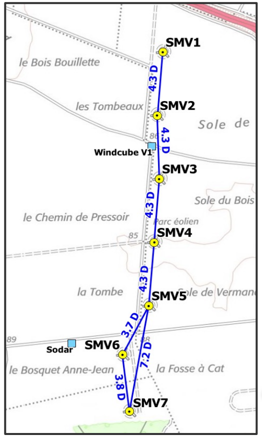


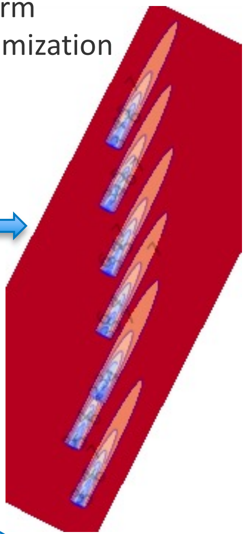
NRWAL: Offshore wind system cost and scaling model

Wind Farm Controls and Analysis

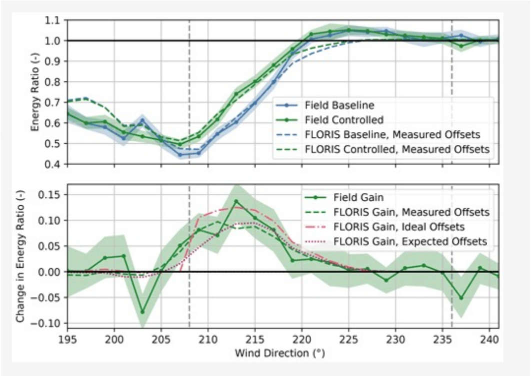
Workshop: June 18

FLORIS: Steady-state modeling, farm controls optimization

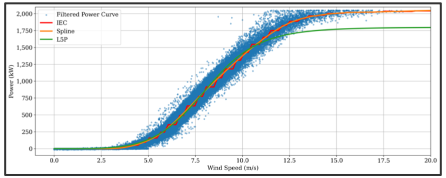




FLASC: Validate FLORIS model with SCADA, compare control methods



Hercules: Realtime high-fidelity simulator for hybrid power plants with a specific focus on wind farm controls.

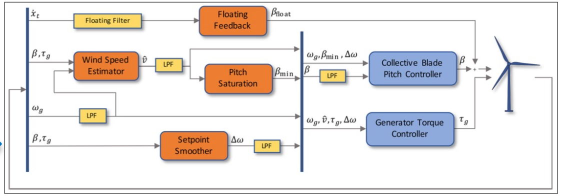


OpenOA: Characterize plant performance and quantify sources of operational loss

OpenFAST+

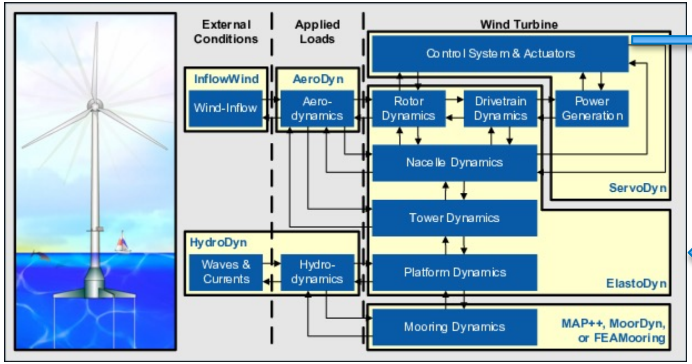
Workshop: June 20

ROSCO



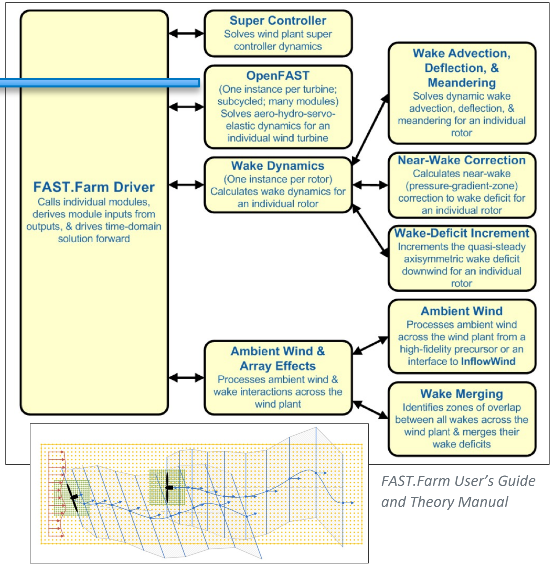
N. J. Abbas et al.: A reference controller for wind turbines

OpenFAST



OpenFAST v3.5.3 documentation

FAST.Farm



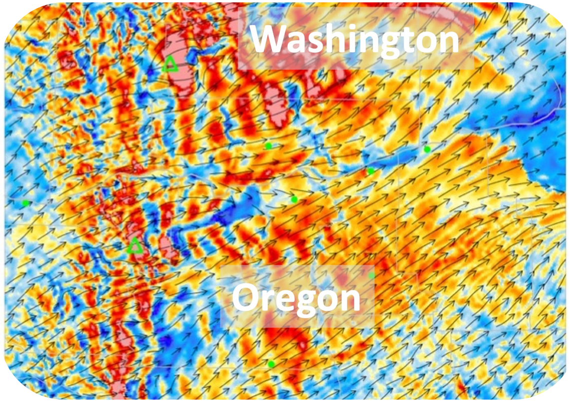
FAST.Farm User's Guide and Theory Manual

openfast_toolbox

High Fidelity Models

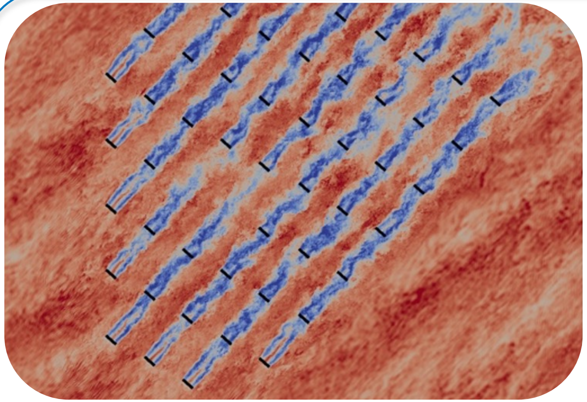
Workshop: TBD

ExaWind



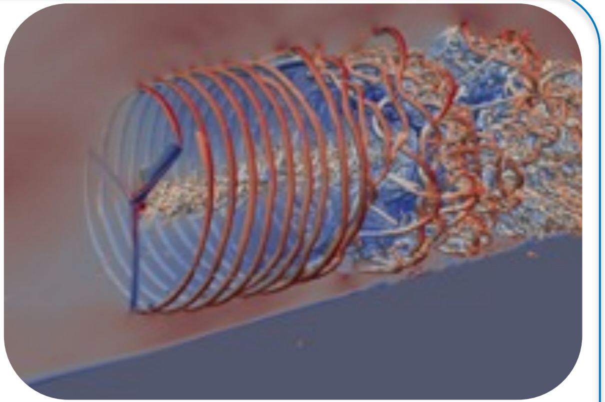
Mesoscale: ERF

- Regional scale weather
- **Scales 10 km to 1000 km**
- WRF numerics & models, built on AMReX
- GPU compatible
- Compressible



Microscale: AMR-Wind

- Atmospheric boundary layer
- **Scales less than 10 km**
- Large Eddy Simulation built on AMReX
- GPU compatible
- Structured grid with refinement zones
- Incompressible



Turbine scale: NALU-Wind

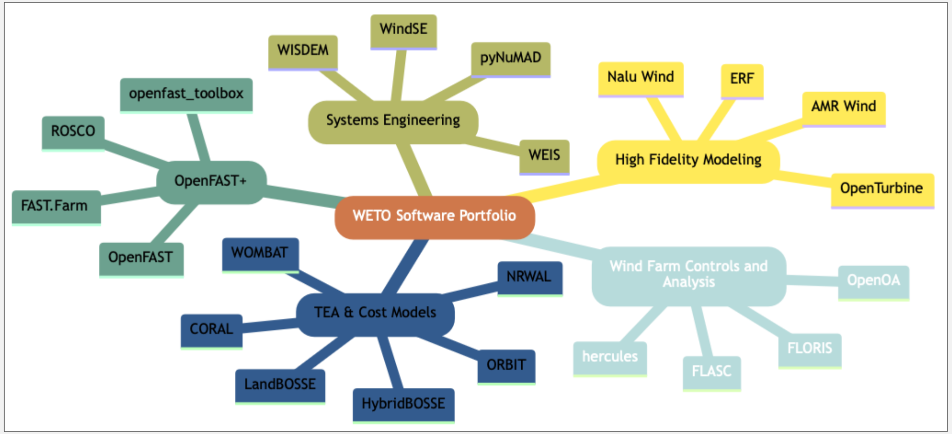
- Turbine, rotor, tower, nacelle
- **Scales less than 1 km**
- Unsteady Reynolds Averaged Navier Stokes
- GPU compatible
- Unstructured grid, geometry resolving
- Incompressible

WETO Software Stack

Open Discussion

WETO Software Stack

- **Discussion topics**
  - What's missing here?
  - What have been your primary pain points or bottlenecks?
  - What has or has not worked in integrating WETO software into your workflows?



Raise your "hand" and we'll call your name to ask your question.

FLORIS

Misha Sinner











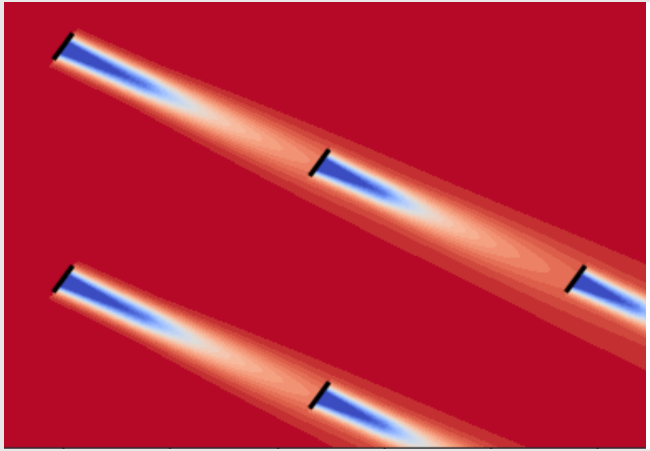

Wake models

Turbine models

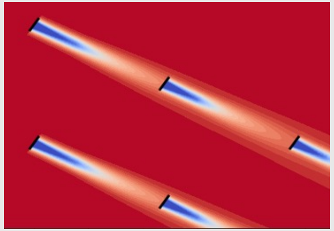
Wind data

Design tools

FLORIS



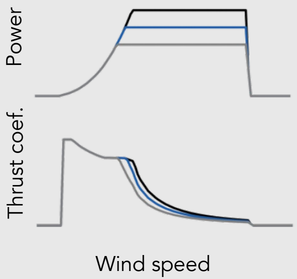
Wake models



Flow velocity deficit models

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

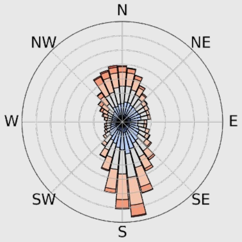
Turbine models



Actuator disks with power, thrust coefficient curves

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

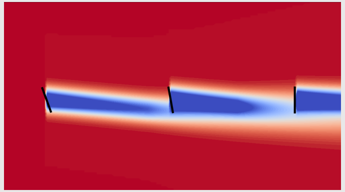
Wind data



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

```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

```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

```python
def get_farm_power(
    self,
    turbine_weights=None,
    use_turbulence_correction=False,
):
    """
    Report wind plant power from instance of floris. Optionally includes
    uncertainty in wind direction and yaw position when determining power.
    Uncertainty is included by computing the mean wind farm power for a
    distribution of wind direction and yaw position deviations from the
    original wind direction and yaw angles.

    Args:
        turbine_weights (NDArrayFloat | list[float] | None, optional):
            weighing terms that allow the user to emphasize power at
            particular turbines and/or completely ignore the power
            from other turbines. This is useful when, for example, you are
            modeling multiple wind farms in a single floris object. If you
            only want to calculate the power production for one of those
            farms and include the wake effects of the neighboring farms,
            you can set the turbine_weights for the neighboring farms'
            turbines to 0.0. The array of turbine powers from floris
            is multiplied with this array in the calculation of the
            objective function. If None, this  is an array with all values
            1.0 and with shape equal to (n_findex, n_turbines).
            Defaults to None.
        use_turbulence_correction: (bool, optional): When True uses a
            turbulence parameter to adjust power output calculations.
            Defaults to False. Not currently implemented.

    Returns:
        float: Sum of wind turbine powers in W.
    """
```

Search

FLORIS Wake Modeling & Wind Farm Controls

Getting Started

Installation

Switching from FLORIS v3 to v4

User Reference

Introductory Concepts

Advanced Concepts

Wind Data Objects

HeterogeneousMap

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

FLORIS Wake Modeling & Wind Farm Controls

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!

Quick Start

FLORIS is a Python package run on the command line typically by providing an input file with an initial configuration. 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()`.

```python
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()
```

Finally, results can be analyzed via post-processing functions available within `FlorisModel` such as `FlorisModel.get_turbine_layout()`, `FlorisModel.get_turbine_powers()` and `FlorisModel.get_farm_AEP()`, and a visualization package is available in `floris.flow_visualization`. A collection of examples are included in the repository and described in detail in examples.

Engaging on GitHub

FLORIS leverages the following GitHub features to coordinate support and development efforts:

- Discussions: Collaborate to develop ideas for new use cases, features, and software designs, and get support for usage questions
- Issues: Report potential bugs and well-developed feature requests
- Projects: Include current and future work on a timeline and assign a person to "own" it

Name	Last commit message
..	
examples_control_optimization	Refactor examples (#843)
examples_control_types	Peak shaving turbine operation model (#888)
examples_emgauss	Remove setpoints and wind condition specifics from calculate_X...
examples_floating	Add helix model operation mode (#842)
examples_get_flow	Refactor examples (#843)
examples_heterogeneous	Add z/3d to HeterogeneousMap (#915)
examples_layout_optimization	Randomized layout optimization (#697)
examples_multidim	Refactor examples (#843)
examples_turbine	Refactor examples (#843)
examples_uncertain	Add approximate FLORIS model (#877)
examples_visualizations	Remove setpoints and wind condition specifics from calculate_X...
examples_wind_data	Improvements to WindRose resampling (#857)
inputs	Update Empirical Gaussian default deflection_rate (#875)
inputs_floating	Update Empirical Gaussian default deflection_rate (#875)
001_opening_floris_computing_power.py	Refactor examples (#843)

Discussions

Upvotes	Discussion	Comments/Answers
1	**Wake model** — prith-gs asked on Dec 8, 2022 in Q&A · Answered	8
1	**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	0
2	**CubatureGrid design and validation** — rafmudaf started on Apr 18 in v4 Design Discussion	2
2	**Layout optimization in FLORIS** floris.optimization — Marcodep23 started on Apr 30 in General	3
1	**Crespo-Hernandez turbulence model** — cheoljoon asked last week in Q&A · Unanswered	0
1	**V3 Simulation Boundary Condition Influence** — nrz22 asked 3 weeks ago in Q&A · Unanswered	4

FLORIS v4 enables modular development

- New models for turbines operating in yaw and derating
- Implementing more wake models
- Data readers for compatibility with other tools
- Convergence, speed, and accuracy studies and enhancements

FLASC

Paul Fleming

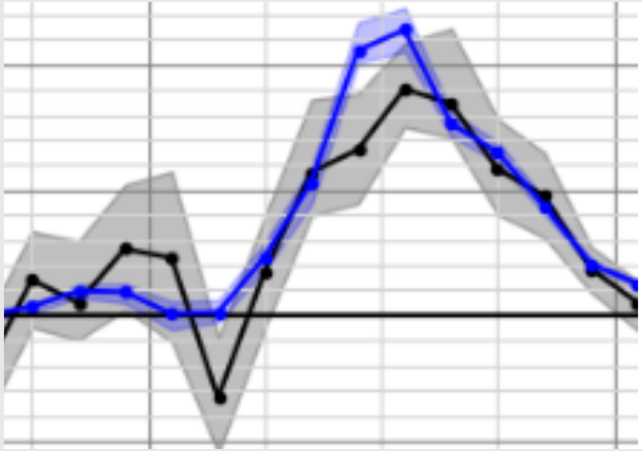
SCADA filtering

Bias correction

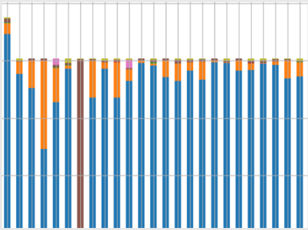
Uplift analysis

Model fitting

FLASC



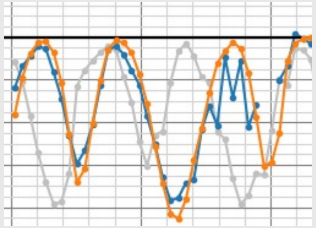
SCADA filtering



Filtering and outlier detection for power curves

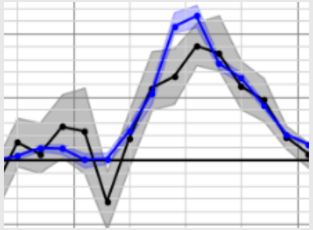
- Abnormal conditions
- Abnormal operation
- Stuck sensors

Bias correction



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

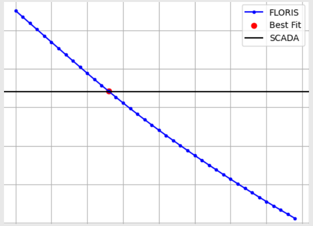
Uplift analysis



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

- Energy ratio
- Total uplift

Model fitting

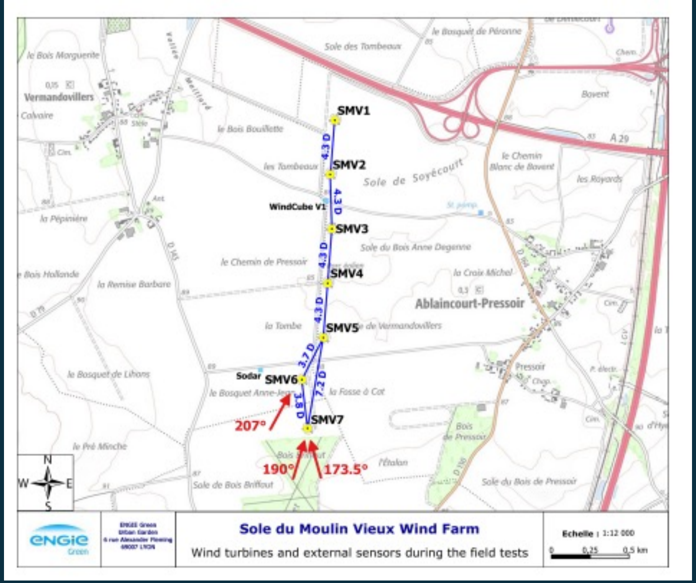


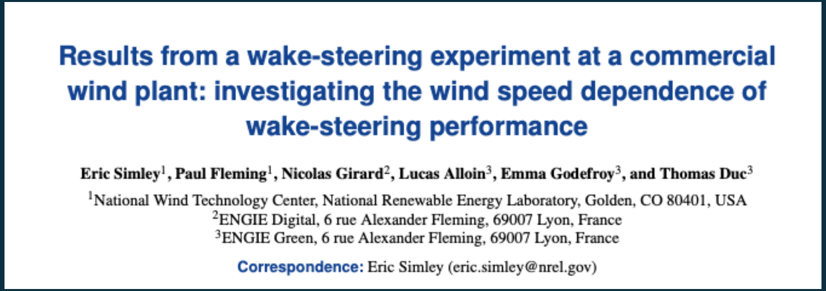
Parameter fitting for FLORIS turbine and wake models to SCADA records

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

Smarteole Example Set

- FLASC includes two sets of examples
- A first set demonstrates usage via artificially generated data sets
- A second set uses the Smarteole wake steering field trial data to illustrate usage





Results from a wake-steering experiment at a commercial wind plant: investigating the wind speed dependence of wake-steering performance

Eric Simley[1], Paul Fleming[1], Nicolas Girard[2], Lucas Alloin[3], Emma Godefroy[3], and Thomas Duc[3]
[1]National Wind Technology Center, National Renewable Energy Laboratory, Golden, CO 80401, USA
[2]ENGIE Digital, 6 rue Alexander Fleming, 69007 Lyon, France
[3]ENGIE Green, 6 rue Alexander Fleming, 69007 Lyon, France

Correspondence: Eric Simley (eric.simley@nrel.gov)

Example Files

Includes files go through the process of downloading assembling and cleaning the data and regenerate the paper results

- 00_inspect_floris_model.ipynb
- 01_precalculate_floris_solutions.ipynb

→ Prepares the FLORIS model

- 02_download_and_format_dataset.ipynb
- 03_filter_ws_power_curves.ipynb
- 04_northing_calibration.ipynb

→ Download and clean data

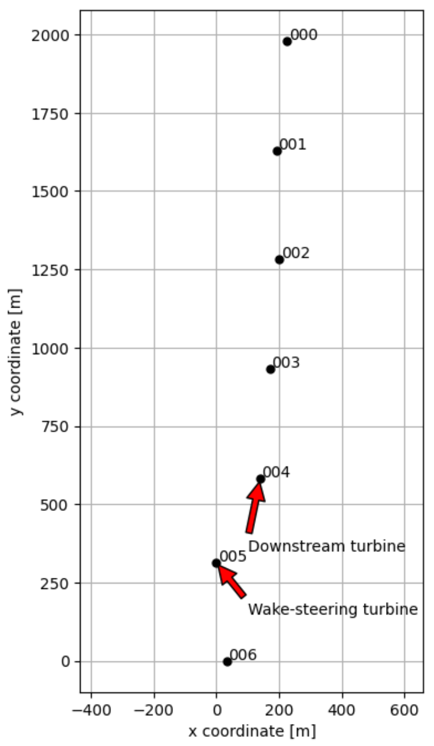
- 05_baseline_energy_ratio_analysis.ipynb
- 06_wake_steering_energy_ratio_analysis.ipynb

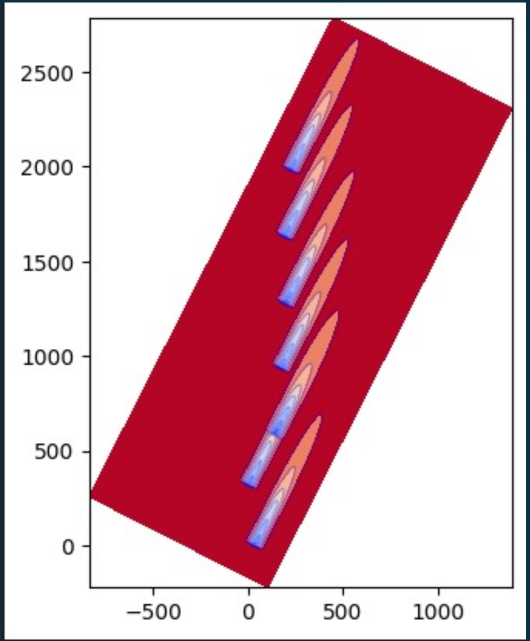
→ Analysis

- 07_emgauss_scada_tuning_optimization_method.ipynb
- 08_emgauss_tuning_day_night.ipynb

→ Model Tuning

Preparation – FLORIS Model





Preparation – Download and Cleaning

- FLASC uses a

```python
# Now map columns to conventional format
```

```python
df.head()
```

✓ 0.0s Python

	time	pow_000	pow_001	pow_002	pow_003	pow_004	pow_005	pow_006	ws_000	ws_001	...	is_operation_normal_004	is_operation_norr
0	2020-02-17 16:30:00	2023.746948	2045.376953	2031.724976	NaN	2028.063965	2032.461060	1983.390991	13.066	12.337	...	True	
1	2020-02-17 16:31:00	1959.036011	2050.572998	2034.890991	NaN	2017.777954	1943.764038	2046.568970	12.091	13.057	...	True	
2	2020-02-17 16:32:00	2053.658936	2032.191040	2011.870972	NaN	NaN	2052.092041	2039.948975	13.381	12.213	...	None	
3	2020-02-17 16:33:00	2044.296997	2060.478027	1995.057983	NaN	NaN	2008.868042	2058.000000	14.345	13.141	...	None	
4	2020-02-17 16:34:00	2058.281006	2042.703003	2031.723999	NaN	NaN	1819.896973	2059.760010	14.338	12.723	...	None	

```python
# df_list = []
print("formatting dataframe...")
df_scada = df_scada.rename(columns=scada_dict)
```

Standard names and 0-indexing

```
df.head()
```

✓ 0.0s Python

	time	pow_000	pow_001	pow_002	pow_003	pow_004	pow_005	pow_006	ws_000	ws_001	...	is_operation_normal_004	is_operation_norr
0	2020-02-17 16:30:00	2023.746948	2045.376953	2031.724976	NaN	2028.063965	2032.461060	1983.390991	13.066	12.337	...	True	
1	2020-02-17 16:31:00	1959.036011	2050.572998	2034.890991	NaN	2017.777954	1943.764038	2046.568970	12.091	13.057	...	True	
2	2020-02-17 16:32:00	2053.658936	2032.191040	2011.870972	NaN	NaN	2052.092041	2039.948975	13.381	12.213	...	None	
3	2020-02-17 16:33:00	2044.296997	2060.478027	1995.057983	NaN	NaN	2008.868042	2058.000000	14.345	13.141	...	None	
4	2020-02-17 16:34:00	2058.281006	2042.703003	2031.723999	NaN	NaN	1819.896973	2059.760010	14.338	12.723	...	None	

UTC timestamp

NaNs for missing/faulty data

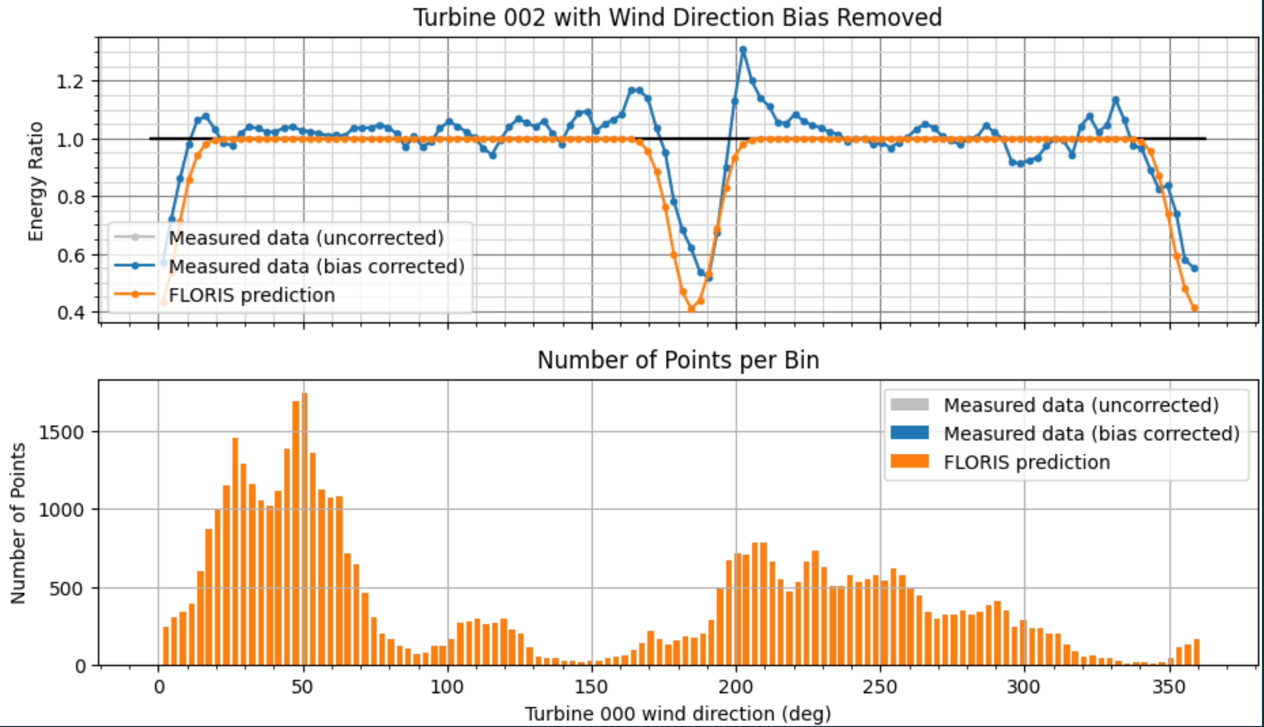
Estimating Bias

- Use FLASC to automatically identify freestream turbines
- Set reference wind speed, direction and reference power by these sets and proximity to a test turbine

```python
# Calculate which turbines are upstream for every wind direction
df_upstream = ftools.get_upstream_turbs_floris(fm, wd_step=2.0)

# We assign the total datasets "true" wind direction as equal to the wind
# direction of the turbine which we want to perform northing calibration
# on. In this case, turbine 'ti'.
df = dfm.set_wd_by_turbines(df, [ti])

# We define a function that calculates the freestream wind speed based
# on a dataframe that is inserted. It does this based on knowing which
# turbines are upstream for what wind directions, and then knowledge
# of what the wind direction is for every row in the dataframe. However,
# since the shift the "true" wind direction many times to estimate the
# northing bias, we cannot precalculate this. It changes with every
# northing bias guess. Hence, we must insert a function.
def _set_ws_fun(df):
    return dfm.set_ws_by_upstream_turbines_in_radius(
        df=df,
        df_upstream=df_upstream,
        turb_no=ti,
        x_turbs=fm.layout_x,
        y_turbs=fm.layout_y,
        max_radius=5000.0,
        include_itself=True,
    )

# We similarly define a function that calculates the reference power. This
# is typically the power production of one or multiple upstream turbines.
# Here, we assume it is the average power production of all upstream
# turbines. Which turbines are upstream depends on the wind direction.
def _set_pow_ref_fun(df):
    return dfm.set_pow_ref_by_upstream_turbines_in_radius(
        df=df,
        df_upstream=df_upstream,
        turb_no=ti,
        x_turbs=fm.layout_x,
        y_turbs=fm.layout_y,
        max_radius=5000.0,
        include_itself=True,
    )
```

Estimating Bias

```
fsc = best.bias_estimation(
    df=df,
    df_fm_approx=df_approx,
    test_turbines_subset=test_turbines,
    df_ws_mapping_func=_set_ws_fun,
    df_pow_ref_mapping_func=_set_pow_ref_fun,
)
```



Energy Ratio and Uplift

- The data is split by control mode

```
# Split df_scada into baseline and wake steering on "controlled"
# periods
df_base = df_scada[df_scada.control_mode == "baseline"]
df_con = df_scada[df_scada.control_mode == "controlled"]
```

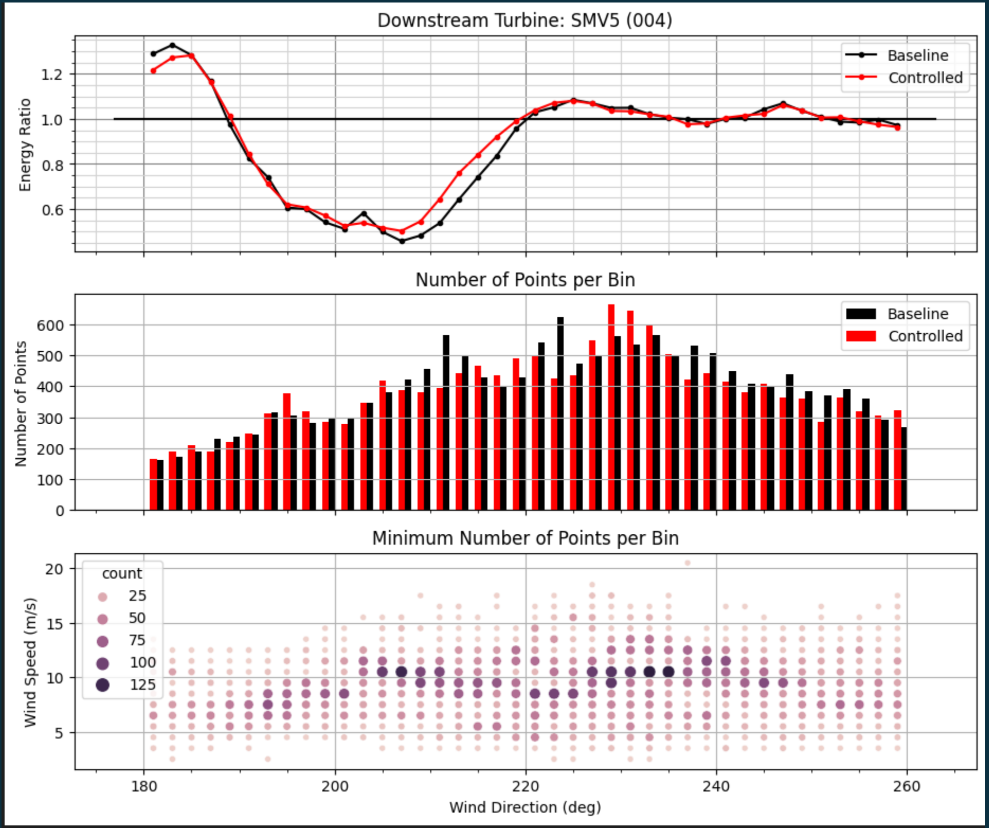
- Energy Ratio Input Built

```
# Construct energy ratio input object using default 10 blocks
er_in = EnergyRatioInput([df_base, df_con], ["Baseline", "Controlled"])
er_colors = {"Baseline": "black", "Controlled": "red"}
```

- Calculate Energy Ratios

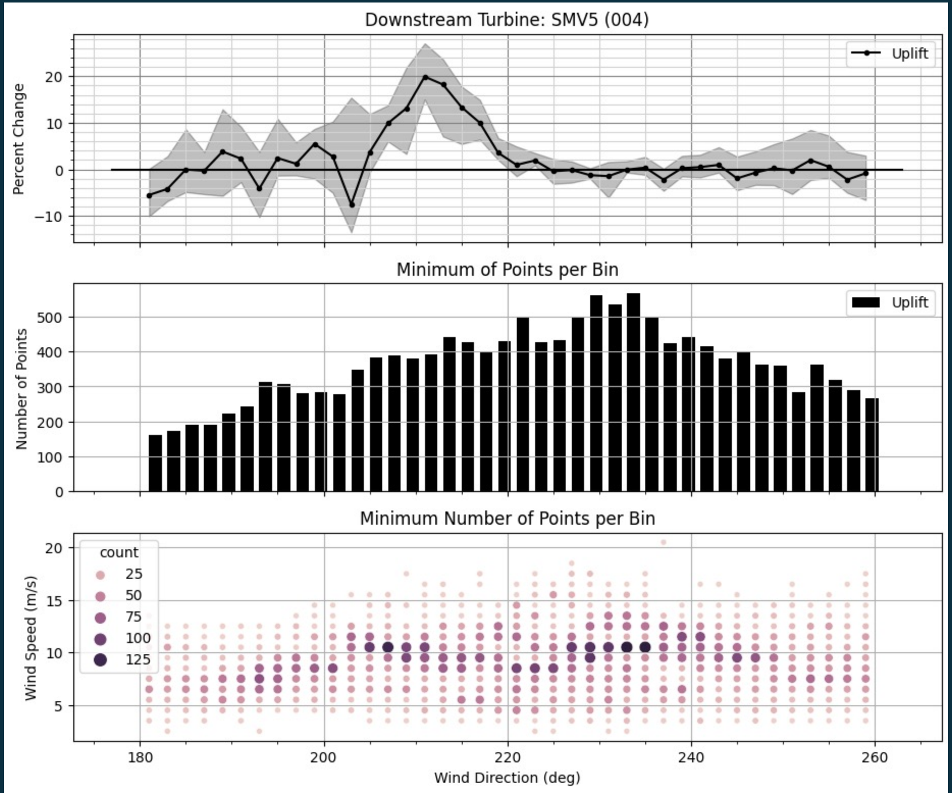
```
# Check energy ratios on SMV5 (index=4) (downstream turbine)
er_out = er.compute_energy_ratio(
    er_in,
    test_turbines=[4],
    use_predefined_ref=True,
    use_predefined_wd=True,
    use_predefined_ws=True,
    wd_step=2.0,
    ws_step=1.0,
)
ax = er_out.plot_energy_ratios(color_dict=er_colors)
ax[0].set_title("Downstream Turbine: SMV5 (004)")
```

Energy Ratio



Uplift with bootstrapping

```
# SMV5 (Downstream)
er_out = er.compute_energy_ratio(
    er_in,
    test_turbines=[4],
    use_predefined_ref=True,
    use_predefined_wd=True,
    use_predefined_ws=True,
    wd_step=2.0,
    ws_step=1.0,
    uplift_pairs=[("Baseline", "Controlled")],
    uplift_names=["Uplift"],
    N=40,
)
ax = er_out.plot_uplift(color_dict={"Uplift": "black"})
ax[0].set_title("Downstream Turbine: SMV5 (004)")
```



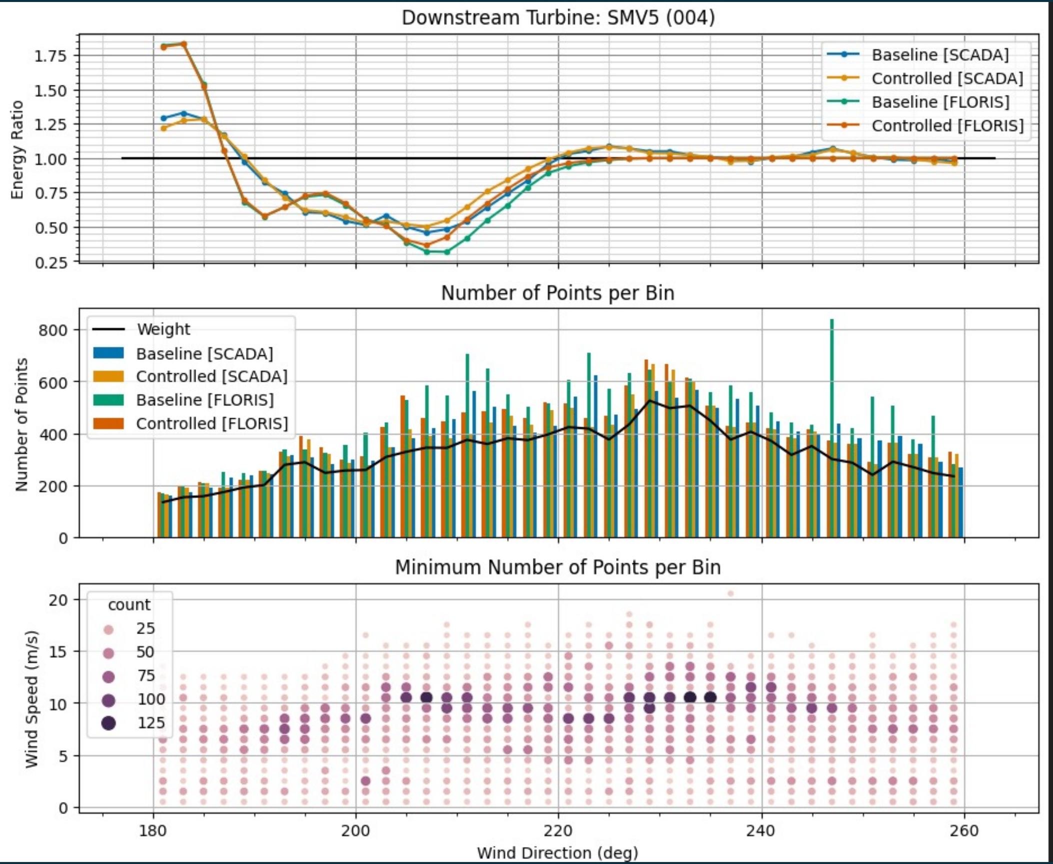
Comparison with FLORIS

```python
# Resimulate FLORIS using time-domain sim assuming all
# yaws are 0 except for SMV5, which follows either target exactly or
# what is measured via the vane

# Baseline / Perfect yawing
wind_speeds_baseline = df_base.ws.values
wind_directions_baseline = df_base.wd.values
yaw_angles_baseline_target = None
yaw_angles_baseline_measured = df_base.wind_vane_005.values

wind_speeds_con = df_con.ws.values
wind_directions_con = df_con.wd.values
yaw_angles_con_target = df_con.target_yaw_offset_005.values
yaw_angles_con_measured = df_con.wind_vane_005.values

# Compute FLORIS assuming target offsets and no wd std
df_floris_target_offset_baseline = build_floris_data_frame(
    fm, wind_speeds_baseline, wind_directions_baseline, yaw_angles_baseline_target
)
df_floris_target_offset_con = build_floris_data_frame(
    fm, wind_speeds_con, wind_directions_con, yaw_angles_con_measured
)
```

Comparison with FLORIS

```
# Construct energy ratio object
er_in = EnergyRatioInput(
    [df_base, df_con, df_floris_target_offset_baseline, df_floris_target_offset_con],
    ["Baseline [SCADA]", "Controlled [SCADA]", "Baseline [FLORIS]", "Controlled [FLORIS]"],
)

er_out = er.compute_energy_ratio(
    er_in,
    test_turbines=[4],
    use_predefined_ref=True,
    use_predefined_wd=True,
    use_predefined_ws=True,
    wd_step=2.0,
    ws_step=1.0,
)
ax = er_out.plot_energy_ratios(overlay_frequency=True)
ax[0].set_title("Downstream Turbine: SMV5 (004)")
```

Comparison with FLORIS



Total Uplift

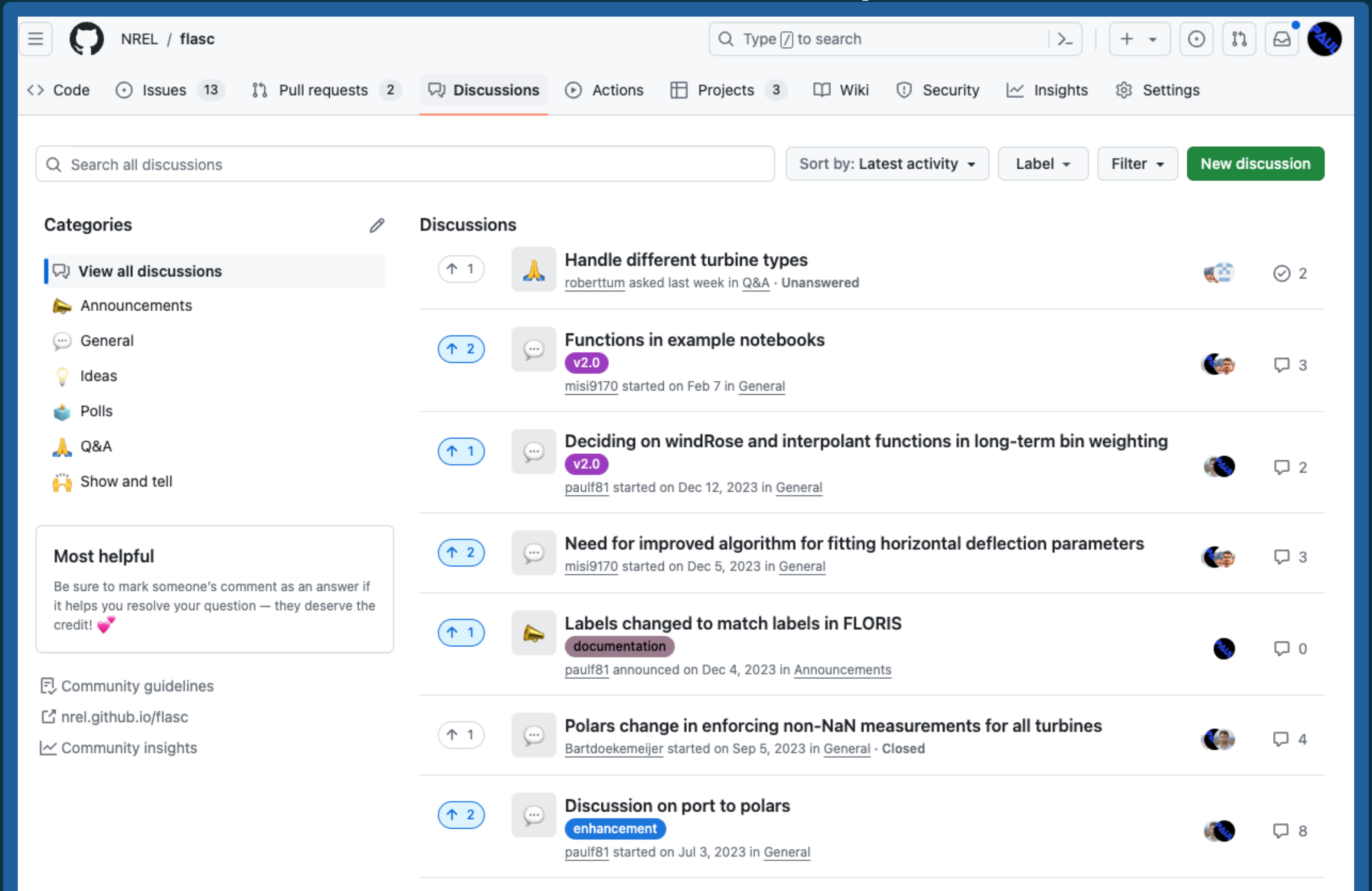
```python
# Downstream only
total_uplift_result = tup.compute_total_uplift(
    er_in,
    test_turbines=[4, 5],
    use_predefined_ref=True,
    use_predefined_wd=True,
    use_predefined_ws=True,
    wd_step=2.0,
    wd_min=195.0,  # As in paper
    wd_max=240.0,  # As in paper
    ws_step=1.0,
    ws_min=4.0,
    uplift_pairs=[
        ("Baseline [SCADA]", "Controlled [SCADA]"),
        ("Baseline [FLORIS]", "Controlled [FLORIS]"),
    ],
    uplift_names=["Uplift [SCADA]", "Uplift [FLORIS]"],
    N=100,
    percentiles=(10, 90),  # Use P10 and P90
)


print(
    f"Percent increase in total energy production for combined turbines: "
    f"{total_uplift_result['Uplift [SCADA]']['energy_uplift_ctr_pc']:.3f}% (SCADA)"
)
print(
    f"Percent increase in total energy production for combined turbines: "
    f"{total_uplift_result['Uplift [FLORIS]']['energy_uplift_ctr_pc']:.3f}% (FLORIS)"
)
print(" ")
print("Full contents of dictionary including P10 and P90 values...")
total_uplift_result
```

Total Uplift

```python
# Downstream only
total_uplift_result = tup.compute_total_uplift(
    er_in,
    test_turbines=[4, 5],
    use_predefined_ref=True,
    use_predefined_wd=True,
    use_predefined_ws=True,
    wd_step=2.0,
    wd_min=195.0,  # As in paper
    wd_max=240.0,  # As in paper
    ws_step=1.0,
    ws_min=4.0,
    uplift_pairs=[
        ("Baseline [SCADA]", "Controlled [SCADA]"),
        ("Baseline [FLORIS]", "Controlled [FLORIS]"),
    ],
    uplift_names=["Uplift [SCADA]", "Uplift [FLORIS]"],
    N=100,
    percentiles=(10, 90),  # Use P10 and P90
)


print(
    f"Percent increase in total energy production for combined turbines: "
    f"{total_uplift_result['Uplift [SCADA]']['energy_uplift_ctr_pc']:.3f}% (SCADA)"
)
print(
    f"Percent increase in total energy production for combined turbines: "
    f"{total_uplift_result['Uplift [FLORIS]']['energy_uplift_ctr_pc']:.3f}% (FLORIS)"
)
print(" ")
print("Full contents of dictionary including P10 and P90 values...")
total_uplift_result
```

```
Percent increase in total energy production for combined turbines: 0.446% (SCADA)
Percent increase in total energy production for combined turbines: 0.677% (FLORIS)
```

FLASC Documentation and Examples



FLASC v2 streamlines package

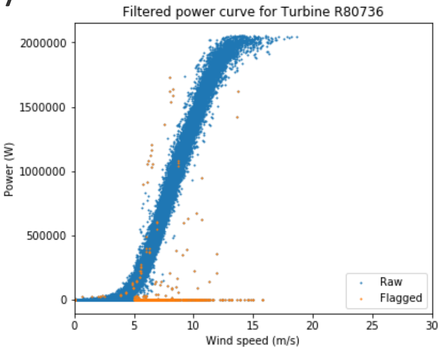
- Expand on model fitting capabilities
- Implement further methods for assessment of power, energy differences
- Make front end simpler by adding convenience functions to package
- Build out documentation further, add API docs

OpenOA

Eric Simley & Rob Hammond

Overview of OpenOA

- Open Operational Assessment
  - **PlantData and PlantMetaData** classes for organizing data
  - **Utils** toolkits for lower-level wind plant data operations
  - **Analysis** methods for performing specific operational analyses
- Documented examples
  - Example **Jupyter Notebooks** illustrating all operational analysis methods



PlantData
- Asset
- Rev. Meter
- Curtailment
- SCADA
- Met. Tower
- Reanalysis
- ...

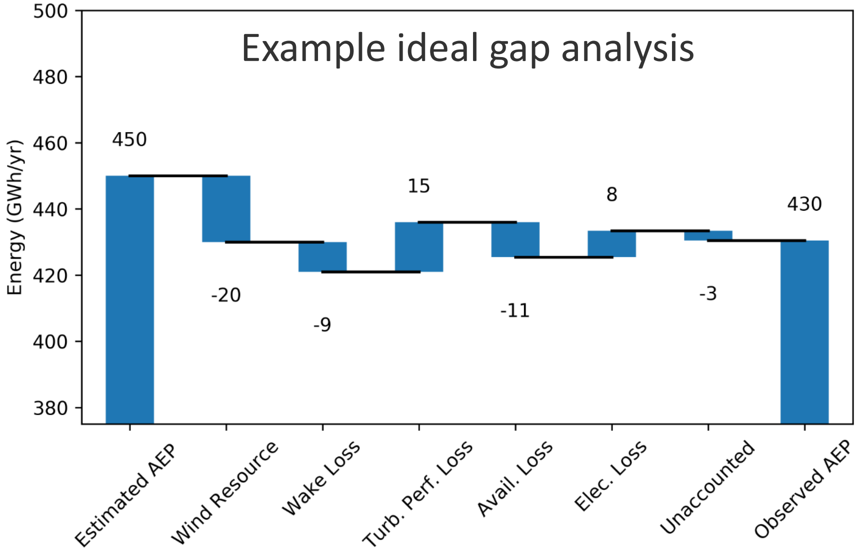
Utils
- Quality Assurance
- Filters
- Met. Data Processing
- Power Curves
- Reanalysis Downloading
- Plotting
- ...

OA Methods
- AEP Analysis
- Turbine Ideal Energy
- Electrical Losses
- EYA Gap Analysis
- Wake Loss Analysis
- Static Yaw Misalignment
- ...

Objective of the Software: Performing Gap Analyses

- AEP = Gross Energy – Losses
- Loss categories in Energy Yield Assessment (EYA) process
  - Availability
  - Electrical
  - Environmental
  - Turbine performance
  - Wake effects

- Can we determine how each EYA category contributes to **gap in AEP estimates**?

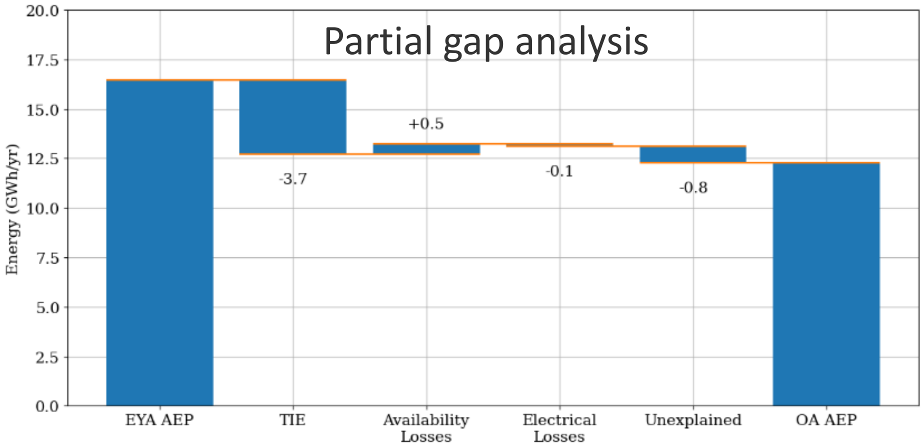


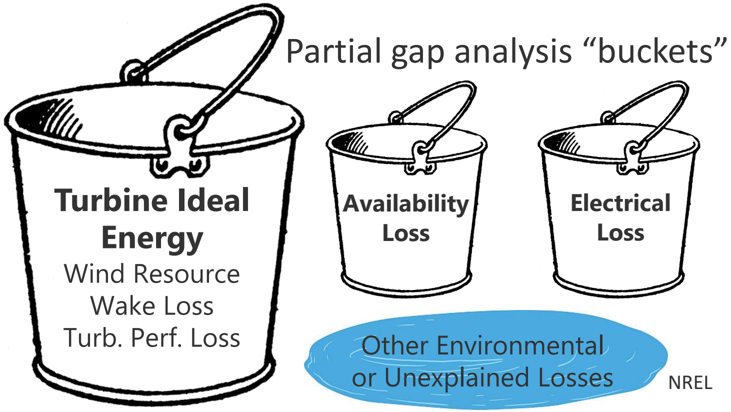
Objective of the Software: Performing Gap Analyses

- AEP = Gross Energy – Losses
- Loss categories in Energy Yield Assessment (EYA) process
  - Availability
  - Electrical
  - Environmental
  - Turbine performance
  - Wake effects

- Can we determine how each EYA category contributes to **gap in AEP estimates**?

- We can perform a **partial gap analysis**





PlantData and PlantMetaData Classes

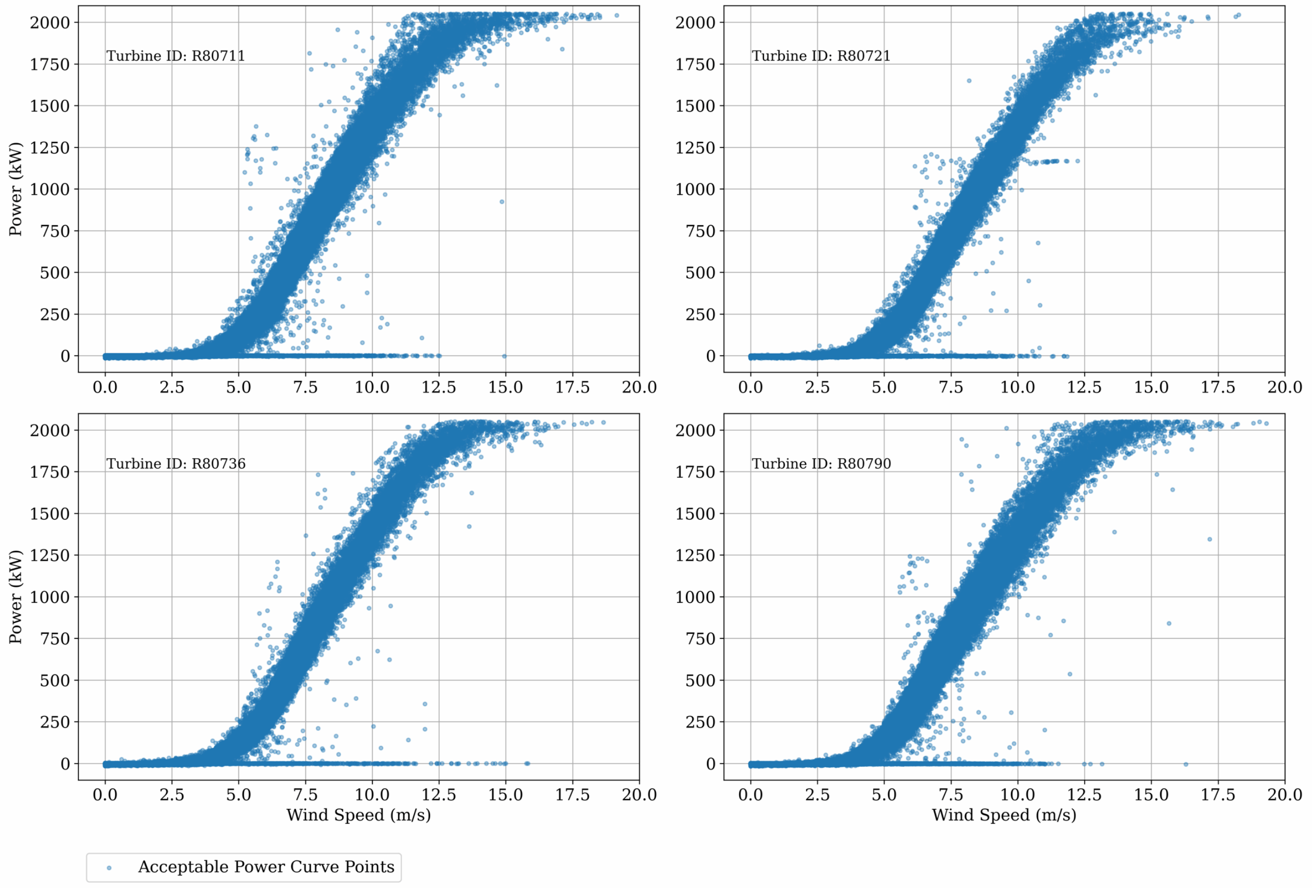
- OpenOA **PlantData** object
  - List of analysis types for which the data will be validated
  - Pandas DataFrames or CSV files for each operational data type
  - Metadata file
    - Maps tag names from user-provided data to OpenOA IEC 61400-25 convention

```
project = PlantData(
    analysis_type=["MonteCarloAEP", "ElectricalLosses"],
    metadata="data/plant_meta.yml",
    scada=scada_df,
    meter=meter_df,
    curtail=curtail_df,
    asset=asset_df,
    reanalysis=reanalysis_dict,
)
```

```
scada:
  frequency: 10T  # timestamp frequency
  asset_id: Wind_turbine_name  # Unique ID of wind turbine
  WROT_BlPthAngVal: Ba_avg  # pitch angle, degrees
  WTUR_W: P_avg  # power produced, kW
  WMET_EnvTmp: Ot_avg  # temperature, C
  time: Date_time  # timestamps
  WMET_HorWdDir: Wa_avg  # wind direction, degrees
  WMET_HorWdDirRel: Va_avg  # wind direction relative to nacelle orientation
  WMET_HorWdSpd: Ws_avg  # non-directional windspeed, m/s
```

Wind_turbine_name	Date_time	Ba_avg	P_avg	Ws_avg	Va_avg	Ot_avg	Ya_avg	Wa_avg
R80736	2014-01-01T01:00:00+01:00	-1	642.78003	7.1199999	0.66000003	4.6900001	181.34	182.00999
R80721	2014-01-01T01:00:00+01:00	-1.01	441.06	6.3899999	-2.48	4.9400001	179.82001	177.36
R80790	2014-01-01T01:00:00+01:00	-0.95999998	658.53003	7.1100001	1.0700001	4.5500002	172.39	173.50999
R80711	2014-01-01T01:00:00+01:00	-0.93000001	514.23999	6.8699999	6.9499998	4.3000002	172.77	179.72
R80790	2014-01-01T01:10:00+01:00	-0.95999998	640.23999	7.0100002	-1.9	4.6799998	172.39	170.46001
R80736	2014-01-01T01:10:00+01:00	-1	511.59	6.6900001	-3.3399999	4.6999998	181.34	178.02
R80711	2014-01-01T01:10:00+01:00	-0.93000001	692.33002	7.6799998	4.7199998	4.3800001	172.77	177.49001

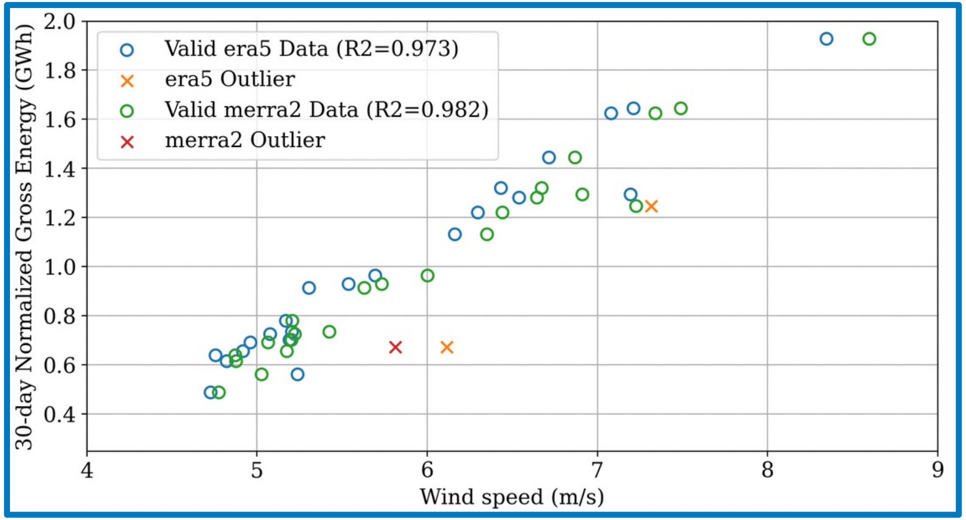
Power Curve Filtering



Running Long-Term AEP Estimation

```python
pa = MonteCarloAEP(
    project,
    reanalysis_products = ['era5', 'merra2'],
    time_resolution="M",
    reg_model="lin"
)
```

```python
pa.plot_reanalysis_gross_energy_data(
    outlier_threshold=3,
    xlim=(4, 9),
    ylim=(0.25, 2),
    plot_kwargs=dict(s=60)
)
```



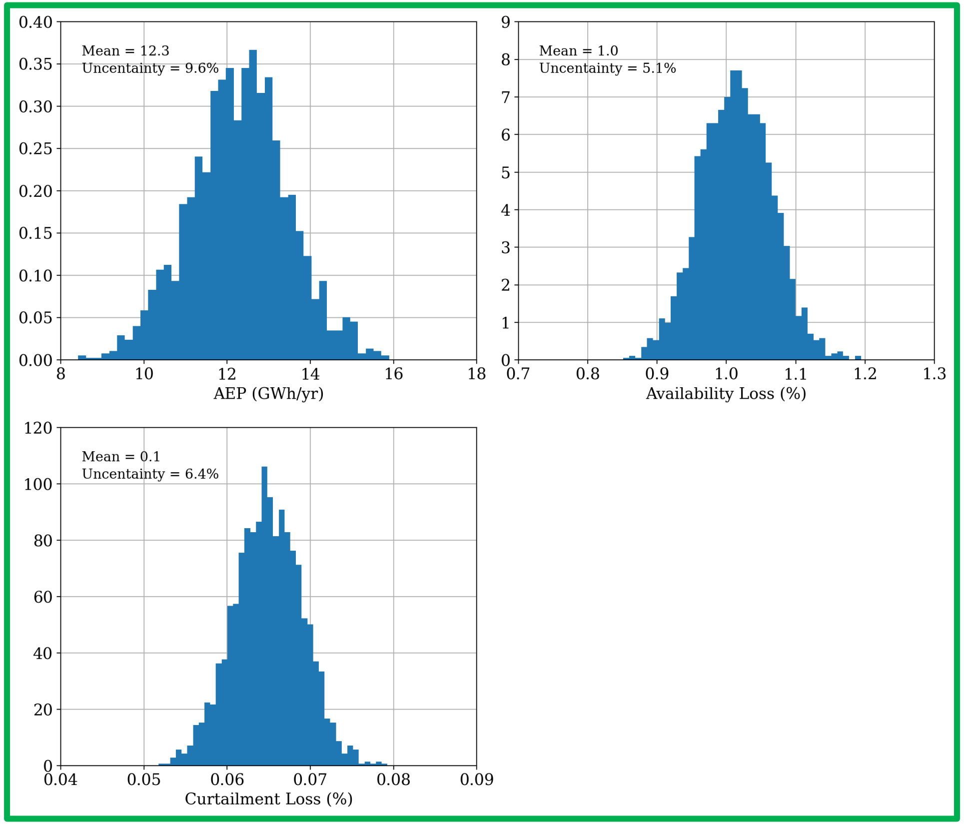
Running Long-Term AEP Estimation

```python
pa = MonteCarloAEP(
    project,
    reanalysis_products = ['era5', 'merra2'],
    time_resolution="M",
    reg_model="lin"
)
```

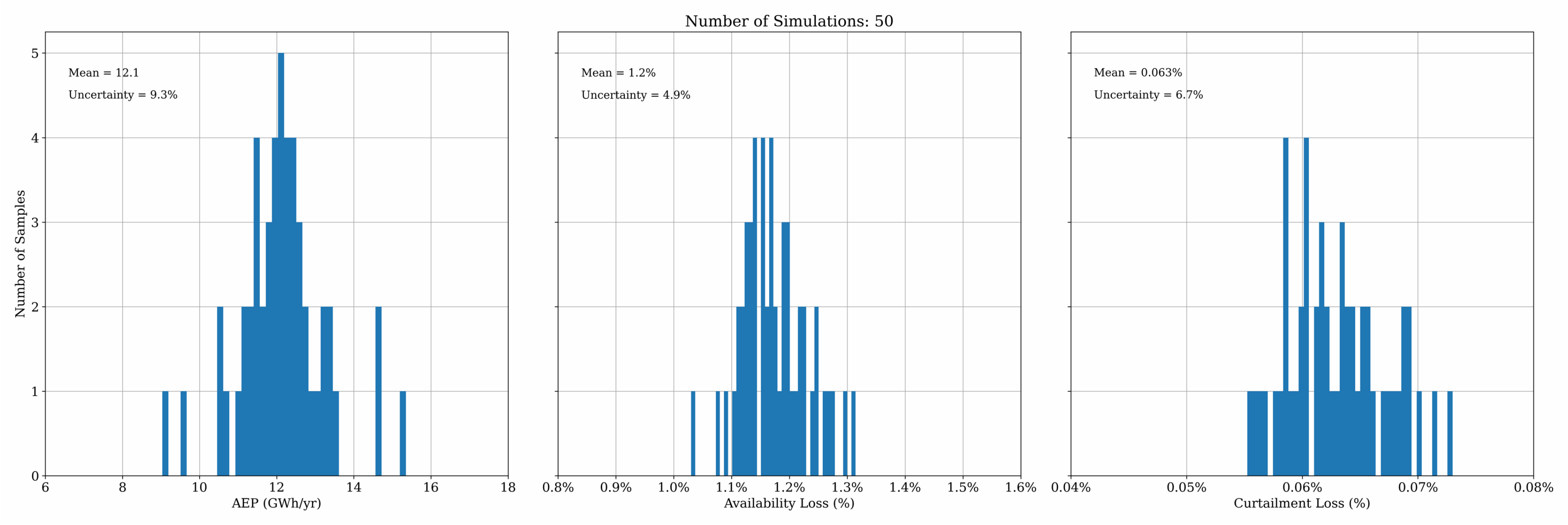
```python
pa.plot_reanalysis_gross_energy_data(
    outlier_threshold=3,
    xlim=(4, 9),
    ylim=(0.25, 2),
    plot_kwargs=dict(s=60)
)
```

```python
pa.run(num_sim=2000)
```

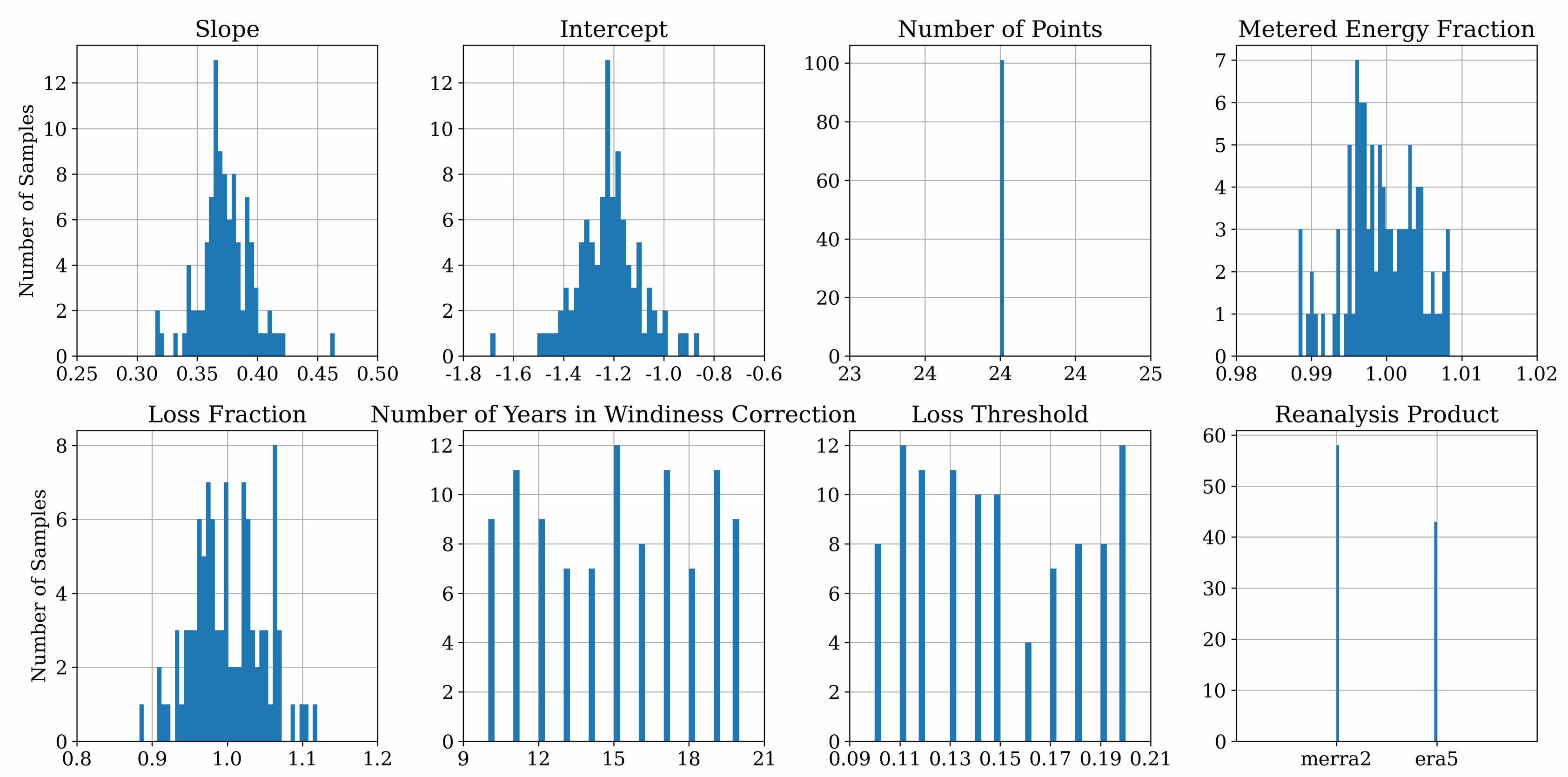
```python
pa.plot_result_aep_distributions()
```



Convergence of AEP Results



Model Parameter Distribution



Software Development Roadmap

OpenOA

- Near-Term Priorities
  - Account for spatial wind speed heterogeneity in Wake Loss method
  - Add hourly reanalysis downloading
  - Outreach and promotion (e.g., WeDoWind challenges)
  - Improve reliability of Yaw Misalignment method

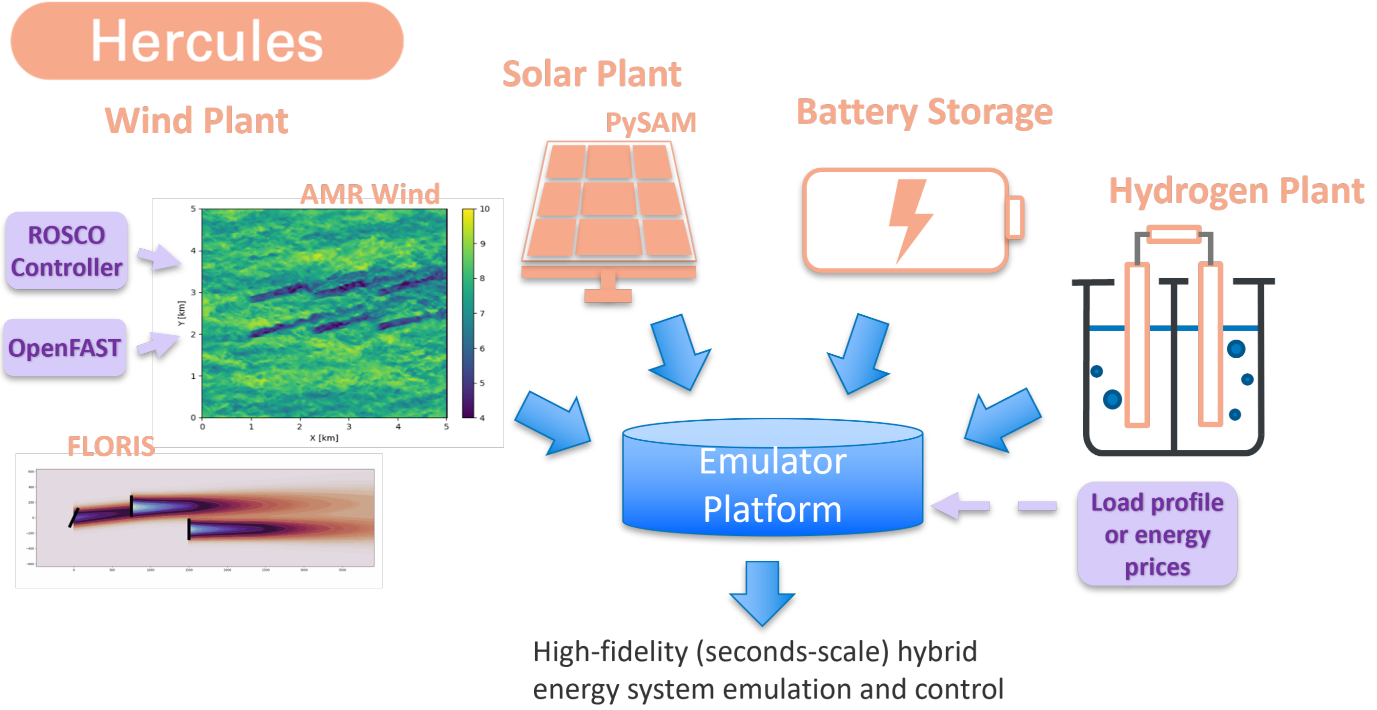
- Long-Term Priorities
  - Expand analysis methods to meet industry needs
    - Energy-based availability, performance degradation, failure prediction
  - Expand data sources (e.g., SQL, Greenbyte)
  - Perform and publish validation studies
  - Expand business adoption of OpenOA as a 3rd party validation tool

Hercules

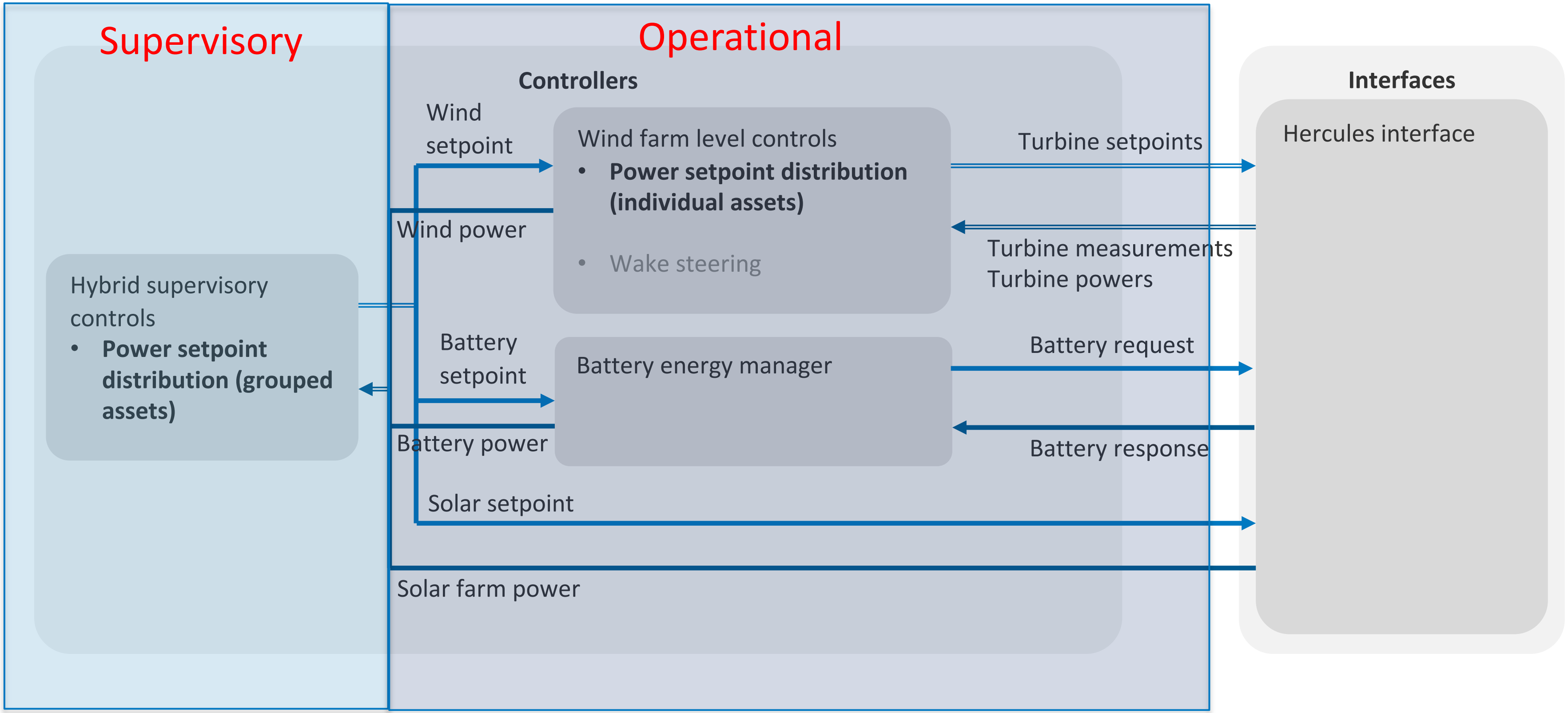
Misha Sinner & Gen Starke

Hercules: Hybrid Energy and Control Using Large Eddy Simulations

Hercules is an **open-source tool** for wind-based hybrid plant simulation in **real time**. Hercules is based around high fidelity wind farm flow simulations through AMR-Wind, co-simulated with a hybrid plant that includes solar, storage, and electrolysis.



Wind-Hybrid Open Controller (WHOC)

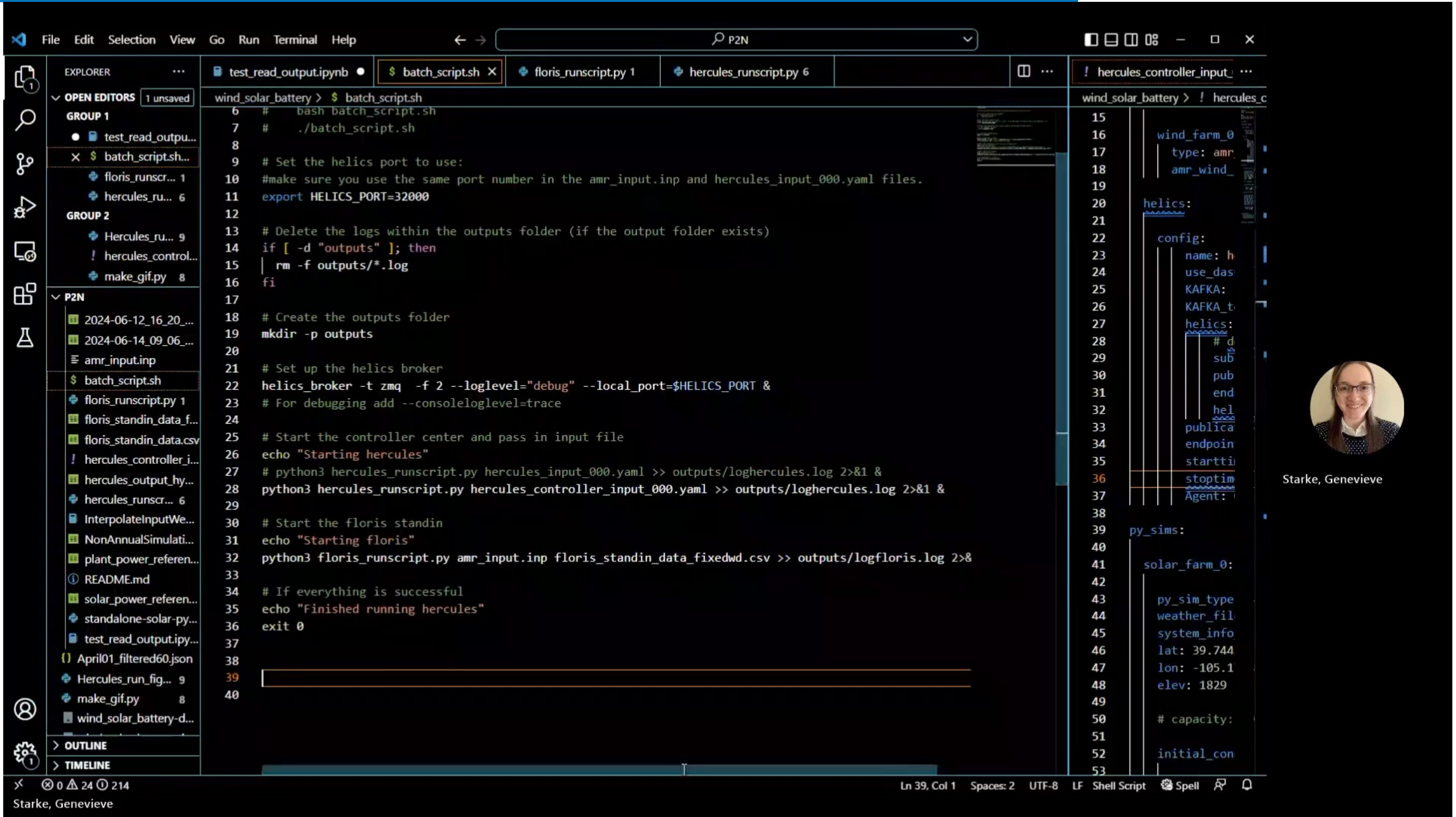


Hercules Input files

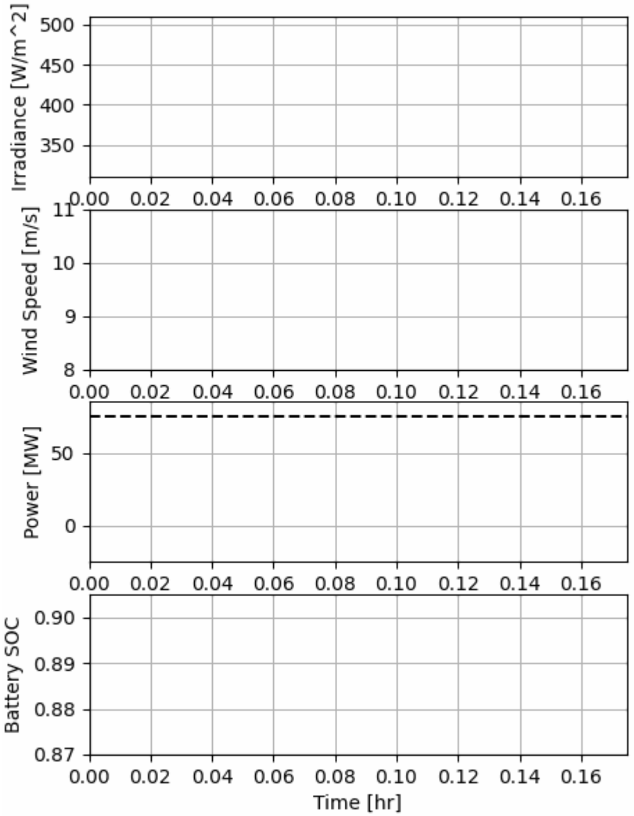
```yaml
name: example_000

###
# Describe this emulator setup
description: Floris, Solar PV and Battery Plant Example

dt: 0.5

hercules_comms:

  amr_wind:

    wind_farm_0:
      type: amr_wind_local #options are amr_wind or amr_wind_local
      amr_wind_input_file: amr_input.inp

  helics:

    config:
        name: hercules # What is the purpose of this name
        use_dash_frontend: False
        KAFKA: False
        KAFKA_topics: EMUV1py
        helics:
            # deltat: 1 # This will be assigned in software
            subscription_topics: [status]
            publication_topics: [control]
            endpoints: []
            helicsport: 32000
        publication_interval: 1
        endpoint_interval: 1
        starttime: 0
        stoptime: 600 # must be at least 2*dt smaller than last timestep in weather file
        Agent: ControlCenter

py_sims:

  solar_farm_0: # The name of py_sim object 1

    py_sim_type: SolarPySAM
    weather_file_name: NonAnnualSimulation-sample_data-interpolated-daytime.csv
    system_info_file_name: 100MW_1axis_pvsamv1.json
    lat: 39.7442
    lon: -105.1778
    elev: 1829
```

```yaml
py_sims:

  solar_farm_0: # The name of py_sim object 1

    py_sim_type: SolarPySAM
    weather_file_name: NonAnnualSimulation-sample_data-interpolated-daytime.csv
    system_info_file_name: 100MW_1axis_pvsamv1.json
    lat: 39.7442
    lon: -105.1778
    elev: 1829

    # capacity: 100 # MW

    initial_conditions:

      power: 25 # MW
      dni: 1000

  battery_0: # The name of py_sim object 1

    py_sim_type: LIB
    size: 20  # MW size of the battery
    energy_capacity: 80 # total capacity of the battery in MWh (4-hour 20 MW battery)
    charge_rate: 20 # charge rate of the battery in MW
    discharge_rate: 20 # discharge rate of the battery in MW
    max_SOC: .9 # upper boundary on battery SOC
    min_SOC: 0.1 # lower boundary on battery SOC

    initial_conditions:
      SOC: 0.88  # initial state of charge of the battery in percentage of total size


controller:
  num_turbines: 10 # Should match AMR-Wind! Ideally, would come from AMR-wind
  wind_capacity_MW: 50 # Should match AMR-Wind! Ideally, would come from AMR-wind
  solar_capacity_MW: 100 # Should match solar system info file TODO: fix so this comes from


external_data_file: plant_power_reference.csv
```

Workflow Demo



Outputs



Irradiance [W/m^2]
500
450
400
350
Wind Speed [m/s]
11
10
9
8
Power [MW]
50
0
Battery SOC
0.90
0.89
0.88
0.87
0.00 0.02 0.04 0.06 0.08 0.10 0.12 0.14 0.16
Time [hr]

Future Development

- Hierarchical hybrid plant controller [collaboration with PNNL]
- Realistic and correlated inflow
- Developments of both high fidelity and medium fidelity wind simulators
- Implementation of further wind farm control approaches (consensus, advanced wake steering)
- Connection to ROSCO turbine-level controller

Wind Farm Analysis and Controls Models

Polls

Open Discussion

Wind Farm Analysis and Controls Models

Raise your hand and we'll call your name to ask a question.

- **Discussion topics**
  - Prospective / new users:
    - What are your thoughts on the learning or onboarding process?
  - Experienced users:
    - What have been your primary pain points or bottlenecks?
    - What has worked or not worked in helping to integrate these software into your workflows?
    - How thoroughly do you understand the capability of these tools?
    - What has helped or hindered your open-source contribution to these software?

Thank you for your time today!

- Need help with a particular problem?
  - GitHub Issues or Discussions pages for any of the models
  - NREL User Forum (for NREL models): forums.nrel.gov

- Have further thoughts that you want to share?
- How could we have done better?
  - Send feedback to Rafael.Mudafort@nrel.gov

- Software repositories:
  - FLORIS: https://github.com/NREL/FLORIS
  - FLASC: https://github.com/NREL/FLASC
  - OpenOA: https://github.com/NREL/OpenOA
  - Hercules: https://github.com/NREL/HERCULES